Projet Programmation temps réel n°1

Programmation temps-réel

—IMAC troisième année—

Un jeu temps réel

Ce projet a pour but de vous faire coder un jeu tournant "en temps réel" c'est à dire à 30 images par seconde. Il vous permettera d'acquérir une bonne pratique des difficultés : de la programmation temps réel, de la programmation en équipe.

---

1 Introduction

Le projet consiste en l'élaboration d'un jeu multijoueur temps réel type FPS en décor d'extérieur. Le jeu devra tourner sur les machines de la faculté donc sous linux avec OpenGL. Le codage de ce jeu a été décomposé en trois parties distinctes mais interdépendantes : environnement, animation et IHM/réseau. Pour réaliser ces parties, toutes librairies pourra être utilisée si elle est installable sur la machine cible (les machines de la fac). Il est conseillé d'utiliser, pour les trois parties, d'une librairie mathématique commune comme la librairie VBVector3D disponible sur `igm.univ-mlv.fr/~biri/`. La répartition des étudiants dans les groupes sera la suivante :

Environnement :

- Rodolphe MOYSE
- Loic DE LA TULLAYE
- Jean-Baptiste WALCZAK
- Eva GABRIEL
- Pierre PRIMEN
- Florent PAULAIS

Animation :

- Elya RANDRIANAIVO
- Simon LIEUTAUD

- Camille PAPILLIER
- Pablo HUE
- Fabien BENARD
- Jean-Baptiste VERDIER

IHM/Réseau :

- Yannick DUPUIS
- Laurent BARBAT
- Thierry GERBEAU
- Raphael LOYET
- Magali MOTARD
- Franck TRAN
- Emilie DARDEVET

Chaque groupe nommera un responsable de module. Le groupe IHM/réseau est en charge de la cohérence du projet. Ainsi, le chef de ce groupe sera le chef du projet en son entier. De plus, deux membres de ce groupe seront responsables de la communication avec les deux autres modules. Les travaux a effectuer pour chacune de ces trois parties sont explicités dans les sections suivantes...

2 Partie environnement

Cette partie permettera au moteur de rendu d'afficher tout l'environnement du jeu à savoir : le terrain, le décor et tout élément mobile autre que les joueurs. Les éléments du jeu sont en effet séparés en trois catégories : le terrain, les modèles qui sont les éléments de décor du jeu mais qui contiennent également les joueurs (mobiles donc) et enfin les particules qui représentent tous les éléments mobiles du jeu hormis les joueurs (comme des particules d'explosion, des missiles, des tirs...).

Le module environnement devra :

1. charger une carte de hauteur représentant le terrain.
2. réaliser l'ombrage (et l'auto ombrage) de ce terrain.
3. texturer le terrain.
4. afficher de manière optimale les éléments du décor.
5. Générations et gestions des particules.

6. gérer les trajectoires et déplacements des particules et des joueurs (le déplacement du joueur est géré par l'IHM mais sa trajectoire dépend de l'environnement) et donc de gérer les problèmes de collision entre modèles et terrain, modèles et modèles, et enfin entre modèles et particules.

Afin de passer à l'implémentation de ces modules, il est nécessaire de réaliser une étude préalable des données (notamment du fichier de données d'entrée) et des spécifications de chacune de ces parties.

2.1 Chargement d'une carte de hauteur

La carte de hauter est en réalité une image en niveau de gris de type PPM. Chaque pixel représente un point de la carte et son niveau de gris représente sa hauteur (0 pour la hauteur minimale, 255 pour la hauteur maximale). Un fichier texte annexe (de suffixe .scn) représentant la scène décrira notamment le nom du fichier image utilisé pour la carte de hauteur, la distance entre deux pixels voisin en abscice et en ordonnée, la hauteur minimal et maximale. Un exemple est donné dans la section 2.7 mais vous avez toute latitude pour modifier ce format.

Bonus : Vous pouvez rajouter à la description de la scène, une modélisation du ciel sous la forme d'un dome ou d'un cube texturé. Dans ce cas, vous pouvez rajouter aux fichiers de format `.scn` le type de ciel ainsi que les fichiers de texture associés.

2.2 Ombrage du terrain

Le terrain sera éclairé par une lumière directionnelle représentant la lumière du soleil (ou d'une lune ou de toute autre astre de votre choix). Cette direction d'éclairement pourra être indiquée dans le fichier `.scn` (cf. section 2.7). Pour ajouter à votre terrain des ombres réalistes, vous utiliserez un algorithme dit de shadow mapping. Pour ce faire, vous placerez à un endroit approprié une camera OpenGL (avec une projection appropriée) afin de capter une image de la profondeur des élements relativement à la source de lumière.

Lors du rendu du terrain, tout sommet S à afficher devra être transformé dans le repère de la source de lumière (et donc dans celui de la carte de profondeur ci dessus). Il faudra alors projeter S sur la carte de profondeur afin de comparer la profondeur p_S du sommet à celle p_I contenue dans l'image de profondeur. Si $p_S > p_I$ alors le sommet est dans l'ombre.

2.3 Texture du terrain

Vous devrez également appliquer une texture à votre terrain. Pour ce faire, vous pouvez modifier le fichier `.scn` (cf. section 2.7) afin d'introduire les fichiers de texture utilisés. Vous avez toute latitude pour cette partie.

2.4 Affichage de l'environnement

En plus du terrain, l'environnement contient un certain nombre de décors qui sont placés dans le fichier `.scn` (cf. 2.7). Ces éléments de décors représentent par exemple des batiments,

des arbres et tout autre objets statiques. Notez que les éléments du décor (batiments, arbres...) pourront être des éléments 3DS. Chaque élément de décor est un objet dans son repère "canonique" et qui sera transformé par l'application à sa place finale grâce à une transformation indiquée dans le fichier `.scn`. Par contre et dans un but de simplification, les joueurs ne pourront en aucun cas rentrer dans les décors ni voir à l'intérieur.

A l'aide d'une représentation en quadtree ou en octree de la scène (soit terrain, modèles et particules), vous implémenterez un algorithme de *frustum culling* qui permet de n'afficher que les éléments potentiellement visible de la scène. Les éventuels paramètres du quadtree pourront être insérés dans le fichier `.scn`.

Important : Pour le chargement de modèles complexes 3DS, il peut être indispensable d'avoir une sortie "propre" des fichiers 3DS. Cela ne peut se faire que grâce à un plugins à 3DS Max nommé Polytrans3D dont une version est disponible auprès de moi.

Bonus : L'application pourra éventuellement gérer des LOD (Level Of Details) pour tous les modèles de la scène. Le choix de l'algorithme du LOD est laissé à votre discrétion.

2.5 Génération et gestion de particules

Tous les tirs et les missiles du jeu seront considérés comme des particules. Leur trajectoire suivront les lois classiques de la dynamique (des solides). Lors de la collision d'un tir ou d'un missile avec un élément du décor, un certain nombre de particules représentant une explosion seront générées. Ces particules explosives seront de deux types : particules de feu, surtout utiles pour représenter le feu issu de l'explosion, et des particules de débris, simples triangles ou quads tournoyant, et représentant les débris issus de l'explosion. Ces particules explosives auront une certaine durée de vie mais ne participeront pas aux collisions (c'est la seule exception à la gestion des collisions).

2.6 Gestion des déplacements et collisions

Le module devra être capable de calculer la trajectoire des particules et de gérer les interactions entre le mouvement des joueurs et le terrain ainsi que les collisions pouvant intervenir entre tout élément mobiles et tout élément du jeu. Pour ce faire, chaque élément du jeu, à l'exception du terrain, sera muni d'une *boite englobante* de votre choix permettant de connaitre l'intersection ou non. Ainsi, et dans un but de simplification, les intersections ne se feront qu'entre boites englobantes ou entre boite englobante et le terrain.

2.7 Le fichier de description de scène : un exemple

Voici un exemple de ce à quoi peut ressembler le fichier de description de la scène. Vous pouvez ou non insérer la gestion des commentaires.

```
# J'aime bien pouvoir mettre des commentaires dans des fichiers
#   mais ce n'est pas obligatoire
/home/biri/Projet/IMAC3/Donnees/carte1.ppm
1 1
```

```
0 12.0
# Suit la description du ciel de l'environnement
sky 1
/home/biri/Projet/IMAC3/Donnees/sky1.jpg
/home/biri/Projet/IMAC3/Donnees/sky2.jpg
# Textures du terrain
/home/biri/Projet/IMAC3/Donnees/terrain1.ppm
/home/biri/Projet/IMAC3/Donnees/terrain2.jpg
/home/biri/Projet/IMAC3/Donnees/terrain3.ppm
# Paramètres du Quadtree
4 8
# Les éléments du décors
# Indication du nombre d'objet
5
# Objet suivi de la transformation (les 3 translations x,y,z
# plus les 3 rotations par rapport aux axes x,y et z (en degree)
# et enfin un facteur d'homothétie)
/home/biri/Projet/IMAC3/Donnees/Decor/arbre.3ds
1 2 0.5 90.0 45.0 0.0 0.001
/home/biri/Projet/IMAC3/Donnees/Decor/arbre.3ds
-1 0.2 0.5 -90.0 45.0 0.0 0.001
/home/biri/Projet/IMAC3/Donnees/Decor/arbre.3ds
0.0 -2 0.5 0.0 45.0 0.0 0.001
/home/biri/Projet/IMAC3/Donnees/Decor/batiment_1.3ds
10.0 -2 5.0 0.0 0.0 0.0 0.001
/home/biri/Projet/IMAC3/Donnees/Decor/batiment_2.3ds
-4.0 -2.1 -3.2 90.0 -45.0 0.0 0.001
# Ici on a utilisé que des objets 3DS mais rien ne vous empeche
# de générer des modèles simples ou complexes vous meme
```

3 Partie animation

Cette partie représentera l'animation des personnages joueurs (ou non) du jeu. Le groupe proposera d'une part une application *stand-alone* permettant de visualiser et construire les animations et d'autre part, tout le nécessaire pour l'intégration de ces animations dans le jeu.

Le module animation devra :

1. charger des modèles de personnage au format `.md4` et/ou `.hal`
2. construire une animation de ces squelettes
3. visualiser le modèle animé par une animation
4. réaliser un mélange d'animation

Avant d'implémenter ce module, mettez au point un format de fichier permettant de stocker vos animations.

3.1 Chargement de modèle de personnage

Votre module devra être capable de charger des personnages au format `.md4` ou `.hal`. Ces fichiers décrivent des personnages en 3D ainsi que toutes les informations nécessaires à la construction de leur squelette.

Une fois chargé, ces modèles devront être visible, au moins de façon non animée, dans votre application stand-alone, autant que dans la plateforme du jeu. Dans l'application stand-alone, le squelette du personnage devra être visible.

Vous pouvez trouver des modèles de personnages (MODS) sur `http://www.planetquake.com/`

3.2 Construction d'une animation

Votre application stand-alone devra être capable d'éditer des animations simples avec la technique des images clés. Vous avez toute liberté quand aux spécifications de ces animations. Les animations résultantes devront pouvoir être stockée dans un fichier dont le format est laissé à votre discrétion.

3.3 Visualisation des animations

Votre application stand-alone devra ensuite être capable d'afficher en temps réel les modèles animés. Elle devra également pouvoir afficher le squelette seul.

Ces animations devront être visible également dans le moteur de jeu.

3.4 Mélange des animations

Enfin votre application devra être capable de mélanger deux animations ou plus.

4 Partie IHM/réseau

4.1 Le boulot

Tout d'abord, le groupe chargé de la partie IHM/Réseau est également plus ou moins chargé de l'intégration étant donné que le boulot des deux autres groupes doit pouvoir s'intégrer à ce que vous allez développer.

Ensuite comme le titre de cette partie l'indique, il y a deux sous-groupes à former, un sous-groupe IHM et un sous-groupe réseau.

4.2 IHM

Votre tâche consiste principalement à gérer l'interaction entre l'utilisateur et le système de jeu. Il s'agit donc principalement de proposer une solution permettant d'afficher les informations importantes pour le joueur (points de vie, armes utilisées, etc) et de gérer les événements clavier et réseau.

Vous devrez également vous occuper de tout ce qui est configuration de l'application, tout d'abord en proposant une solution de fichiers de configurations et ensuite en proposant un système de menus permettant de changer les diverses options du jeu tels que l'adresse du serveur de jeu, le port sur lequel se connecter, le pseudo du joueur, etc.

Techniquement vous pouvez utiliser GLUT ou SDL comme système de gestion d'événements. Nous n'avons rien contre d'autres systèmes mais les deux systèmes proposés ont l'avantage d'une assez grande simplicité.

4.3 Réseau

Votre boulot consiste à proposer un protocole d'échange permettant la propagation des divers événements vers les différents joueurs d'une partie. Ce protocole devra être utilisable sur un réseau local type LAN.

Le système sera de type client/serveur et l'un des joueurs hébergera le serveur.

Le protocole devra permettre de chercher les parties en cours, de proposer une partie, de se joindre à une partie, de jouer une partie.

Il devra également donner la possibilité de fournir la map proposée aux joueurs n'en disposant pas.

Vous pourrez vous procurer une jolie petit bibliothèque c++ (assez bas niveau) à l'adresse suivante: http://www.waba.be/c++/wfd/.