HAI918I
Projet d'3D

Moteur de Jeu

Melvin Bardin
Laurine Jaffret

Encadré par Mme. Faraj

2021-2022

UNIVERSITÉ
DE MONTPELLIER

FACULTÉ DES SCIENCES
MONTPELLIER

Table des matières

1 Contributeurs

Projet réalisé par Melvin Bardin et Laurine Jaffret sous la directive de Noura Faraj .

1.1 Lien du Github et vidéo youtube

Github : https ://github.com/Kyrial/Moteur-de-jeu-jeu

Vidéo youtube : https ://youtu.be/InkqABgvRUg

2 Introduction

Les moteurs de jeu sont des ensembles de composants logiciels effectuant des calculs de géométrie et de physique. Ces derniers sont devenues indispensable à la réalisation de jeux vidéo car ils permettent aux équipes de développement de se concentrer sur le contenu du jeu plutôt que réinventer la roue à chaque nouveau projet.
Nous nous somme donc consacré à la réalisation d'un moteur de jeu ainsi qu'un jeu afin de mieux comprendre leurs fonctionnements

3 contenu du projet

Notre projet contient un terrain avec un joueur et du décors. Nous avons implémenté toute la pipeline d'OpenGL (vertex, control, evaluation, geometry, fragment). la tesselation étant implémenté, nous avons pu augmenté drastiquement la qualité des maillage proche du joueur et la réduire progressivement avec la distance. L'herbe apporte une touche de réalisme avec leurs mouvement sinusoïdale.

Nous avons aussi implémenté des Animations pour les arbres, l'herbe, les nuages et l'eau,

Le terrain, les nuage, les arbre et tout ce qui a besoin de pseudo aléatoire est généré à l'aide d'un bruit de Perlin. Le terrain est infinie et celui ci se centre automatiquement sur le joueur. Idem pour les arbre mais avec une portée moins élevé car les arbre prennent beaucoup de ressource s'ils sont trop nombreux. Quant aux nuages, la caméra et le soleil, ils sont fixé au joueur (en enfant du joueur dans le graphe de scène). Leurs coordonnées n'ont donc pas besoin d'être mis à jour, ils suivent automatiquement le joueur.

Pour les collision, sont géré : AABB-terrain, AABB-AABB et AABB-AABB instancié. N'a pas été implémenté la collision entre instance. L'objet heurtant est réfléchie, l'objet heurté est poussé s'il est mobile. Une hiérarchie de AABB englobe chaque objet et les enfant s'ils contienne un colliders. Cela permet de recherché efficacement dans l'arborescence si une collision a lieux

3.1 Choix technologique

Nous avons utiliser le code source de base fournit par Mme Faraj en C++ ainsi que la librairie QtOpenGl. Nous avons utiliser la version de glsl 410 afin de pouvoir utiliser des fonctionnalité récente.

4 structure du moteur

4.1 Les différentes classes et leurs utilités

4.1.1 BasicIO

librairie fournie afin de pouvoir lire un fichier. a été modifié afin de pouvoir lire un fichiers "off" et des fichier "obj" tout en récupérant les coordonnée de textures.

4.1.2 GeometryEngine, GeometryMeshEngine, GeometryUI

geometryEngine est la classe mère et les deux autres les enfants. Ces groupement d'objet on été conçus afin d'effectuer toute les actions en rapport aux maillages :

- GeometryEngine
  - gestion et allocation memoire des maillages
  - effectuer les opérations des différentes boites englobantes
  - gestion des collisions
  - subdivision en triangle d'un plan (pour le terrain/le ciel)
  - recallage du terrain lorsque le joueur se déplace
  - instenciation des maillages
- GeometryMeshEngine
  - récupération des maillages lu par BasicIO
- GeometryUI
  - bare de vie

4.1.3 Object, GameObject, MobileObject, CameraObject, BillboardObject

ce groupement d'objet on été conçus afin de contenir tout les différents objet de jeu du projet. Object est la classe mère et est abstraite, elle contient les comportement par défaut. Les autres classes sont ses filles et spécifient les comportement selon leurs besoins.
GameObject contient les objets qui ne subisse pas la gravité, comme le terrain, le soleil, les arbres etc ...
MobileObject contient les objets subissant la gravité, on peut y trouver le joueur, la boite.
BillboardObject devait etre à la base capable d'afficher des billboard, mais par manque de temps, elle est capable de contenir uniquement la barre de vie.

- Object
  - boucle de rendu
    - calculs des transformations
    - bind des différents parametres
    - mise à jour AABB des objets
    - mise à jour AABB des noeuds
  - Calculs transfert de force

4.1.4 La suite dans Doxygen

Nous avons eu l'idée de faire une documentation Doxygen, donc nous nous arrêtons là pour l'explication des classes pour évité une duplication des informations.

4.2 Graphe de scène

Le graphe de scène est entièrement conçu dans la fonction ***"scene()"*** de la classe **MainWidget**.

```
//divers objet mobile
GeometryEngine *geo_decors1 = new GeometryMeshEngine;
geo_decors1->withNormal = true;
geo_decors1->withTextureCoord =true;
geo_decors1->initMeshObj(":/Mesh/Cube_obj.obj");
Transform *t_decors1 = new Transform;
t_decors1->setScale(0.005,0.005,0.005);
t_decors1->setTranslate(60,60,60);
Transform *anim_decors1 = new Transform;
anim_decors1->setTranslate(0,0,2);
Object* decors1 =addMobileObject(allShaders[1],Terre, t_decors1,   geo_decors1, anim_decors1,
                        new QOpenGLTexture(QImage(":/Texture/Cube_diffuse.jpg").mirrored()));
decors1->transfertForce = 1;
```

FIGURE 1 – Création d'un objet mobile

Prenons comme Exemple la figure 1, il suffit d'appeler la fonction "addMobileObject()" avec un certain nombre de paramètre et celle ci va créer et initialiser les données. les seules paramètre sont : le shader associé, le parents, la transformation, le geometryEngine (gestionnaire des maillages), la tranformation pour l'animation, et une eventuel texture.
Chacun de ses objects a un certain nombre de paramètres. Par exemple "withNormal" indique de prendre en compte les normal du maillage associé, et "withTextureCoord" indique que lors de la lecture du fichier Obj, les coordonnée de texture doivent être récupérer.

4.2.1 boucle de rendu

La boucle de rendu se traduit par un simple appel récursif de updateScene(). Chaque objet du graphe de scène va effectuer son traitement associé a sa classe et appeler la méthode de la super-classe.
Le traitement spécifique des sous classe correspond à :

- cameraObject
  - calcul de la transformation
  - lookat()
- mobileObject
  - calcul gravité et perte cinétique
  - test de collision
  - calcul de la transformation

Table des figures