BARRIOL Rémy
TUCHOLSKI Lilian

Compte Rendu #4
Projet : Ultimate 3D Crypto-Compressor
Sujet #1 Crypto-compression d'objets 3D

Lien Git : https://github.com/BarriolRemy/Ultimate-3D-Crypto-Compressor
Lien Replit : https://replit.com/@theghoul/CryptoCompressor

Nous allons à présent commencer l'implémentation du projet. Nous allons naturellement nous diriger vers du C++ pour réaliser le projet, notamment en prévision de potentiellement utiliser nos TPs en *HAI911I – Développement d'applications interactives* pour pouvoir réaliser le logiciel demander (dans le cas où la visualisation du modèle serait nécessaire à l'application).

I. Gestion des fichiers 3D et visualisation

Pour pouvoir compresser un modèle 3D, il faut tout d'abord avoir la possibilité de le lire pour le modifier, ainsi que pouvoir réécrire le dit fichier 3D. Il faut donc pouvoir choisir un ou plusieurs type de fichier pas trop complexe à lire et écrire mais suffisamment courant pour pouvoir trouver des fichiers tests.

Nous allons tout d'abord nous diriger intuitivement vers les fichiers .OBJ. Ils ne possèdent pas une entête complexe et sont facilement gérable depuis un programme puisque qu'il n'y a d'écrit que les coordonnées des vertex et les faces. Pour les modèles 3D, nous ne prenons pas en compte la gestion des normales ainsi que les coordonnées de textures. Nos exemples seront des modèles 3D uniquement composés de triangles.

Pour avoir des modèles 3D, comme ceux de Stanford (http://graphics.stanford.edu/data/3Dscanrep/) par exemple, nous passerons par Blender pour convertir les fichiers en .OBJ, tout en ignorant l'écriture des normals, coordonnées de textures, sans les matériaux ainsi qu'avec seulement des triangles en tant que faces. (Nous utiliserons donc le modifiers Triangulate).

Nous considérons pour l'instant que nous ne prenons pas en compte les coordonnées de textures et les normales, notamment par le fait que les articles que

nous avons analysés n'en font pas mention. Nous considérerons donc les fichiers sans ces éléments lorsque nous calculerons le taux de compression.

Tant que nous n'aurons pas de logiciel permettant de visualiser les modèles, nous utiliserons Blender pour visualiser les résultats. On peut remarquer que nous pourrons comparer nos résultats par rapport au modification Decimate de Blender.

II. Implémentation : Lecture de fichier

Le code est en train d'être implémenté sur Replit.com (d'où le fait qu'il n'est pas encore présent sur le git.

Notre première objectif est l'implémentation d'une lecture de fichier .OBJ, et donc avoir un tableau contenant toutes les vertices du modèle ainsi qu'un tableau contenant l'ensemble des faces.

Nous avons réussi cette tâche. Nous avons aussi implémenté l'écriture d'un fichier OBJ à partir de ces deux tableaux.

III. Implémentation : KD-tree Coder

Nous avons actuellement fini la phase de "Split", qui divise le kd tree (de la taille de la boîte englobante du modèle) jusqu'à ce qu'il ne reste qu'un seul vertex par zone du kd tree.

Les vertex utilisés plus tard lors de la phase de merge seront les "centres de gravité" des zones créés (lorsqu'il y a une vertex du modèle de base dans la zone).

Nous sommes actuellement en train d'implémenter la phase de merge, où l'on va ainsi "fusionner", toutes les itérations, des vertex adjacents par des opérateurs de décimations. Chaque tour va ainsi faire une itération de décimations.

La phase de merge sera la plus longue à implémenter, puisqu'elle est centrale de tout le processus.

Après le merge, l'étape sera de commencer à sauvegarder les opérations de décimations binaires pour ensuite prévoir de les écrire, dans le sens inverse, dans le fichier binaire.

IV. Bibliographie

[1] PIERRE-MARIE GANDOIN, OLIVIER DEVILLERS. (2002) Progressive Lossless Compression of Arbitrary Simplicial Complexes. *ACM Transactions on Graphics, Association for Computing Machinery*, pp.372-379.
https://hal.inria.fr/file/index/docid/167216/filename/hal.pdf