Benjamin Gaillard
Nicolas Riegel

17 mai 2006

Matériels et Interfaces Graphiques

Jeu 3D en OpenGL/GLUT

Rapport

Master 1 RIA — Option MIG

ulp
UNIVERSITÉ LOUIS PASTEUR
STRASBOURG

Table des matières

1

Introduction

1.1 Choix du sujet

Nous avons choisi la réalisation d'une application interactive de type navigation dans un monde en 3D. C'est en s'inspirant d'un jeu existant, Wipeout, que nous est venue l'idée de faire ce jeu. Il consiste en une course chronométrée avec un petit vaisseau qui vole le long d'un circuit. Le sujet comporte plusieurs exigences :

- l'utilisation exclusive d'OpenGL avec comme système de fenêtrage GLUT (utilisé uniquement pour ouvrir une fenêtre et un contexte OpenGL pour récupérer les évènements) ;
- il est imposé d'utiliser des modèles 3D et des textures 2D ;
- il est imposé d'utiliser l'éclairage OpenGL (donc au moins une source de lumière OpenGL).

Le jeu devra aussi fonctionner correctement sur les machines de la salle C310 sous Linux. Il est important et imposé que le jeu comporte des éléments de navigation 3D à l'aide des touches de clavier et/ou de la souris. L'environnement devra être modélisé en 3D à l'aide de polyèdres. Les textures peuvent être récupérées sur le web ainsi que les modèles 3D (s'il y a lieu). Les éléments actifs (personnages ou certains objets par exemple) pourront être modélisés soit explicitement en 3D soit sous la forme de *billboards*. On intégrera des objets tri-dimensionnels animés. Ces objets pourront répondre à un stimulus extérieur (suite à une action de l'utilisateur) ou avoir une autonomie propre.

1.2 Compilation et exécution

La projet utilisant les GNU Autotools, sa compilation se fait de la même manière que la plupart des applications Unix. Placez-vous dans la racine des sources du projet et tapez :

```
./configure [options...]
make
```

Puis, éventuellement, `make install` pour l'installer dans la hiérarchie spécifiée au script `configure` (/usr/local par défaut). Pour le lancer, tapez `podz` si vous l'avez installé, on `src/podz` si vous souhaitez l'exécuter directement depuis le répertoire des sources.

Dans ce programme, vous pourrez utiliser les touches suivantes :

Échap ou Q : Quitte le jeu.

Flèche haut : Accélère.

Flèche bas : Freine jusqu'à l'arêt.

Flèche droite ou gauche : Tourner à droite ou à gauche.

F : Met ou sort du mode plein écran.

T : Désactive les textures.

L : Désactive les lumières.

Espace ou P : Met en pause.

1.3 Informations supplémentaires

Ce jeu a été réalisé intégralement en anglais (à quelques commentaires près) afin qu'il puisse éventuellement être repris et amélioré par d'autres personnes si nous mettons à disposition notre projet.

La compilation a été testée avec GCC sous Linux et Microsoft Visual C++[1] sous Windows. Elle devrait également se dérouler correctement sous Mac OS X, mais ne possédant pas de Mac, nous n'avons pas pu tester cette possibilité. Nous nous sommes basés sur des renseignements trouvés sur Internet pour une compatibilité avec ce système d'exploitation.

[1] Version gratuite disponible sous l'intitulé « Microsoft Visual C++ Express Edition »

2

Structure du programme

La fonction main() est dans le fichier Application.cpp. Elle ne fait qu'instancier la classe `Application` qui contient les objets principaux et lancer la boucle principale de GLUT.

GLUT utilise un mécanisme de *callback* pour informer l'application des événements qui se produisent. L'interfaçage avec nos classes se fait en utilisant des variables statiques contenant les instances des classes qui géreront les événements. Les fonctions de *callback* les utilisent pour appeler les méthodes correspondantes. Cela nous permet d'interfacer GLUT avec notre projet orienté objet.

2.1 Interfaçage avec GLUT

GLUT permet de gérer plusieurs sources d'évéments. Nous avons classés ceux qui nous intéressaient en trois catégories, chacune représentée par une classe.

2.2 L'affichage : classe Display

Cette classe contient tout le système de gestion d'affichage. Lors de son instanciation, on initialise la fenêtre GLUT. Pour passer en mode plein écran, on utilise la fonction glutEnterGameMode(). Malheuresement cette fonction nous oblige à relancer toutes les fonctions de rappel GLUT ainsi qu'à recharger toutes les textures. Pour l'affichage de texte nous avons créé une petite fonction dans la classe `Display` : DisplayText(). Cette fonction prend en paramètre le texte et les coordonnées, et éventuellement la taille.

Les différents objets susceptibles d'avoir un effet avec l'affichage sont, après instanciation de cette classe, enregistrés (ils sont stockés dans une liste). Les classes de ces objets dérivent toutes de la classe abstraite (virtuelle) `Object`.

Lors d'un événement demandant l'affichage de la scène, trois méthodes sont appelées successivement, et chacune pour tous les objets :

– mise au point de la matrice de *modelview* ;
– paramétrage des lumières OpenGL ;
– finalement, affichage des objets 3D.

2.3 La gestion du clavier : classe Keyboard

Pour pouvoir gérer les touches multiples (par exemple avant et droite en même temps), on a mis un booléen pour chacune des 4 touches de direction. On définit les 3 fonctions de callback GLUT qui gèrent l'appuie d'une touche normale, l'appuie d'une touche spéciale et le relachement d'une touche spéciale. Pour cela on utilise la même astuce que précédemment. Les deux dernières mettent simplement à jour les booléens. C'est avec le timer que l'on viendra regarder quelles touches sont enfoncées ou non et agir en conséquence.

2.4 Le timer : classe Timer

C'est dans cette classe que vont être gérés les différents états du jeu, le chronomètre, ainsi que les déplacements du vaisseau. Pour le timer, on utilise la fonction GLUT. Lorsque le timer expire, on le réarme, puis selon l'état on affiche le message correspondant ou on déplace le vaisseau.

3

Rendu 3D

3.1 Classe de base Object

Les objets qui vont pouvoir être affichés ont tous une classe qui dérive de la classe `Object`. Cette classe définit les méthodes virtuelles appelées par `Display` ainsi que deux fonctions permettant de dessiner un triangle et un quadrilatère de manière simple (calcul de la normale, coordonnées de texture automatiques...).

Le vaisseau a été modélisé « à la main », c'est-à-dire que nous n'avons pas utilisé de logiciel de modélisation, ni de modèle trouvé sur internet. L'affichage du vaisseau utilise les fonctions DrawQuad() et drawTriangle() qui sont définies dans la classe `Object`. Pour celà on procède par héritage. De la même manière, toutes les classes qui représenteront des objets 3D hériteront de la classe `Dessin`. C'est le cas par exemple de la classe `Circuit`. Ces deux fonctions prennent en paramètre respectivement 4 et 3 points, un numéro de texture et éventuellement les coordonnées d'affichage de la texture. La fonction met aussi en place les normales des polygones.

3.2 Textures : classe Texture

La gestion des textures se fait avec la classe `Texture`. Elle permet de charger des images au format BMP (format brut non compressé) dans la mémoire de la carte graphique. La méthode Select permet de sélectionner une textures pour les opérations graphiques courantes.

3.3 Objets 3D du jeu : classes Cube, Circuit et Vehicle

Ces trois classes redéfinissent la méthode virtuelle d'affichage de la classe Object. Certaines redéfinissent aussi la méthode de paramétrage des lumières et Vechicle celle

de placement de la caméra (matrice de *modelview*). Ces trois classes sont :

- **`Cube` :** c'est un cube géant englobant toute la scène et qui affiche un décor au moyen de 6 textures, une par face (cet objet n'étant pas influencé par la lumière) ;
- **`Circuit` :** cette classe permet entre autre d'afficher le circuit chargé en mémoire ;
- **`Vehicle` :** celle-ci définit les lumières avant du vaisseau ainsi que son desssin, en plus d'autres fonctions.

3.4 Listes d'affichage

La classe `Object` stocke les commandes OpenGL de paramétrage des lumières et d'affichage dans des listes d'affichage (*Display List*). Cela ne concerne évidemment que les commandes qui sont toujours les mêmes et pas celles qui varient comme l'affichage d'informations textuelles sur le jeu en cours.

L'avantage d'utiliser ces listes est d'accroître les performances générales du programme puisqu'il n'est dès lors plus nécessaire de faires des calculs pour obtenir les coordonnées des objets à afficher. Cela est particulièrement intéressant pour la classe `Circuit` qui fait des calculs vectoriels pour l'affichage.

4

Le moteur de jeu

4.1 Classes utilitaires

Afin de (grandement) faciliter la programmation des classes liées au moteur du jeu, deux classes représentant des objets génériques. Il s'agit des classes :

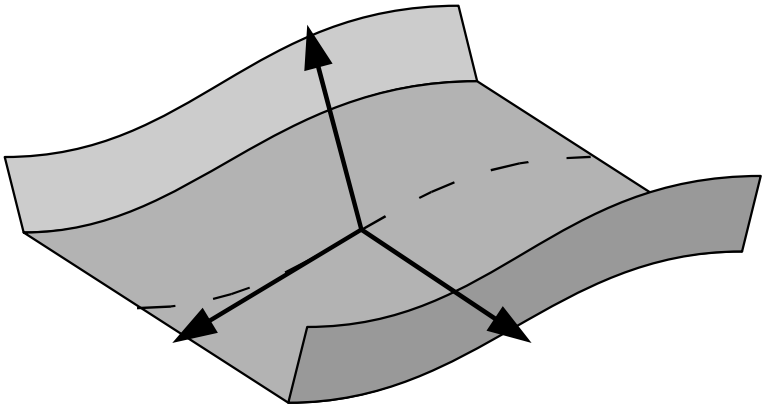
- **`Vector` :** opérations sur les vecteurs à trois dimensions (addition, produit vectoriel, mise à l'échelle, rotation, normalisation…) ainsi que la surcharge des opérateurs correspondant pour une meilleure facilité d'utilisation ;
- **`Basis` :** un repère dans un environnement 3D (origine et vecteurs unitaires), utilisé pour faire des changements de repère dans l'espace (utilisation d'une matrice de transformation et de son inverse).

4.2 Les données du circuit

La classe `Circuit` charge et stocke les informations relatives au circuit. Pour construire un circuit, elle lit depuis un fichier les points de l'espace par lesquels il passe, ainsi qu'une normale indiquant l'inclinaison du circuit à ce point.

Au moyen d'une méthode d'interpolation, les points intermédiaires sont calculés et à chacun un repère (classe `Basis`) est associé, indiquant donc quatre informations sur ce point : sa localisation (origine) et trois vecteurs unitaires qui sont : la tangente, la normale et un vecteur allant vers le bord (côté).

Cette classe nous situe sur le circuit en utilisant une valeur appelée *position* : il s'agit de la distance que l'on aurait parcourue si l'ont était toujours resté exactement au centre de la piste. En utilisant une telle valeur, la classe `Circuit` est capable de donner un repère (par interpolation) dans l'espace.



4.3 Le vaisseau

La classe `Vehicle` s'occupe de tout ce qui touche le vaisseau que le joueur contrôle. Ses méthodes sont appelées par la classe `Keyboard` pour réagir aux événements clavier de l'utilisateur. Elle est aussi responsable de gérer la physique grâce à une méthode appelée périodiquement par le timer.

C'est la classe la plus complexe car elle effectue beaucoup d'opérations vectorielles et de changement de base.

5

Conclusion

Ce projet nous a permis d'approfondir nos connaissances dans le langage C++ acquises l'année dernière, ainsi que dans le fonctionnement d'OpenGL. Même si la programmation OpenGL n'est pas très adaptée au C++, nous avons réussi à nous débrouiller. Concernant le choix du jeu, nous avons essayé de redonner vie à un jeu qui nous a passionné durant notre jeunesse, même si nous n'arrivons pas tout à fait à la qualité du Wipeout original. Nous avons aussi voulu insister sur la jouabilité et non pas uniquement sur la beauté des graphismes. Bon amusement !