Rapport TP 4 GMIN 317

MORIN Matthieu 20135804

Présentation des nouvelles fonctionnalités

Sauvegarde de la scène

La sauvegarde de la scène est effectuée au sein même de la classe « *TriangleWindow* », dans la fonction :

```
void TriangleWindow::saveToFile( const char *c )
```

Je commence par sauvegarder le temps (le jour et la saison). Je sauvegarde ensuite toute les données de la caméra (échelle, animation, état, etc.). Par la suite, j'enregistre le nombre de points, le chemin vers l'image, et, enfin, l'ensemble de la géométrie (chacun des points à la suite. Le caractère séparateur est l'espace.

Pour une utilisation réelle, je n'aurais pas sauvegardé le nombre de points ni le chemin vers l'image, mais la largeur et la hauteur de l'image. En effet, toutes les fonctions de dessin utilisent l'objet *m_image*, pour dessiner. Cela m'aurait évité de recharger inutilement l'image, en utilisant ces deux valeurs, et donc d'alléger le programme puisqu'il n'y aurait plus besoin de l'image. Cependant, pour simplifier, et pour éviter de modifier tout le code, je sauvegarde le nombre de points, et le chemin vers l'image. C'est pour cela que le nom de l'image ne peut comporter d'espace (le caractère séparateur étant, justement, l'espace).

Chargement de la scène

Le chargement de la scène se fait dans la fonction :

```
void TriangleWindow::loadFromFile( const char *c )
```

Dans celle-ci, je réinitialise le jour et la saison, puis la caméra. Je pourrais ensuite juste lancer la fonction :

```
void TriangleWindow::loadMap(QString localPath)
```

Cependant, ça n'est pas le but de ce TP. Je charge donc l'image afin d'initialiser *m_image*, sans faire appel à cette fonction. Le nombre de vertex étant sauvegardé dans le fichier, je peux donc allouer suffisamment de mémoire pour stocker mes points. Il me suffit donc de parcourir le fichier et de remplir mon tableau de points.

Je peux également garantir qu'après le chargement, les points seront dessinés dans le même ordre qu'avant la sauvegarde, puisque la sauvegarde garde l'ordre.

Chargement de données au format PLY

Le chargement de données au format PLY se fait dans la classe « *MeshParser* », et plus précisément dans la fonction :

```
void MeshParser::from_PLY_file( const char * fileName )
```

Pour simplifier, je suppose que tous mes fichiers auront un en-tête similaire, et auront tous une normale et une position. Je passe donc l'en-tête, en gardant les données importantes (nombre de faces et nombre de points).

En lisant un fichier PLY, on remarque qu'une face peut être composée de plus de 3 points, et qu'il peut y avoir des segments et des points. Toutes les faces (ainsi que les segments et les points) sont donc chargées avec leurs points respectifs. Cependant, au rendu, j'ai fait le choix de n'afficher que les triangles et les quadrilatères, car je n'ai pas trouvé de faces de plus de 4 côtés, ni de segment ou de points.

Le code ne gérant pas la lumière, les normales sont utilisées pour la couleur. En effet, une normale est composée de 3 composantes X, Y et Z, comprises entre 0 et 1. Une couleur est, elle, composée de 4 composantes R, G, B et A, comprises entre 0 et 1. Je garde l'opacité au maximum (Alpha), et fais :

R = X ; G = Y ; B = Z ;

De cette manière, on peut voir que les normales sont correctement chargées.

Chaque objet a également :

- Une position ;
- Une rotation ;
- Une échelle ;

Qui lui sont propres. Ces données sont modifiables en temps réel.

Note

Actuellement, le programme charge le fichier de sauvegarde créé par le programme, lors d'un précédent test.
Le chargement se fait dans « TriangleWindow.cpp », ligne 100 et 102.
La sauvegarde, elle, se fait en dé-commentant les lignes 158 et 159, du fichier TriangleWindow.cpp.
Tel quel, seule la fenêtre « maîtresse » fait la sauvegarde. Au chargement, toutes chargent le même fichier.

Parties bonus

Ajouter d'autres formats 3D

Pour ajouter d'autres formats 3D, il suffit d'ajouter les fonctions permettant de *parser* ces derniers. Bien sûr, le mieux serait de modifier la structure du programme, afin de suivre le design pattern « *factory* ». Ainsi, on pourrait ajouter plusieurs fonctions différentes permettant d'avoir un résultat équivalent sur des données sauvegardées différemment. En effet, généralement, en utilisant l'extension, ou, plus intelligemment, en lisant l'en-tête du fichier, on peut connaître le format d'encodage du fichier. On peut donc faire un premier filtrage pour savoir quel *parser* utiliser si ce dernier est développé. Une fois le format trouvé, le *parser* ne ferait que remplir les mêmes structures de données que les autres. Le rendu ne changerait donc pas, puisqu'au final, tout ce dont nous avons besoin, c'est d'un maillage ainsi que de données annexes (couleur, coordonnées de texture, normales, textures…).