**IMT Atlantique**

Filière par apprentissage

655 Avenue du Technopôle

29280 Plouzané

**Projet :**

**Compression Vidéo**

|  |  |
| --- | --- |
| **Groupe :** | NGUYEN Binh Minh  GONG Xiang  SANTOS SEISDEDOS Carlos |
| **Formation :** | FISE 2A – UE G : Compression de données : du codage de sources à la réalité virtuelle |
| **Date :** | 18 Mars 2020 |
| **Lieu :** | IMT Atlantique (Brest) |
| **Encadrant :** | DUPRAZ Elsa |

# Sommaire

[Sommaire 3](#_Toc35468131)

[Contexte 4](#_Toc35468132)

[Codage indépendant d’images suivant l’algorithme JPEG 5](#_Toc35468133)

[Codage indépendant des images d’un vidéo suivant l’algorithme JPEG 10](#_Toc35468134)

[Codage prédictif des images d’un vidéo 11](#_Toc35468135)

# Contexte

Le but principal de ce projet c’est de **mettre en œuvre une solution de compression et décompression vidéo**. Afin d’organiser notre code MATLAB, nous avons créé un dépôt git en ligne sur GitHub.

Vous trouverez nos fichiers dans l’adresse suivante : [github.com/chletes/IMTA-CD-Video\_Compression\_Project](https://github.com/chletes/IMTA-CD-Video_Compression_Project).

Dans le dépôt GitHub, vous trouverez trois dossiers :

* Le dossier /code/, qui contient les *scripts*, commençant par ‘s\_’, et les fonctions, commençant par ‘f\_’, que nous avons codé pour la réalisation de ce projet.
* Le dossier /data/images/, qui contient les images et les vidéos que nous utilisons pour faire les démonstrations (donc, utilisés par les *scripts*).
* Le dossier /ressources/, qui contient l’énoncé du projet (‘project\_description.pdf’) et autres dossiers.
  + Le dossier /ressources/biblio/ contient des informations que nous avons utilisé pour comprendre l’algorithme de compression JPEG ou l’algorithme d’estimation de mouvement, entre autres.
  + Les dossiers /ressources/BlockMatchingAlgoMPEG et /ressources/video\_and\_code contiennent des fonctions externes (pas à nous).
  + Les dossiers /ressources/TP1\_Lossless\_Coding et /ressources/TP2\_Lossy\_Source\_Coding contiennent des fonctions dont nous avons codé pour les Travaux Pratiques 1 et 2 de l’UE.

Nous avons procédé de la suivante façon pour mettre en œuvre notre solution de compression et décompression vidéo.

Dans un premier temps, nous avons mis en œuvre une **solution complète pour encoder et décoder des images**.

Dans un deuxième temps, nous avons mis en œuvre une **solution complète pour encoder et décoder les images de la vidéo indépendamment les unes des autres**.

Dans un troisième temps, nous avons mis en œuvre un **codage prédictif pour encoder et décoder les images successives d’un vidéo les unes par rapport aux autres**.

Vous trouverez, dans la suite de ce document, des explications sur notre projet, des explications sur les fichiers les plus importants – *scripts* et fonctions – ainsi que sur les **performances**, en termes de taux et de distorsion, **de notre compression**.

# Codage indépendant d’images suivant l’algorithme JPEG

* Petite introduction sur l’algorithme JPEG.

L’algorithme que nous avons choisi pour le codage et le décodage d’images indépendantes c’est **l’algorithme de compression JPEG**. Il s'agit d'un algorithme de compression avec perte utilisé pour réduire la taille des fichiers d'images. Cela signifie qu'en décompressant ou en affichant l'image, vous n'obtenez pas exactement la même image qu'avant la compression. La compression avec perte ne convient pas aux images ou aux graphiques dont le texte, les lignes ou les bordures sont très nettes, mais elle est adaptée aux fichiers qui contiennent de grandes zones de couleur unie.

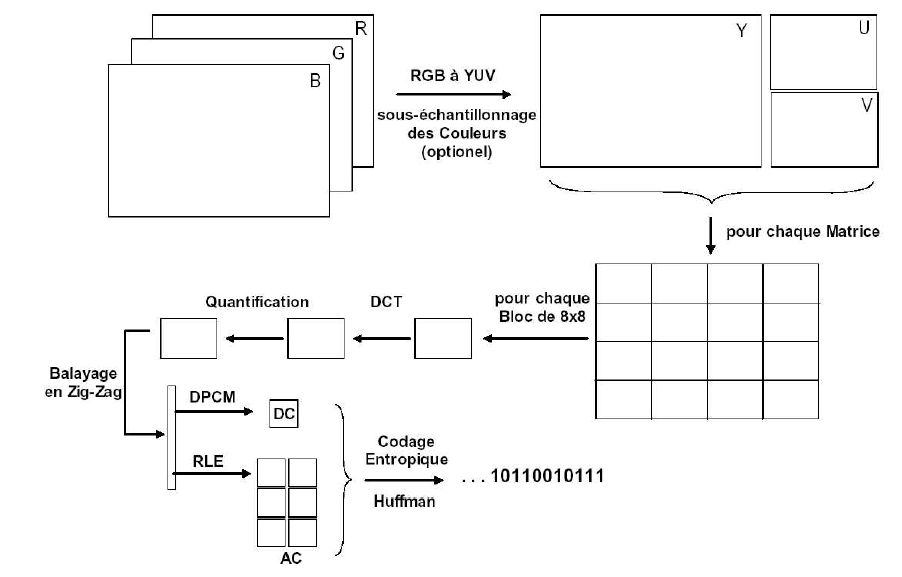
L'algorithme JPEG est basé sur deux phénomènes visuels de l'œil humain :

1. L’œil humain est beaucoup plus sensible aux changements de luminance que de chrominance, c'est-à-dire qu'il capte les changements de luminosité plus clairement que les changements de couleur.
2. L’œil humain détecte plus facilement les petites variations de luminosité dans les zones homogènes que dans les zones où la variation est importante, par exemple sur les bords des corps des objets.

Une des caractéristiques de JPEG est la possibilité d’ajuster le degré de compression. Un taux de compression très élevé se traduira par une taille de fichier réduite, au prix d'une perte de qualité importante. Un faible taux de compression produira une qualité d'image très similaire à celle de l'original, mais la taille du fichier sera plus importante.

* Description de l’algorithme suivi.

Pour coder l’algorithme JPEG, nous avons suivi la suivante image, issue du document ‘C1\_C2\_TraitementMonomedia.pdf’ que vous pouvez trouver dans le dossier /ressources/biblio.

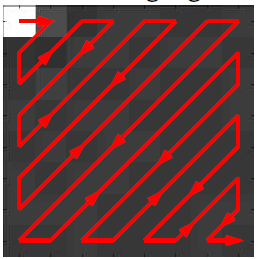


En effet, l’algorithme suit ces étapes :

1. Nous transformons l’image dans l’espace de couleur YUV. Y représente la luminance relative, et les deux autres, U et V, représentent la chrominance. En MATLAB, ces composantes sont représentées en sous forme de matrices.
2. Nous décomposons chaque composante en blocs de taille 8x8 et chaque bloc de taille 8x8 suit les suivantes étapes :
   1. Nous calculons la transformée en cosinus discrète (de l'anglais : DCT ou Discrete Cosine Transform) de chaque bloc pour obtenir les coefficients DCT de chaque bloc.
   2. Nous pondérons les coefficients DCT de chaque bloc l’aide d’une table de quantification que prévoit la norme JPEG et qui dépend de la qualité désirée de l’image comprimée. Voici la table de quantification que prévoit la norme JPEG pour obtenir une qualité égale à 50 :

À partir de cette matrice de quantification, nous pouvons calculer d’autres matrices pour d’autres qualités .

* 1. Grâce à un balayage en zig-zag, indiquée dans la suivante image, nous séparons le premier coefficient de chaque bloc quantifié – appelé composante continue ou DC – des restes des coefficients. Cela est fait parce qu’un traitement (un codage) spécifique est réservé pour eux : le codage prédictif DPCM (de l’anglais *Differential Pulse Code Modulation*). Le reste de coefficients – appelés composants AC – sont associés entre eux par un codage RLE (de l’anglais *Run-Length Encoding)*



* 1. Pour finir avec la compression, le codage entropique de Huffman est utilisé pour les 64 coefficients du bloc – soit les coefficients DC et AC.

Pour décoder les images encodées, nous avons suivi les étapes de l’algorithme de compression JPEG dans l’ordre inverse.

* Description de notre algorithme.

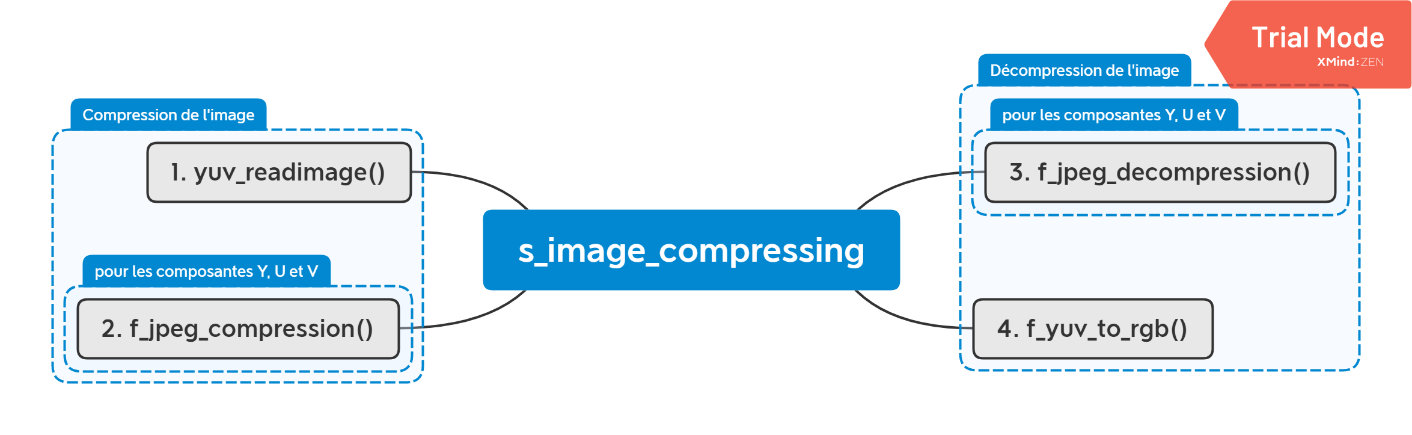
Afin de pouvoir faire une démo de notre algorithme, vous trouverez le *script* s\_image\_compressing.m dans le dossier /code/ du dépôt GitHub. Nous avons aussi créé la suivante carte mentale pour vous expliquer comment nous avons codé la compression et la décompression d’une image, et notamment pour suivre le *script* d’une manière plus claire. La partie gauche de la carte mentale permet la compression d’une image ; et la partie droite permet la décompression de l’image.

Figure 1. Carte mentale du *script* s\_image\_compressing.m

Quant à la compression d’une image, après la vérification de l’existence du fichier (ou de l’image), le *script* s’appuie sur la fonction yuv\_readimage(), disponible dans le dossier /ressources/video\_and\_code, pour obtenir les 3 composantes de l’espace de couleurs YUV. Ensuite, la fonction f\_jpeg\_compression() est appelé pour obtenir chaque composante comprimée. En effet, c’est la fonction f\_jpeg\_compression() celle qui met en œuvre l’algorithme de compression JPEG, expliqué dans la section d’avant.

Et quant à la décompression de l’image, la fonction f\_jpeg\_decompression() est appelé pour obtenir chaque composante décomprimée. C’est cette fonction là celle qui met en œuvre l’algorithme de décompression JPEG. Ensuite, la fonction f\_yuv\_to\_rgb() permet de transformer les trois composantes YUV à l’espace de couleurs RGB, grâce à ces formules :

Quant aux fonctions f\_jpeg\_compression() et f\_jpeg\_decompression(), nous recommandons jeter un coup d’œil au code directement (nous avons mis beaucoup de commentaires pour l’expliquer).

Quant à la performance de notre code pour la compression et la décompression d’une image, nous avons mesuré le temps écoulé pour des qualités allant de à . Le suivant graphique montre la moyenne du temps écoulé pour des qualités allant de à . Nous pouvons donc distinguer une tendance croissante, montrée par la ligne en pointillés rouges, du temps de compression et décompression d’une image avec la qualité de l’image demandée. Nous pouvons conclure que plus on demande une qualité élevée, plus de temps l’algorithme écoule.

Moyenne du temps écoulé lors de la compression et décompression JPEG d’une image pour des qualités allant de à .

Quant à la distorsion d’une image avant la compression et après la décompression, nous avons mesuré PSNR pour des qualités allant de à d’après les suivantes formules :

Où, représente la valeur maximale de I, soit l’image avant compression, et

Où,

* représente l’image avant compression, et donc représente le pixel de l’image sans compression,
* représente l’image après décompression, et donc représente le pixel de l’image décomprimée,
* et représentent la taille de l’image (nombre de pixels en colonne et nombre de pixels en ligne).

Le suivant graphique montre la variation du PSNR pour des qualités allant de à . Nous pouvons donc distinguer une tendance croissante du PSNR, montrée par la ligne en pointillés rouges, avec la qualité de l’image demandée. Nous pouvons conclure que plus on demande une qualité meilleure, plus le PSNR est élevé, comme c’est normal.

PSNR d’une image pour des qualités allant de à .

# Codage indépendant des images d’un vidéo suivant l’algorithme JPEG

# Codage prédictif des images d’un vidéo