Moteur de jeu

TP : Graphe de scene

par Maxime Isnel

lien du Git de Green Engine : https://github.com/TheSpyGeek/Green-Engine

Compiler le code

Sur Linux

Dependencies :

```
sudo apt-get install -y build-essential cmake xorg-dev libgl1-mesa-dev libfreetype6-dev
```

Pour compiler :

```
cd build && cmake .. && make
```

Sur Windows

Dependencies :

- mingw 64 ou mingw 32
- cmake

Pour compiler :

- Lancer `Cmake`
- Configurer avec les `Mingw Makefile` pour compiler dans le dossier `build`
- Lancer `mingw64`
- Aller dans le dossier `build`
- Compiler avec `mingw32-make`

Pour lancer le programme :

```
./green-engine
```

Structure du code

Les grandes classes du programme sont :

- **MainRenderer** : il fait le rendu d'une scène dans une texture et puis l'affiche dans une texture
- **InputManager** : gère les entrées clavier et les associent avec des fonctions du moteur
- **UI** : gère tout l'affichage des w idgets sur l'écran (hors rendu). Le méthode d'affichage de chaque composant d'un objet sont défini dans les objet eux mêmes
- **Scene** : gère tous les objets du moteur (ajout, suppression, mise à jour). Elle possède une liste d'**ObjectEngine**
- **ObjectEngine** : Est un objet géré par la scène. Il possède un **Transform** ce qui permet de le déplacer. Il possède une liste d'**ObjectEngine** qui sont ses fils et dont les positions et rotations vont dépendre de la matrice model du parent.
- **MeshObject** : Il possède un maillage donc ils sont affichable. Il possède un **Material** qui va faire le shading et la couleur de l'affichage de cette objet.

Graphe de scène

Pour faire mon graphe de scène, j'ai d'abord créé une classe **Transform** qui me permet de controller la position, la rotation, le scale ce qui me donne la matrice model pour pouvoir afficher cette objet. Je peux aussi dissocié la matrice model qui me permet d'afficher l'objet et celle que je vais utiliser avec les objets enfants de cet objet.

Pour dissocier ces deux matrices il faut aller sur l'interface => cliquer sur un objet dans le scène manager => "Model matrix to child" => Uncheck "Same matrix as parent". Vous pouvez donc controller la position, la rotation indépendement.

Vous pouvez aussi animer la rotation de chaque objet dans la partie **Animation** (il y en une pour l'affichage de l'objet et une pour la matrice envoyée aux enfants). Vous pouvez cocher l'axe ou les axes que vous voulez faire tourner et à quelle vitesse. Vous pouvez "reset" l'animation en appuyant sur le bouton reset

Pour vous assurer que la Terre tourne sur son axe et en étant incliné par rapport au soleil, vous pouvez activé les **WireFrame** (`View => Toggle wire frame`).

Transform classe

dans le fichier `src/models/Transform.cpp`

Dans la classe Transform je stocke les vecteurs de position, rotation et scale, pour afficher l'objet mais aussi pour la matrice envoyée aux enfants. Il y a aussi toute les variables nécéssaires à l'animation.

Fonctionnalités

Vous pouvez :

- Vous pouvez redimensionner la fenêtre de l'application et la fenêtre du **scene manager**
- Ouvrir un arbe dans le **scene manager** si l'objet à des fils en clique sur la flèche
- Ajouter des objets avec maillage en faisant `Edit => Add MeshObject` . Vous pouvez ensuite changer le maillage chargé en changeant le chemin du fichier .OFF et en appuyant sur le bouton "Recreate"
- Afficher l'objet avec des `WireFrame` en faisant `View => Toggle wire frame`
- Faire tourner les objets, les déplacer, …
- Avec `CTRL + H` vous pouvez cacher/montrer l'affichage des fenêtre.
- Avec `CTRL + P` pause et play l'animation