Le 19/10/2016

Dominic Bergeron MATRICULE

Jules Thuillier 1521088

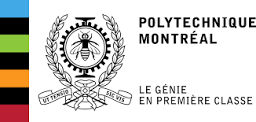
A Montréal

Rapport TP1 - INF4710

Introduction aux technologies multimédia

A2016 - Travail pratique #1

Compression d’images fixes



# Présentation de l’algo et de l’implémentation

L’algorithme de compression LZ77 est un des connus père à la base de nombreux algorithmes de compression utilisés aujourd’hui (Gzip, Winrar, 7zip…). Sa flexibilité lui permet de compresser de nombreux types de fichiers différents, c’est pourquoi de nombreuses variantes en découlent pour s’adapter au mieux à certains types de fichiers.

Pour ce TP nous avons implémenté l’encodage et décodage de signaux au format d’uint8\_t (bien que le format du signal est facilement interchangeable).

Voici comment ce passe **l’encodage**:

**Initialisation de la fenêtre de taille N**  
On remplit la gauche (n1) avec des « 0 » et la droite avec le début du signal

**Itération jusqu’à ce qu’il n’y est plus de signal à traiter**

**Recherche du premier match du premier caractère de la fenêtre droite dans la fenêtre de gauche**

**Détermination de la longueur maximale de la suite de symboles équivalents**

**Décalage des données de la fenêtre droite dans la gauche, on remplit la droite avec le signal**

**FIN**

Et pour le **décodage**:

**Récupération de la série de symbole dans le dictionnaire**

**FIN**

**Itération sur le vecteur LZ77**

**Initialisation du dictionnaire à « 0 »**

**Ajout de la série au signal autant de fois que la longueur nécessaire**

**Ajout du dernier symbole à la fin du signal**

# Validité des taux de compression

|  |  |  |  |
| --- | --- | --- | --- |
| Image | Taille de l’image | Taille compressée | Taux de compression |
| data/test1.png | 307200 | 20721 | 0,9325488281 |
| data/test2.png | 114002 | 7218 | 0,9366853213 |
| data/test3.png | 116353 | 33090 | 0,7156068172 |
| data/test4.png | 921600 | 61131 | 0,9336686198 |
| data/test5.png | 441045 | 29082 | 0,9340611502 |
| data/test6.png | 282375 | 29793 | 0,8944913679 |
| data/test7.png | 110010 | 16686 | 0,8483228797 |
| data/test8.png | 786432 | 1738548 | -1,2106781006 |
| data/test9.png | 196608 | 18525 | 0,9057769775 |
| data/test10.png | 196608 | 583041 | -1,9654998779 |

Pour le calcul du taux de compression, la **taille de l’image** est la taille du vecteur sortie de ***format\_image***, et la **taille compressé** est 3x la taille du vecteur sortant de ***lz77\_encode*** (3x car ce vecteur est composé de 3 uint8\_t).

Les tailles de fenêtre choisies sont :

N = 100

n1 = 50

# Commentaires et discussion sur efficacité

On remarque que l’efficacité peut-être très bonne sur certaines images comme très mauvaise sur d’autres (taux négatif).

Cela dépend entièrement de la composition de l’image :

* Les images comprenant beaucoup de changement de couleurs (photo test8) sont beaucoup plus dures à compresser est peu adapté à notre méthode qui attend une certaine linéarité.
* L’analyse se fait ligne par ligne, donc les images dont les lignes restent assez identiques sur toutes leur longueur sont mieux compressée (test2 > test3). D’ailleurs cela se voit très bien dans les test9 et test10 où la différence n’est quasiment qu’une simple rotation de l’image, mais étant donnée l’analyse par ligne, le gradient devient plus difficile à compresser.

Une image très « simple » avec peu de couleurs et surtout peu de contrastes très proches les uns des autres se compressera bien mieux qu’une photo ou des gradients (à l’exception du gradient horizontal).

Cet algorithme n’est donc pas très bien adapté à des situation réelle de besoin de compression d’image.

# Amélioration proposée

Afin d’améliorer la compression, on pourait :

Optimiser les tailles des fenêtres (dynamiquement).

Passer sur un encodage DEFLATE, comme utilisé pour le PNG, en ajoutant au LZ77 des arbres de Huffman dynamiques pour les probabilités d’apparitions.