INF4710
Introduction aux technologies multimédia

A2017 - Travail pratique #2
Compression avec perte (JPEG/MPEG-1)

Objectif :

- Permettre à l'étudiant de se familiariser avec la compression avec perte (JPEG), et au suivi de mouvement pour la compression de séquences d'images (MPEG-1).

Remise du travail :

- Au plus tard, le 20 octobre 2017, 15h00, sur Moodle - **aucun retard accepté**

Référence principale :

- Notes de cours sur Moodle, chapitre sur compression avec perte

Documents à remettre :

- L'ensemble de votre code source (.m pour Matlab, .hpp/cpp pour C++, mais pas les deux!)
- Un rapport (**format .pdf, max 6 pages**) contenant un survol de votre travail (présentation de toutes les étapes du pipeline), un tableau de taux de compression obtenus pour toutes les images fournies, et les paragraphes de discussion demandés, pour un total de 3-5 pages

Autres directives :

- Pensez à commenter l'ensemble de votre démarche directement dans votre code! Sinon, difficile d'attribuer des points lorsque ça ne fonctionne pas...
- Les TD s'effectuent en équipe de **deux** (peu importe la section de laboratoire). Utilisez le forum Moodle pour trouver, c'est **votre responsabilité**!

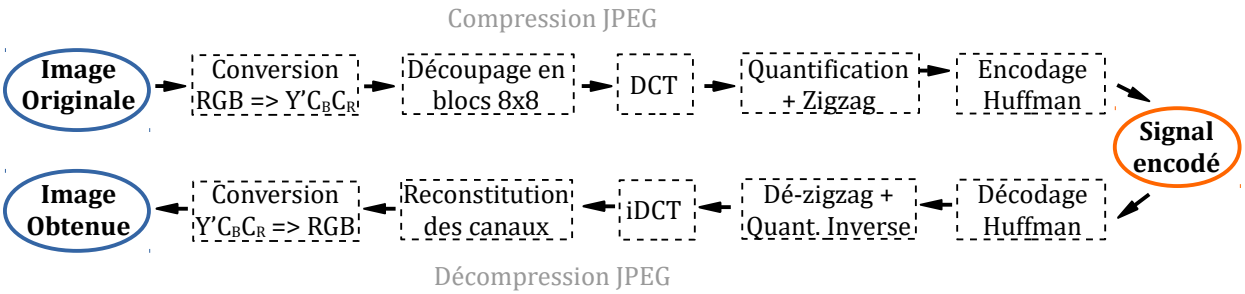
Présentation

L'objectif de ce travail pratique est d'implémenter un pipeline de compression d'images similaire à l'algorithme d'encodage JPEG, qui tient compte du modèle de vision humain (qui lui se comporte comme un filtre passe-bas). Vous aurez donc à implémenter certaines étapes de la procédure d'encodage JPEG, telles le sous-échantillonnage $Y'C_BC_R$, le découpage en blocs, la DCT, et le parcours de blocs en zig-zag. Après avoir complété les fonctions nécessaires, vous pourrez évaluer chaque étape du pipeline et déduire leur rôle ainsi que l'effet de leurs paramètres sur les résultats obtenus. Dans un deuxième temps, une procédure simplifiée de recherche de correspondances temporelles pour des blocs d'images devra être développée — celle-ci constitue la base de l'algorithme de compression vidéo MPEG-1.

Votre rapport devra présenter (brièvement) toutes les étapes du pipeline de traitement JPEG, les images compressées-décompressées, ainsi que leurs taux de compression (toujours en fonction des paramètres du pipeline). Veillez à bien identifier dans votre présentation quelles étapes du pipeline causent une perte, et quelles étapes « compressent » le signal. Pour plus d'informations sur les exigences du TP, voir le barème à la dernière page.

Partie 1 : Codage JPEG d'une image

La norme JPEG constitue une des méthodes de compression les plus répandues pour les images fixes. Elle consiste à transformer l'image en un code, dont l'espace de stockage est plus faible que celui de l'image de départ. Pour réobtenir l'image de départ, ou techniquement l'image de départ dégradée, il s'agit de décoder le code en mémoire. Les étapes de codage/décodage sont illustrées ci-dessous et expliquées dans ce qui suit.



Conversion RGB/Y'C_BC_R :

Cette étape consiste à convertir l'espace de couleur d'une image vers un autre espace à l'aide d'une transformation linéaire. Bien entendu, cette conversion n'est pas nécessaire sur des images qui ne possèdent qu'un seul canal — toutes les images fournies avec le TP ont d'ailleurs trois canaux. Ici, nous nous intéressons aux conversion de RGB vers Y'C_BC_R, et de Y'C_BC_R vers RGB.

Complétez les fonctions dans 'conv_rgb2ycbcr.h/m' et 'conv_ycbcr2rgb.h' qui prennent chacune comme argument une (ou plusieurs) matrice(s) d'une image dans un certain espace de couleur, et qui retourne une copie de l'image dans l'autre espace. **N'utilisez pas les fonctions de Matlab ou d'OpenCV pour faire la conversion — on veut voir vos manipulations au complet!** De plus, ces deux fonctions prennent un autre argument, i.e. un booléen qui dicte si l'on doit considérer un sous-échantillonnage 4:2:0 dans l'espace Y'C_BC_R. S'il n'y a pas de sous-échantillonnage, tous les canaux de données en format Y'C_BC_R seront de même taille, et sinon, les canaux C_B et C_R seront plus petits (voir les notes de cours). Pour les conversions, utilisez les équations suivantes :

$$Y' = 0.299R + 0.587G + 0.114B$$
$$C_B = 128 + 0.564(B - Y')$$
$$C_R = 128 + 0.713(R - Y')$$

$$R = Y' + 1.403(C_R - 128)$$
$$G = Y' - 0.714(C_R - 128) - 0.344(C_B - 128)$$
$$B = Y' + 1.773(C_B - 128)$$

D'autre part, notez qu'en Matlab, la troisième dimension (profondeur) des images lues par 'imread' contient les canaux dans l'ordre R,G,B, tandis qu'avec OpenCV, les triplets dans la matrice retournée par 'cv::imread' sont dans l'ordre B,G,R. Voir les tableaux fournis avec l'énoncé du TP1 pour un exemple!

Suite à l'implémentation de ces deux fonctions, vous pourrez déjà évaluer la perte générée par le changement d'espace de couleur (avec et sans sous-échantillonnage) sur les images de tests fournies avec l'énoncé. Dans votre rapport, **discutez de l'effet de ce changement d'espace sur la qualité de l'image (avec et sans sous-échantillonnage)** sur quelques images de votre choix**,** et **illustrez la différence entre les images originales et celles réobtenues** après l'opération inverse (modifiez le '**main**', et utilisez la fonction `imabsdiff` de Matlab, ou la fonction `cv::absdiff` de OpenCV pour voir la différence).

Le découpage en blocs de pixels :

Cette étape consiste à subdiviser une matrice (ou un 'canal', puisqu'on va les traiter indépendamment) en blocs de taille 8x8. Pour cela, **complétez la fonction dans 'decoup.h/m'** qui prend comme entrée une matrice 2D (dont la longueur et la largeur sont toujours des multiples de 8) et qui retourne en sortie ses blocs dans une matrice 3D (en Matlab) ou dans un vecteur de matrices 2D (en C++) de taille 8x8xN, avec N le nombre total de blocs. On vous demande d'effectuer le découpage en **ligne de bloc par ligne de bloc (i.e. en format « row-major »)**. En d'autres termes, deux blocs consécutifs dans le vecteur de sortie devraient être des voisins horizontaux dans l'image, ou être décalés d'une ligne s'ils touchent à une bordure.

Inversement, vous avez à **compléter la fonction dans 'decoup_inv.h/m'** qui reconstitue une matrice 2D à partir de ses blocs. Cette fonction prend aussi en argument la taille originale de la matrice à reconstruire (nécessaire, comme pour le formatage 1D→2D du TP1!).

Transformée en cosinus discrète (DCT) :

Après le découpage de l'image en en blocs, la transformée en cosinus discrète est appliquée à chaque bloc à partir de l'équation ci-dessous; ***n*** représente la taille du bloc à transformer et ***F(x,y)*** représente la couleur correspondant au pixel (x,y).

$$\boldsymbol{C(u,v) = c(u).c(v).\sum_{x=1}^{n}\sum_{y=1}^{n} F(x,y).\cos\left(\frac{\pi(2(x-1)+1)(u-1)}{2n}\right).\cos\left(\frac{\pi(2(y-1)+1)(v-1)}{2n}\right)}$$

Où:

$$\boldsymbol{c(w) = \sqrt{\frac{1}{n}}} \qquad \text{si } \boldsymbol{w = 1}$$

$$\boldsymbol{c(w) = \sqrt{\frac{2}{n}}} \qquad \boldsymbol{sinon}$$

La DCT prend un ensemble de points d'un domaine spatial et les transforme en une représentation équivalente dans le domaine fréquentiel. Elle permet d'identifier facilement les hautes fréquences afin qu'elles soient filtrées. Ici, nous vous demandons de **compléter la fonction dans 'dct.h/m'** qui calcule la transformée en cosinus discrète d'un bloc carré de l'image. **Encore une fois, n'utilisez pas les fonctions d'OpenCV ou de Matlab — on veut voir vos calculs!** Par contre, pour vérifier le fonctionnement de votre fonction, vous pouvez comparer vos coefficients à ceux obtenus avec la fonction **`dct2`** de Matlab ou **`cv::dct`** de OpenCV. Remarque importante: faire attention aux types utilisés lors des calculs intermédiaires!

Par la suite, **complétez la fonction dans 'dct_inv.h/m'** qui calcule la transformée inverse de la DCT selon l'équation ci-dessous; ***n*** représente la taille du bloc considéré, la fonction ***c(w)*** est la même que celle donnée plus haut, et ***C(x,y)*** représente le coefficient DCT du pixel (x,y).

$$F(x,y) = \sum_{u=1}^{n}\sum_{v=1}^{n} \boldsymbol{c(u).c(v)}C(u,v).\cos\left(\frac{\pi(2(x-1)+1)(u-1)}{2n}\right).\cos\left(\frac{\pi(2(y-1)+1)(v-1)}{2n}\right)$$

Notez bien que le résultat de la DCT est une matrice 8x8 de **type float ou double**, et que l'opération inverse doit retourner le type original de l'image (**uint8/uchar**). Dans votre rapport, **discutez de l'effet de cette transformation** sur quelques images de votre choix.

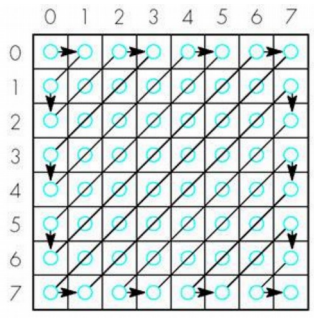
Quantification :

Suite à l'application de la DCT, la quantification s'applique sur chaque bloc de coefficients obtenu. Cette étape consiste à diviser terme par terme le bloc DCT de taille 8x8 par la matrice de quantification elle aussi de taille 8x8, créée à partir du facteur de qualité **F**. Le fichier contenant la fonction **'quantif'** (.h/m) et celui contenant sont opération inverse ('**quantif_inv**') vous sont déjà fournis, mais vous aurez à les compléter (il manque une opération assez triviale). Notez que les matrices de quantification sont naïvement données pour quatre niveaux de qualité, soit **F=10**, **F=50, F=90,** et **F=100**.

Les résultats obtenus après quantification doivent être arrondis vers les valeurs entières les plus proches. Le type entier 16-bit signé **int16/short** (au lieu de **uint8/uchar**) est nécessaire ici afin de préserver les valeurs négatives issues de la DCT — l'encodage de Huffman devrait par contre accommoder n'importe quelle taille d'entier sans affecter la compression finale. La quantification inverse consiste à multiplier les valeurs d'un bloc par la même matrice de quantification, et de les retourner en **float**.

Zigzag :

Les blocs quantifiés sont ensuite parcourus en zigzag afin de créer un vecteur 1D (comme dans la figure ci-dessous). Vous devez **compléter la fonction dans 'zigzag.h/m'** afin de créer un vecteur des valeurs parcourues. Notez que sa fonction inverse, **'zigzag_inv.h/m', vous est fournie, et ne doit pas être modifiée**.



Encodage de Huffman :

Enfin, l'encodage de Huffman est utilisé ici pour compresser les vecteurs 1D obtenus après le zigzag. La même fonction que celle donnée au TP1 est utilisée: pour accéder au code de Huffman, utilisez le champ 'string' de la structure fournie en sortie. Pas besoin de compléter quoi que ce soit dans cette étape!

Analyse du pipeline de compression :

Dans votre '**main**' de la partie 1, pour toutes les images fournies, procédez à une compression selon les quatres niveaux de la fonction **'quantif',** avec et sans sous-échantillonnage. Calculez et transcrivez dans votre rapport tous les taux de compression, et illustrez les images où la compression-décompression semble entraîner le plus de perte. Discutez de l'efficacité de la compression pour toutes les images.

Rappel : Taux de Compr. = 1 - (Longueur du signal compressé / Longueur du signal original)

Partie 2 : Compression par MPEG-1 d'une séquence d'images

La compression de séquences d'images par MPEG-1 (aussi nommé H.261) peut être assez complexe[1]. Dans le cadre de ce TP, notez simplement que l'encodage d'images une à une par le pipeline JPEG de la partie 1 est une configuration possible de MPEG-1 – chaque image encodée est alors considérée comme une « I-frame » (« intra-frame », ou plus communément « keyframe ») dans la séquence de trames encodée. Ce mode de fonctionnement est aussi équivalent à l'encodage MJPEG (Motion-JPEG). Pour éliminer un partie de la redondance dans le contenu temporel de la séquence, MPEG-1 permet l'utilisation de « P-frames » (« predicted-frame ») au lieu de « I-frames », qui elles sont construites en deux étapes (très simplifiées ici!). D'abord, pour chaque bloc de l'image à encoder, on recherche un bloc correspondant à l'intérieur d'un voisinage local dans l'image précédente de la séquence telle que la similarité entre les deux blocs est maximale. Ensuite, le vecteur de déplacement entre ces blocs et la différence d'intensité entre ceux-ci sont encodés dans la « P-frame » (c'est une sorte d'encodage prédictif spatiotemporel).

Ici, on s'intéresse principalement à la recherche de correspondances spatiotemporelles entre blocs de pixels tirés de deux images. Vous aurez à compléter la méthode '**match_block**' permettant de retrouver le vecteur de déplacement **v** dans un voisinage **M**x**N** donné en paramètre qui minimise l'erreur quadratique moyenne (EQM) entre un bloc **b** provenant de l'image $\mathbf{I_t}$ aussi donné en paramètre, et le contenu de l'image $\mathbf{I_{t-1}}$. Vous devrez aussi vous assurer que votre fonction est robuste aux cas spéciaux, c'est à dire peu importe l'emplacement initial ou la taille de **b**, et peu importe la taille du voisinage **M**x**N** donnée en paramètre, vos résultats doivent être valides. Notez ici que pour simplifier le TP, les images $\mathbf{I_t}$ et $\mathbf{I_{t-1}}$ ne possèdent qu'un seul canal (en réalité, ça serait Y, Cb, ou Cr, mais on traite des images en tons de gris pour se simplifier la vie). Vous disposez aussi d'un autre exécutable '**main**' pour tester cette deuxième partie du TP. Assurez-vous de bien tester votre fonction — pas de testbench gratuit cette fois-ci, à vous de tout valider! Finalement, notez que vous n'avez pas besoin de discuter de cette deuxième partie (MPEG) dans votre rapport.

Remarque :

- Si le voisinage **M**x**N** est de taille 16x16 et centré sur le bloc **b** de taille 8x8 fourni, et loin des bordures de l'image, vous aurez à calculer **exactement** 81 valeurs de EQM différentes, pas plus, pas moins! Dessinez le problème sur papier, vous comprendrez mieux!

1 Les plus aventuriers peuvent commencer leur noyade ici: https://www.loc.gov/preservation/digital/formats/fdd/fdd000035.shtml

Références supplémentaires

- Aide-mémoire (« Cheat sheet ») Matlab :
  - http://web.mit.edu/18.06/www/Spring09/matlab-cheatsheet.pdf
- Guide complet Matlab :
  - http://www.mathworks.com/help/pdf_doc/matlab/getstart.pdf
- C++ : utilisez OpenCV pour lire/écrire/modifier vos images!
  - http://opencv.org/
  - http://docs.opencv.org/doc/tutorials/tutorials.html
- Pipeline JPEG :
  - https://en.wikipedia.org/wiki/JPEG
- Pipeline MPEG-1:
  - https://en.wikipedia.org/wiki/MPEG-1

Barême

- **Implémentation et fonctionnement :**
  - conv_rgb2ycbcr/conv_ycbcr2rgb = 1.5 pts
  - decoup/decoup_inv = 1 pts
  - dct/dct_inv = 1.5 pts
  - quantif/quantif_inv = 0.5 pt
  - zigzag = 1.5 pts
  - match_block = 4 pts
- **Rapport :**
  - Présentation du pipeline JPEG (toutes les étapes) = 2 pts
  - Discussion + évaluation perte, étape de conversion couleur seulement = 1 pt
  - Discussion + évaluation perte, étape DCT seulement = 1 pt
  - Discussion + évaluation perte, pipeline JPEG complet = 2 pt
  - Taux de compression et illustrations de pertes = 2 pt
  - Lisibilité, propreté et complétude = 2 pts

(sur 20 pts)