Crypto-Compression 3D

Compte Rendu 5

Potin Clément, Fournier Romain
Master 2 IMAGINE
Université de montpellier
2021

Problématique

L'objectif est de développer un algorithme qui, de manière conjointe, compresse avec ou sans pertes et chiffre un objet 3D. Les performances de cette méthode seront ensuite mesurées en termes de taux de compression et de qualité de reconstruction de l'objet 3D.

Edgebreaker

Traduit et adapté du premier article portant sur l'algorithme Edgebreaker ([6]) :

"*Edgebreaker* est un système simple de compression des graphes maillages triangulaires tridimensionnels.

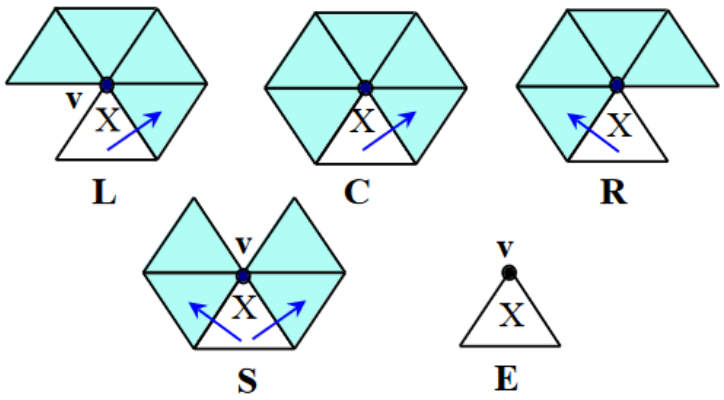
Edgebreaker réduit l'espace de stockage nécessaire à la description du maillage, même dans les cas les plus "défavorables". La plupart des autres algorithmes nécessitent $O(nlogn)$ bits pour stocker le graphe d'incidence d'un maillage de n triangles. *Edgebreaker* ne nécessite que $2n$ bits ou moins pour les maillages simples et peut également supporter des maillages complexes (maillages non homéomorphes à une sphère (ex: Tore), maillages troués/partiels, ...) en utilisant un stockage supplémentaire pour chaque spécificité du maillage.

Les processus de compression et de décompression d'*Edgebreaker* effectuent la même traversée du maillage, d'un triangle à un triangle adjacent. À chaque étape, la compression produit un code (binaire) décrivant la relation topologique entre le triangle actuel et la limite de la partie restante du maillage. La décompression utilise ces mêmes codes pour reconstruire le graphe d'incidence complet.

Comme la compression et la décompression d'*Edgebreaker* sont indépendantes de l'emplacement des sommets, elles peuvent être combinées avec diverses techniques de compression des sommets qui exploitent les informations topologiques du maillage pour mieux estimer l'emplacement des sommets (techniques de prédiction, vecteurs de correction, etc).

Grâce à ses capacités de compression supérieures, à la simplicité de sa mise en œuvre et à sa polyvalence, *Edgebreaker* est particulièrement adapté aux nouvelles normes d'échange de données 3D pour les applications graphiques interactives."

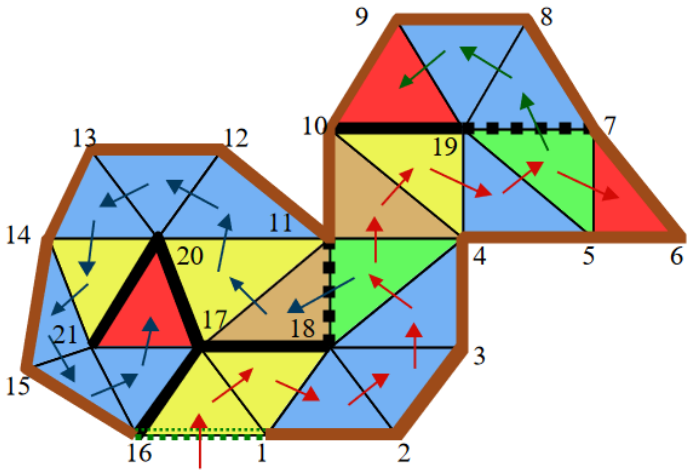
Cette semaine, nous nous sommes donc intéressés à l'implémentation de l'algorithme *Edgebreaker*. Le principe est simple : pour compresser un maillage, on le parcourt triangle par triangle (adjacent) en choisissant le triangle d'origine aléatoirement dans le maillage (généralement, on prendra le premier triangle/triangle d'indice 0), et on encode la configuration dans laquelle on se trouve à chaque étape. Les configurations peuvent être les suivantes :



Configuration possibles d'Edgebreaker

Plutôt qu'un code binaire, moins parlant, on enregistrera plutôt une chaîne "CLERS", nommée à partir des configurations possibles (C, L, E, R et S). Bien qu'enregistrés en ASCII dans un premier temps (et donc sur 8 bits), on pourra ensuite appliquer un codage de *Huffman* sur ces caractères et obtenir le meilleur taux de compression sans perte possible pour cette suite de configurations (meilleur qu'un code binaire "statique").

Par exemple, pour le maillage suivant, parcouru dans le sens des flèches (rouges, puis vertes, puis bleues) :



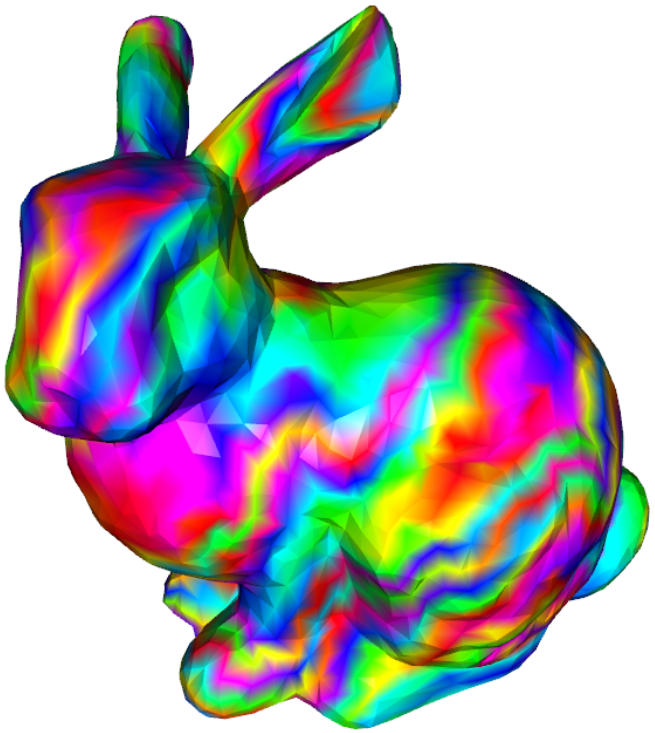
Exemple de maillage traversé par Edgebreaker

On obtiendrait la chaîne CLERS suivante :

CCRRRSLCRSERRELCRRRCRRRE

Si on associe à cette chaîne la position des sommets dans l'ordre de parcours, on compresse grandement l'espace nécessaire pour décrire les triangles de notre maillage.

En appliquant l'algorithme *Edgebreaker* sur le modèle 3D du *Lapin de Stanford* (précédemment utilisé dans nos rapports), et en coloriant les triangles du modèle selon leur ordre de parcours (simplement pour visualiser), on obtient ces résultats :



Lapin de Stanford parcouru par notre implémentation d'Edgebreaker

Si on veut améliorer le taux de compression global, on peut travailler sur les informations décrivant les positions des sommets du maillage en n'enregistrant que la position du premier sommet, puis des vecteurs permettant d'aller du sommet courant au sommet suivant. De cette façon, on obtiendra des nombres plus petits ainsi que des informations récurrentes (car plutôt que d'enregistrer des positions sur l'ensemble de la boîte englobante du maillage, on n'enregistre qu'un vecteur de quelques dixièmes de centimètres). De cette façon, un codage de *Huffman* appliqué à ces informations sera d'autant plus efficace.

Pour aller plus loin, on peut également utiliser une méthode de prédiction de la position des sommets de notre maillage. Par exemple, en connaissant les normales de 2 sommets d'un triangle et en supposant que tous les triangles du maillage sont équilatéraux, on peut prédire la position du 3ème sommet. Plutôt que d'enregistrer cette prédiction, on ne va enregistrer que le vecteur de correction (allant du point prédit au sommet réel). Ce vecteur de correction sera en général très petit (bien plus qu'un vecteur allant d'un sommet à un autre comme dans la méthode précédente), voire presque nul sur les maillages les plus réguliers, et la compression du codage de *Huffman* sera encore améliorée.

À la reconstruction du maillage, il suffira de prédire la position du sommet et d'y appliquer le vecteur de correction enregistré pour retrouver la position exacte (sauf si quantification) du sommet.

Chiffrement

Positions

Pour chiffrer les positions, nous avons utilisé une méthode de chiffrement préservant la géométrie. Ces algorithmes visent à préserver la nature de nos données, afin qu'elles puissent être lues indépendamment du fait qu'on ait, ou pas, utilisé une clé valide pour le déchiffrement.

Nous avons, pour chiffrer nos données, considéré nos N vecteurs à M dimensions comme comme M listes à N éléments. Ainsi, on se retrouve pour les vecteurs :

$$\begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}, \begin{bmatrix} 6 \\ 5 \\ 4 \end{bmatrix}, \begin{bmatrix} 9 \\ 8 \\ 7 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Avec les listes:

$$\{3,\ 6,\ 9,\ 0\}, \{2,\ 5,\ 8,\ 0\} \text{ et } \{1,\ 4,\ 7,\ 0\}.$$

Ces listes représentent nos coordonnées respectives en X, Y et Z. En appliquant un algorithme de chiffrement par transposition, nous pouvons masquer les données originales tout en préservant la nature de nos données. Par exemple, si on transpose nos listes, nous pourrons reconstruire les vecteurs suivants :

$$\{3,\ 0,\ 9,\ 6\}, \{8,\ 5,\ 2,\ 0\} \text{ et } \{0,\ 7,\ 4,\ 1\}.$$

$$\begin{bmatrix} 3 \\ 8 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 5 \\ 7 \end{bmatrix}, \begin{bmatrix} 9 \\ 2 \\ 4 \end{bmatrix}, \begin{bmatrix} 6 \\ 0 \\ 1 \end{bmatrix}$$

On remarquera que, bien que cette méthode puisse créer de manière pseudo-aléatoire des vecteurs qui n'existaient pas à l'origine dans notre maillage, elle conserve la boîte englobante de ce dernier. En effet, chaque dimension étant mélangée indépendamment des autres, le X_{max} et X_{min} restent dans la liste des X.

Pour transposer nos listes de manière contrôlée, nous pré-calculons une liste de transpositions à exécuter sur chaque dimension. Une transposition est représentée comme un tuple $(a, b), \quad 0 \leq a, b < N$, la valeur à l'index a sera échangée avec la valeur à l'index b.

Une liste de transpositions est donc une liste de tuples $\{(a, b), (c, d), \dots, (e, f)\}$. Pour générer cette liste, nous initialisons le générateur de nombres aléatoires avec notre clé concaténée à un sel (technique de "salage" en cryptographie, dont le principe est d'ajouter une donnée de façon à ce que le générateur de nombres aléatoires ne puisse pas, à partir d'une même clé, produire deux fois le même résultat). Puis, nous générons K transpositions par dimension. Pour chiffrer, il suffit d'appliquer les transpositions du début à la fin, et de la fin au début pour déchiffrer.

Normales

Nos normales sont stockées sur 17 $bits$ comme un index dans un dictionnaire à 2^{17} entrées. Pour chiffrer nos normales, nous appliquons un xor sur son index. Ainsi, ce dernier ne pointera plus sur la bonne entrée.

Maillage Original ***Quantifié*** ***Chiffré***

Lien du Git

Notre travail sera mis à jour au lien suivant :

https://github.com/Romimap/3D-CryptoCompression/

Références

Demos & Softwares

1. Vladimir Agafonkin, “Edgebreaker, the Heart of Google Draco” : https://observablehq.com/@mourner/edgebreaker-the-heart-of-google-draco

2. Google Draco: https://github.com/google/draco

Papers

3. Michael Deering, “Geometry Compression”, sun Microsystems, 1995 : http://web.cse.ohio-state.edu/~shen.94/Su01_888/deering.pdf

4. Chandrajit L Bajaj, Valerio Pascucci, Guozhong Zhuang, “Single Resolution Compression of Arbitrary Triangular Meshes with Properties”, University of Texas, 1997: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.946&rep=rep1&type=pdf

5. Mike M. Chow, “Optimized Geometry Compression for Real-time Rendering”, Massachusetts institute of Technology, May 1997: https://www.semanticscholar.org/paper/Optimized-geometry-compression-for-real-time-Chow/39babe9519e6b58bae4350e4f57be86dc45ce6f9

6. Jarek Rossignac, “Edgebreaker: Connectivity compression for triangle meshes”, Georgia Institute of Technology, 1999 :

https://www.cc.gatech.edu/~jarek/papers/EdgeBreaker.pdf

7. Daniel Cohen-Or, David Levin, Offir Remez, “Progressive Compression of Arbitrary Triangular Meshes”, Tel Aviv University, 1999:
https://www.tau.ac.il/~levin/vis99-dco.pdf

8. Jarek Rossignac, Alla Safonova, Andrzej Szymczak, “3D Compression Made Simple: Edgebreaker on a Corner-Table”, College of Computing and GVU Center, Georgia Institute of Technology, 2001:
https://www.researchgate.net/publication/3896746_3D_compression_made_simple_Edgebreaker_with_ZipandWrap_on_a_corner-table

9. Pierre Alliez, Mathieu Desbrun, “Progressive Compression for Lossless Transmission of Triangle Meshes”, University of Southern California, February 2002:
https://www.researchgate.net/publication/2534417_Progressive_Compression_for_Lossless_Transmission_of_Triangle_Meshes

10. Jarek Rossignac, “3D mesh compression”, College of Computing and GVU Center Georgia institute of Technology, January 2003:
https://www.researchgate.net/publication/27521282_3D_Mesh_Compression

11. Esam Elsheh, A. Ben Hamza, "Secret sharing approaches for 3D object encryption", Concordia Institute for Information Systems Engineering, Concordia University, Montréal, QC, Canada, 2011:
https://www.sciencedirect.com/science/article/abs/pii/S095741741100724X

12. In-Ho Lee and Myungjin Cho, "Optical Encryption and Information Authentication of 3D Objects Considering Wireless Channel Characteristics", Department of Electrical, Electronic, and Control Engineering, Hankyong National University, Ansung 456-749, Korea, October 2013:
https://www.osapublishing.org/DirectPDFAccess/766AED72-4C7E-47EE-BB28209C333886A8_276786/josk-17-6-494.pdf

13. Marc Éluard, Yves Maetz, and Gwenaël Doërr, "Geometry-preserving Encryption for 3D Meshes", Technicolor R&D France, November 2013:
https://www.researchgate.net/profile/Gwenael-Doerr/publication/273257218_Geometry-preserving_Encryption_for_3D_Meshes/links/54fc4b660cf2c3f52422a624/Geometry-preserving-Encryption-for-3D-Meshes.pdf

14. Xin Chen, Jingbin Hao, Hao Liu, Zhengtong Han and Shengping Ye, “Research on Similarity Measurements of 3D Models Based on Skeleton Trees”, School of

Mechatronic Engineering, China University of Mining and Technology, Daxue Road 1, Xuzhou 221116, China, State Key Laboratory of Materials Forming and Mould Technology, Huazhong University of Science and Technology, Luoyu Road 1037, Wuhan 430074, China, April 2017:
https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwi91fmtwZj0AhUI2BoKHYcKA1AQFnoECAMQAQ&url=http%3A%2F%2Fwww.mdpi.com%2F2073-431X%2F6%2F2%2F17%2Fpdf-vor&usg=AOvVaw0lfo-8VxxYFuVQTnyCt6JI

15. Ying Zhou, Lingling Wang, Lieyun Ding, Cheng Zhou, "A 3D model Compression Method for Large Scenes", Huazhong Univ. of Science and Technology, 2018:
https://www.iaarc.org/publications/fulltext/ISARC2018-Paper207.pdf