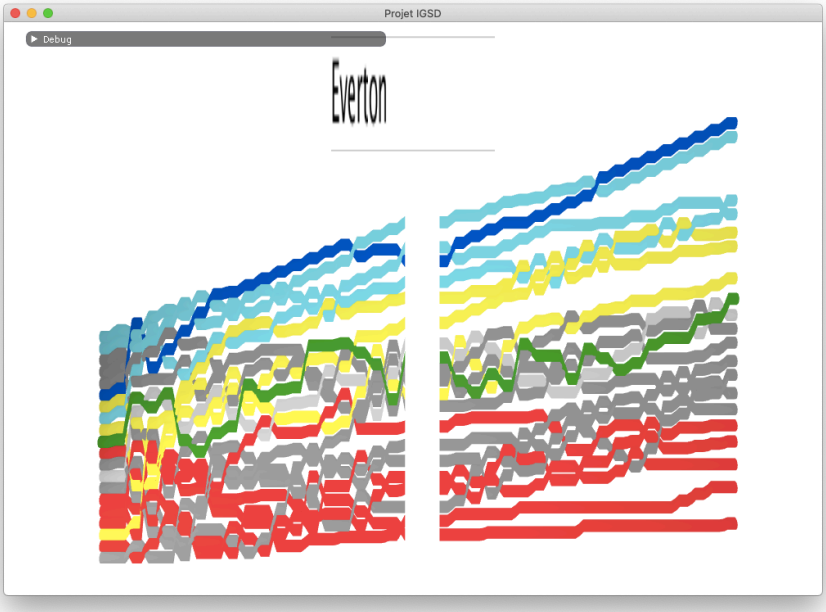
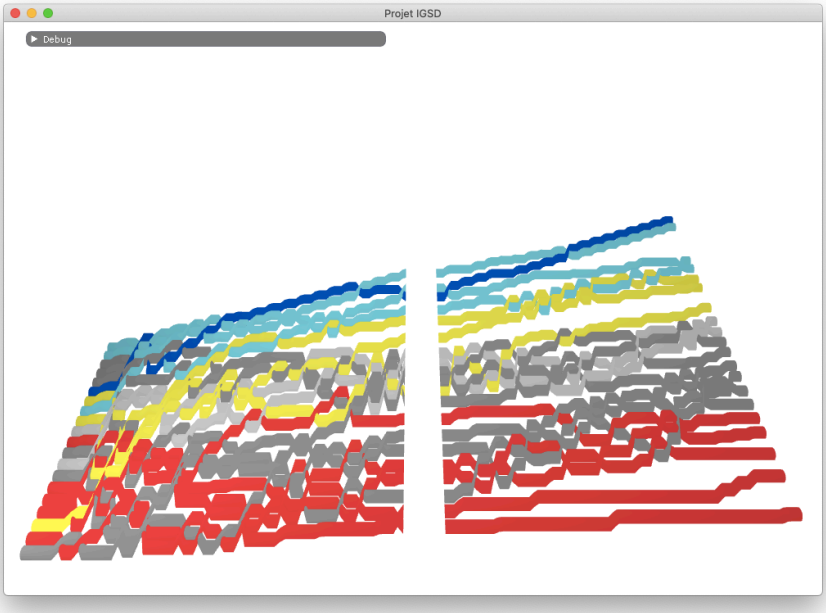
PATAULT Paul

RAPPORT DE PROJET
Informatique Graphique pour la Science des Données





Code source disponible sur GITHUB :
https://github.com/paulpatault/ProjetInfoGraphique

INTRODUCTION

Le projet dont il m'a été confié la réalisation consistait à réaliser un affichage 3D de Gapchart avec OpenGL en C++. Au fur et à mesure de l'avancement de ce travail je suis plusieurs fois reparti de zéro pour des questions de structures de code qui ne me plaisaient plus. Je vais ici vous présenter la version finale en expliquant les choix que j'ai pris pour en arriver là.

Pour résumer, j'ai choisi de séparer au maximum mon code en différentes classes afin qu'il soit plus lisible dans l'ensemble bien que cela complique peut être la première lecture. Les classes sont séparées en deux dossiers : *data*[1] et *screen*. Chaque classe du dossier *screen* traite de tout ce qui sera propre à l'affichage graphique (comme **Camera**[2] ou **Display**). En revanche, chaque classe du dossier *data* traite plutôt les données brutes (comme **Cylinder** ou **LoadData**) avant de les passer à la fenêtre. On repère ces différentes classes dans le fichier *main.cpp* par le namespace *data* et namespace *screen* pour plus de clarté.

PRÉSENTATION DES PRINCIPALES CLASSES

i. data::**LoadData**

Le chargement des données est fait directement dans le constructeur de cette classe par un simple parcourt du fichier en prenant seulement le rank et les points pour chaque jour pour chaque équipe. On établira à partir de ces données l'ordonnée de chaque cylindre pour chaque jour.

ii. data::**Cylinder**

Une instance de cette classe sera attribué à chaque équipe dans un vector. Elle se charge de construire les cylindres de chaque équipe pour tous les jours du championnat.
La méthode de création de cylindre est la suivante : d'abord, on fabrique un tableau des y pour tous les jours pour chaque équipe à partir de **LoadData** ; ensuite on construit à partir de cette liste un tableau des points qui dessinent un rectangle plat (modélisant le fond du cylindre) auquel on ajoute des demi-cercles qui seront posés par dessus cette face plate ; enfin, on relit chaque points de chaque demi-cercle afin de former la totalité du demi cylindre.
C'est aussi à partir de cette classe que sont créés les normales associées à chaque point de chaque face.

iii. data::**Shader**

Un **Shader** possède un ID et de nombreuses méthodes permettant de transmettre aux **.glsl* n'importe quel type de variable.

---

[1] Tous les noms de dossiers et fichiers seront en caractère italique.

[2] Tous les noms de classes seront en caractère gras.

iv. screen::**Render**

Classe qui gère la fenêtre ainsi que l'initialisation du contexte d'OpenGL.

v. screen::**Display**

Classe qui gère l'affichage en direct de la fenêtre et une partie de la sélection clavier. On peut ici gérer la sélection de l'équipe: 0 pour l'équipe arrivée en 1ère place, 1 pour la seconde, etc. , et pour les dizaines il faut maintenir 'x' tout en pressant les précédentes touches.
Exemples : • maj + 1 := équipe rang 1 := 2ème équipe
• maj + x + 3 := 10 + 3 := équipe rang n°13 := 14ème équipe

Une équipe sélectionnée verra alors son nom affiché en haut de la fenêtre et elle sera mise au premier plan (plus de passage par devant/derrière en fonction du gain/perte de rang) et sera affichée en vert (cf. Annexe 1).

vi. screen::**Camera**

Implémentation d'une caméra qui peut se déplacer dans toutes les directions dans la fenêtre. Il s'agit de commandes basiques : ZQSD pour avant, arrière, gauche et droite. Mais on a aussi la possibilité de « tourner la tête » de haut en bas avec les touches flèche du haut / flèche du bas, de même de droite à gauche avec les flèches correspondantes du clavier. Enfin, on peut accélérer la vitesse de déplacement en pressant « entrer » et la réduire en pressant « supprimer » (DEL).
La dernière touche à connaître est TAB, elle permet de revenir à la position initiale de la caméra.

vii. screen::**MVP**

La matrice « **Modèle Vue Projection** » (avec une touche personnelle, séparant la rotation de la matrice de modèle, ce qui donne alors en réalité MRVP). On utilisera les touches IJKL du clavier (disposées comme ZQSD) pour faire tourner sur lui même à sa guise le modèle (cf. Annexe 2)

viii. screen::**c_ImGui**

Une classe utilitaire plutôt utilisée pour le débugage. Cet outil permet de modifier en direct (à la main!) les valeurs de variables quelconques (float, glm::vec3, glm::mat4, etc.) à partir de la fenêtre.
On verra apparaitre cette micro-fenêtre, très pratique dans notre contexte (cf. Annexe 3).

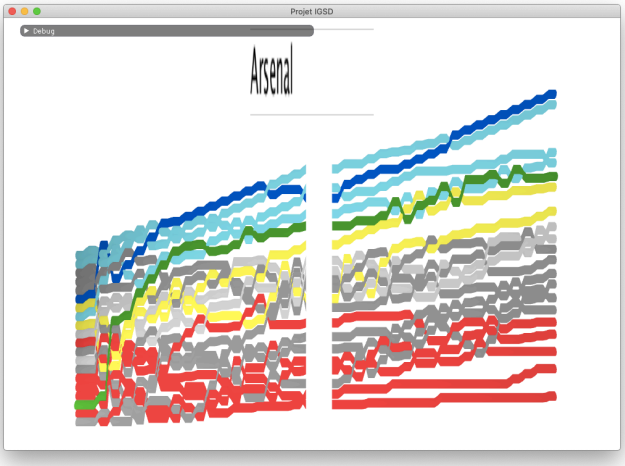
CONCLUSION

Je n'ai pas rencontré de difficultés majeures dans la réalisation de ce projet, seulement d'importants ralentissements. Cela m'a ainsi empêché de traiter certains points, par manque de temps. Je regrette, par exemple, de ne pas avoir pu exactement implémenter les textures comme cela était proposé dans le sujet. J'ai cependant beaucoup apprécié travailler sur ce projet ; je pense, par ailleurs, en poursuivre la réalisation après les échéances de rendu.

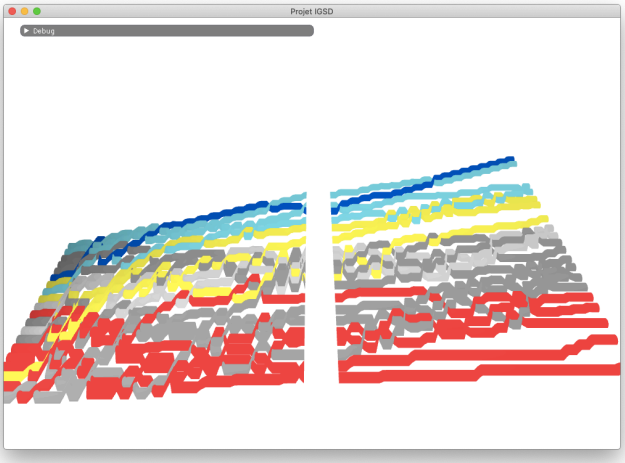
PATAULT Paul

RAPPORT DE PROJET
— Annexe

Annexe 1 : *Sélection de l'équipe Arsenal (n°4, cinquième au classement)*



Annexe 2 : *Vue possible grâce à des rotations (MVP)*



Annexe 3 : *Contexte ImGui*