Drapeau Flottant

Rapport de projet

Réalisée par :

BEN HAMOUDA Amel

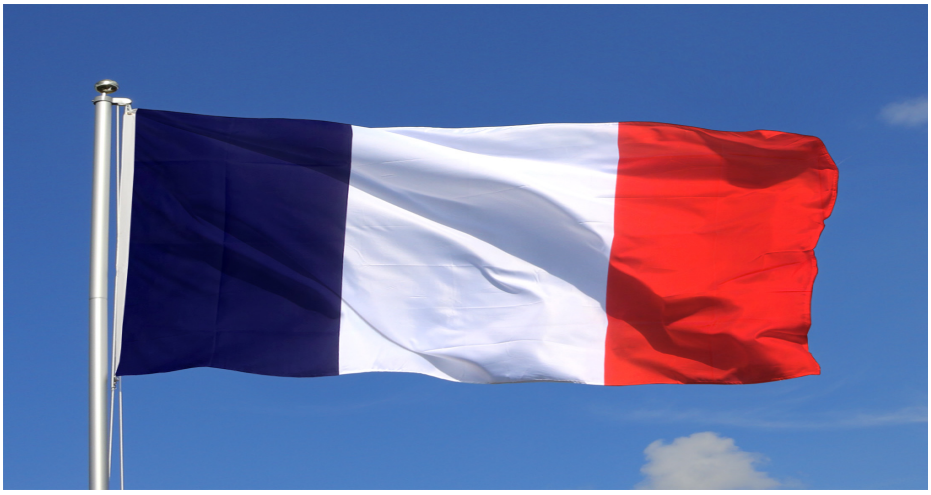
Université Gustave Eiffel
Master 2 Science de l'image
2019/2020

Table des matières

I. Description du projet....3
II. Mode d’emploi....3
III. Diagramme des classes....4
IV. Description de l’architecture....5
1. Drapeau....5
2. Formes....6
3. Mécanique....6
V. Conclusion....8
VI. Annexes....8
1. Dépendances....8
2. Instructions de compilation....8
3. Documentations / Instructions d'utilisation....8

I. Description du projet

Le projet consiste à faire sur la basse des systèmes masse-ressort un modelé du drapeau flottant dans le vent. Dans ce projet on a donc besoin de plusieurs objets tels que des cubes et une sphère.



Ce projet est programmé en C++ à l'aide des librairies GLEW et GLUT (freeGlut) d'OpenGL.

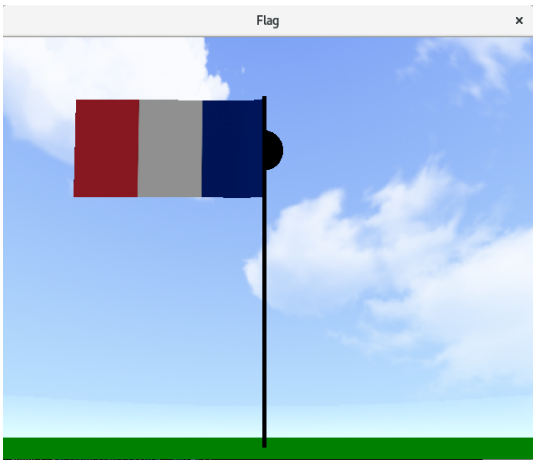
Ce rapport a pour but d'expliquer à l'utilisateur le mode de fonctionnement du projet et de décrire l'architecture qui a était développé pour le projet.

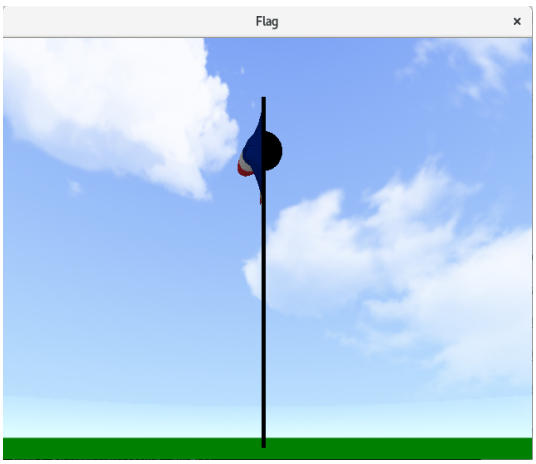
II. Mode d'emploi

Au lancement du programme, on obtient ceci sur le terminal :

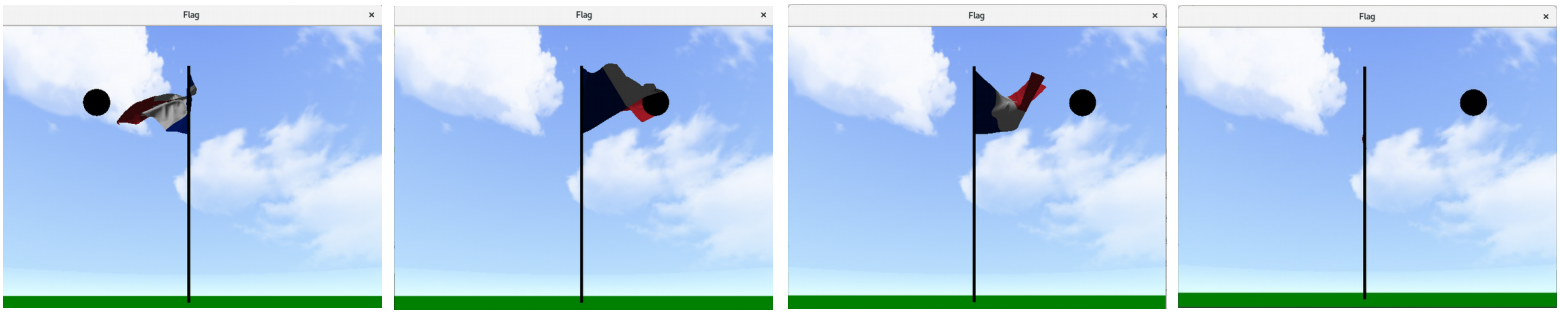
```
amel@debian:~/Documents/M2_Image/Option_AnimationSimulation/PROJET/Projet-BEN_HAMOUDA-Amel$ ./Flag
Choisir quel drapeau vous souhaitez :
   1. France ,
   2. Canada,
   3. Tunisie,
   4. Nouvelle Zélande,
   5. Etat-Unis
```

Il faut donc choisir quel drapeau on souhaite faire flotter. Une fois le choix effectué une fenêtre apparaît :





On peut voir, ci-dessus, sur l'image de gauche la fenêtre à l'exécution du programme et sur l'image de droite la fenêtre après exécution de quelque seconde sans rien toucher (aucun mouvements).
Le "trait" noir représente le porte drapeau et la sphère noir permet l'illustration du vent.



On peut voir sur les images ci-dessus une successions de mouvements de la balle qui permet de faire flotter le drapeau. On peut remarquer que sur la dernière image on ne voit plus le drapeau puisque la balle a cessé de bouger depuis un moment.

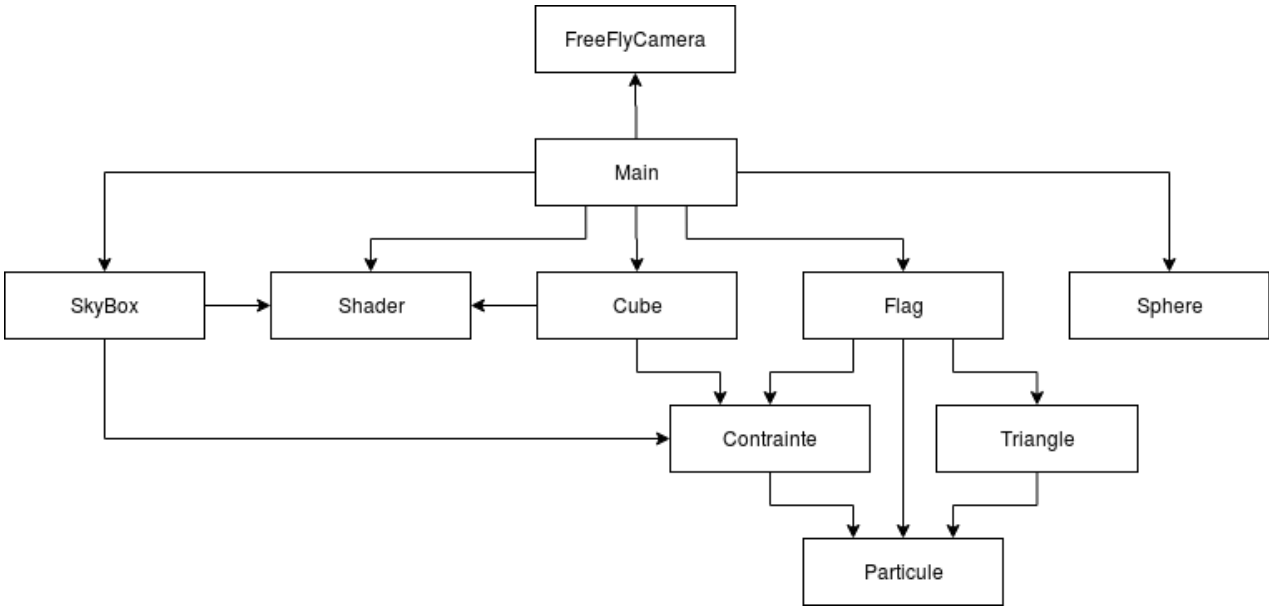
Les commandes pour gérer les mouvements sont les suivantes :

Drapeau (Balle)	
Boutons	**Actions**
s	vers le bas
z	vers le haut
d	vers le droite
q	vers le gauche
f	en profondeur vers le bas
r	en profondeur vers le haut

Caméra	
Boutons	**Actions**
↓	vers le bas
↑	vers le haut
→	vers le droite
←	vers le gauche

III. Diagramme des classes

Le schéma suivant montre l'inclusion de chaque classe :



La majorité des classes utilisent les librairies 3D.

IV. Description de l’architecture

Cette partie permet de rentrer plus en détail dans l’implémentation du programme.
L’explication du programme sera divisée en trois parties. La première pour tout ce qui porte au drapeau, la seconde aux objets crées pour mener à bien ce projet et la dernière pour toute une partie que l’on pourrait appeler la mécanique ou la technique du projet avec la caméra et les shaders.

1. Drapeau

Cette partie explique comment a était implémenté le drapeau.

A – Particule

Cette classe permet la création des particules (points) du drapeau. Elle est constituée de plusieurs méthodes :

→ un constructeur qui prend en paramètre la position, la masse, l’amortissement et les coordonnées de texture d’un point précis dans le drapeau,

→ **move()** qui gère le mouvement d’une particule en vérifiant en amont si cette particule est static ou pas. Si celle-ci est static rien ne se passe sur cette particule, par contre à l’inverse on modifiera sa position.

→ **step()** qui gère / modifie la force sur le mouvement de la particule du drapeau, puisque par défaut la force est à 0 au début.

B – Flag

Cette classe permet la création des particules (points) du drapeau. Elle est constituée de plusieurs méthodes :

→ un constructeur qui prend en paramètre la largeur et hauteur du drapeau, la largeur et hauteur d’une particule, le poids, l’amortissement et la transformation du drapeau.
Le constructeur crée les particules nécessaires ainsi que les triangles de maillage pour construire le drapeau. Il définit aussi certaines contraintes entre les particules et il fixe les particules qui seront static, ces dernières ne bougeront pas quand la balle noir (sphère) entrera en collision avec le drapeau.

→ **loadTexture()** qui charge la texture qui sera appliquée au drapeau,

→ **display()** qui construit le drapeau avec tout ce qui a été crée en amont (particules, maillages,...),

→ **displayContraintes()** qui effectue la même chose que **display()** mais applique les contraintes en plus,

→ **step()** qui gère la gravité et la force sur le mouvement du drapeau, en fonction des contraintes.

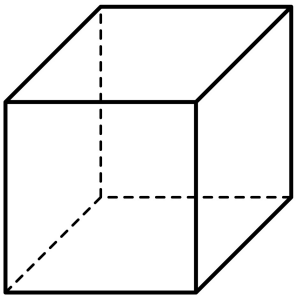
2. Formes

Cette partie explique comment ont été implémenté les différentes formes du projet.

A – Cube

Cette classe permet la création de la skybox et du porte drapeau. Elle est constituée de plusieurs méthodes :

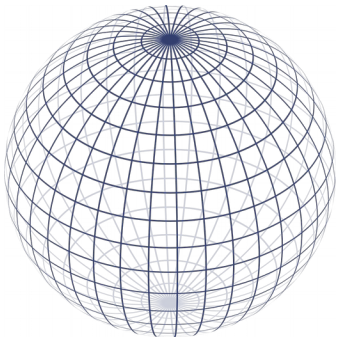
→ un premier constructeur qui prend en paramètre la position, la taille et la couleur du cube,
→ un second constructeur qui prend en paramètre la taille du côté du cube,
→ **display()** qui construit un cube avec la librairie GLUT,
→ **build()** qui construit le cube en OpenGL à l'aide de la taille du cube,
→ **gerDataPointer()** qui renvoie les données des vertex du cube,
→ **getVertexCount()** qui renvoie le nombre de vertex.



La différence entre les méthodes **display()** et **build()** est que la première méthode ne permet par l'application de texture sur le cube alors que la seconde méthode le permet.
C'est pour cela que pour la création de la skybox on utilise la méthode **build()** alors que pour le porte drapeau on utilise **display()**.

B – Sphère

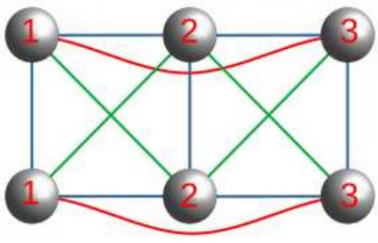
Cette classe permet la création de la balle qui simule la force du vent. Elle est constituée de plusieurs méthodes :



→ un constructeur qui prend en paramètre la position, le rayon et la couleur de la sphère,
→ **display()** qui construit une sphère avec la librairie GLUT,
→ **collision()** qui gère la collision de la sphère avec les particules du drapeau,
→ **move()** qui gère le déplacement de la sphère en fonction des touches du clavier.

C – Triangle

Cette classe permet la création des maillages triangulaire entre les particules du drapeau. Elle est constituée de plusieurs méthodes :



→ un constructeur qui prend en paramètre trois particules (points) du maillage.
→ **display()** qui crée le maillage entre ces trois particules.

3. Mécanique

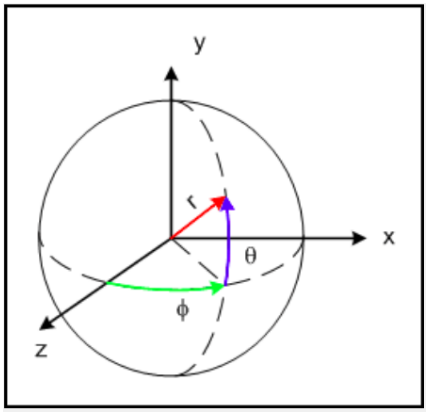
A – Shader

Cette classe gère le chargement des vertex et fragmente shaders, ainsi que le stockage des variables de ces shaders.
Pour cela à la construction d'un shader, le constructeur effectue directement le chargement et le stockage des données associées.

Ensuite il y a la fonction **use()** qui permet d'installer un objet programmé dans le cadre de l'état du rendu actuel du shader. Cette fonction est appelée lorsque l'on souhaite utiliser un shader.
On retrouve aussi dans cette classe des méthodes qui mettent à jour les différentes matrices des shaders tels que **setModelMatrix()**, **setViewMatrix()** et **setProjectionMatrix()**.

B – FreeFlyCamera

Cette classe gère la caméra avec un modèleFreefly qui permet ce déplacement librement dans la scène.
Ce type de caméra est utilisée dans les jeux à la première personne (les jeux où on voit à travers les yeux du héros, comme les FPS). Elle permet de se déplacer dans toutes les directions.
Sa ViewMatrix peut être calculé par un simple appel à la fonction **glm::lookAt(eye, point, up)**. Le premier argument est simplement la position de la caméra exprimée dans le monde. Le deuxième argument est un point que la caméra regarde. Le dernier argument est l'axe vertical de la caméra.



Les arguments **point** et **up** doivent être calculés en passant par les coordonnées sphériques, illustrées ci dessous.
On a donc un ensemble de fonction qui permet d'implémenter ce modèle :

→ un constructeur où on initialise la position de la caméra, **m_fPhi** à 0, **m_fTheta** à -π /2. Dans le corps du constructeur on fait appel à la méthode **computeDirectionVectors** afin que les vecteurs directionnels soit correctement initialisés également.

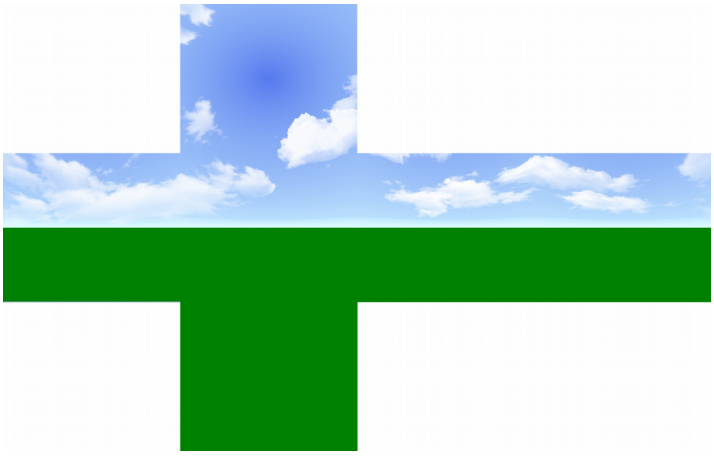
→ **computeDirectionVectors()** qui calcule les vecteurs **m_FrontVector**, **m_LeftVector** et **m_UpVector** à partir des coordonnées sphériques **m_fPhi** et **m_fTheta**.

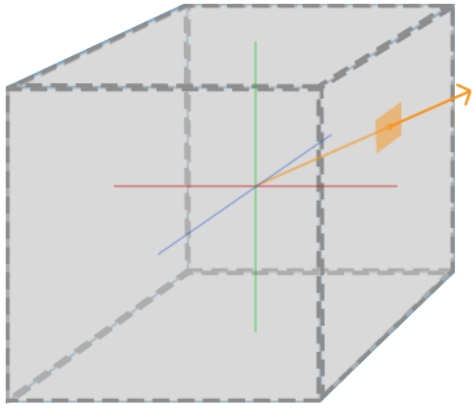
→ **moveLeft(float t)** et **moveFront(float t)**, ces méthodes sert à déplacer la position de la caméra respectivement le long des vecteurs.

→ **getViewMatrix()** qui calcule et renvoi la matrice View associée à la caméra. Comme dis plus haut, il suffit d'appeler **glm::lookAt** avec les bon arguments.

C – SkyBox

Une skybox est un cube qui englobe l'intégralité de la scène et contient six images de l'environnement l'englobant.
Du coup on doit avoir 6 images pour chaque faces du cube pour bien avoir une illusion de la skybox. La texture utilisée pour ce projet est la suivante :





Une fois cette image divisée en 6 "imagettes", on obtient des images de taille 512x512 qui sont de bonne qualité pour un meilleur rendu visuelle de l'illusion.

Pour charger la skybox , il suffit d'utiliser la fonction **loadCubemap()** qui permet le chargement des images de la skybox.
Et pour l'affichage de cette skybox on a besoin des shaders associés et de la création de nouveau Vertex Array Object (VAO) et Vertex Buffer Object (VBO), ainsi que de récupérer la ViewMatrix de la caméra.

V. Conclusion

Ce projet a permis l'application des connaissances vues en cours et d'approfondir les connaissances liées à la programmation C++ avec les librairies de la 3D (OpenGl, Glut, Glew).
Le modèle obtenu dont les éléments principaux (notamment les particules du drapeau, et quelques objets permettant la flottaison) ont été modélisés.

La principale difficulté de ce projet a été le calcul des particules du drapeau avec leurs maillages pour permettre la flottaison du drapeau.

On pourrait penser (en tant que projet à plus long terme) à coder le modèle du vent sans utiliser d'objet permettant d'avoir la force du vent grâce à la collision de la sphère avec les particules du drapeau.

VI. Annexes

1. Dépendances

Nécessite la prise en charge d'OpenGL et les bibliothèques GLEW et GLUT (paquets libglew-dev et freeglut3-dev dans Debian).

→ " GLUT " https://www.opengl.org/resources/libraries/glut/
→ " GLEW " http://glew.sourceforge.net/

2. Instructions de compilation

Pour compiler le programme avec les modules il suffit de lancer le Makefile :

mkdir bin *(si nécessaire)*
make

Pour l'exécution il suffit de taper :

./Flag

3. Documentations / Instructions d'utilisation

Pour lancer le programme il suffit de taper *./Flag* dans le terminal.
Ensuite pour la suite, voir Mode d'emploi et le README.md.