SOUTENANCE 1 : Conception FPGA d'un compresseur vidéo

Vincent Robin : FIPA 3A, Promotion 2023, Option SE

Décembre 2022

ENSTA BRETAGNE

Objectifs du projet :

- Concevoir un modèle de référence de compression/décompression vidéo en langage Python.

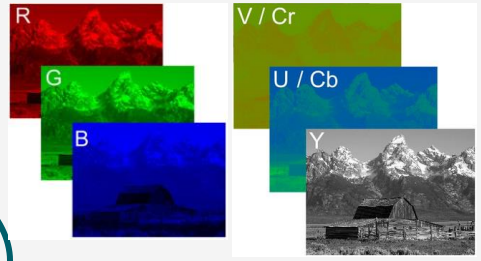
- Générer des images synthétiques avec des motifs géométriques simples.

COMPRESSION

Génération d'une image simple au format PPM

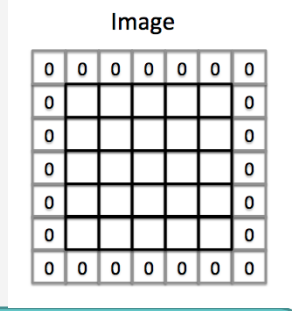
Lecture d'une image en couleurs (RGB) au format .png, .jpg, …

Flux vidéo (format VGA : 480x640)

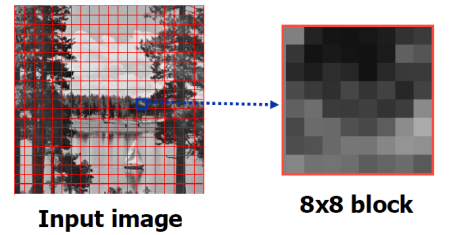


Conversion RGB → YCrCB

Chroma sub-sampling → format 4:2:0

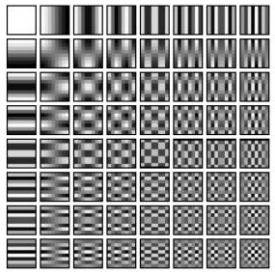


Zero-padding (si nécessaire)



Découpage en blocs 8x8 pour chaque canal (Y,Cr,Cb)

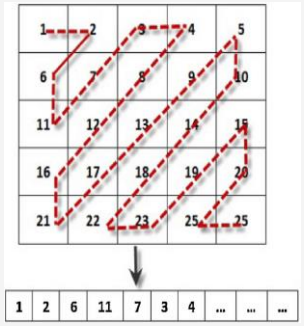
DCT (*Discrete Cosine Transform*)



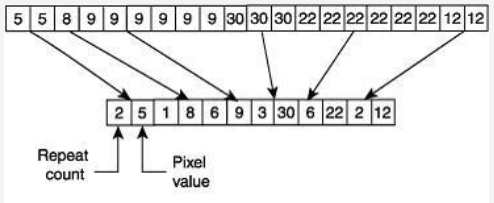
Quantification

Tables de quantification

Zig-zag



Run Length Encoding (RLE)

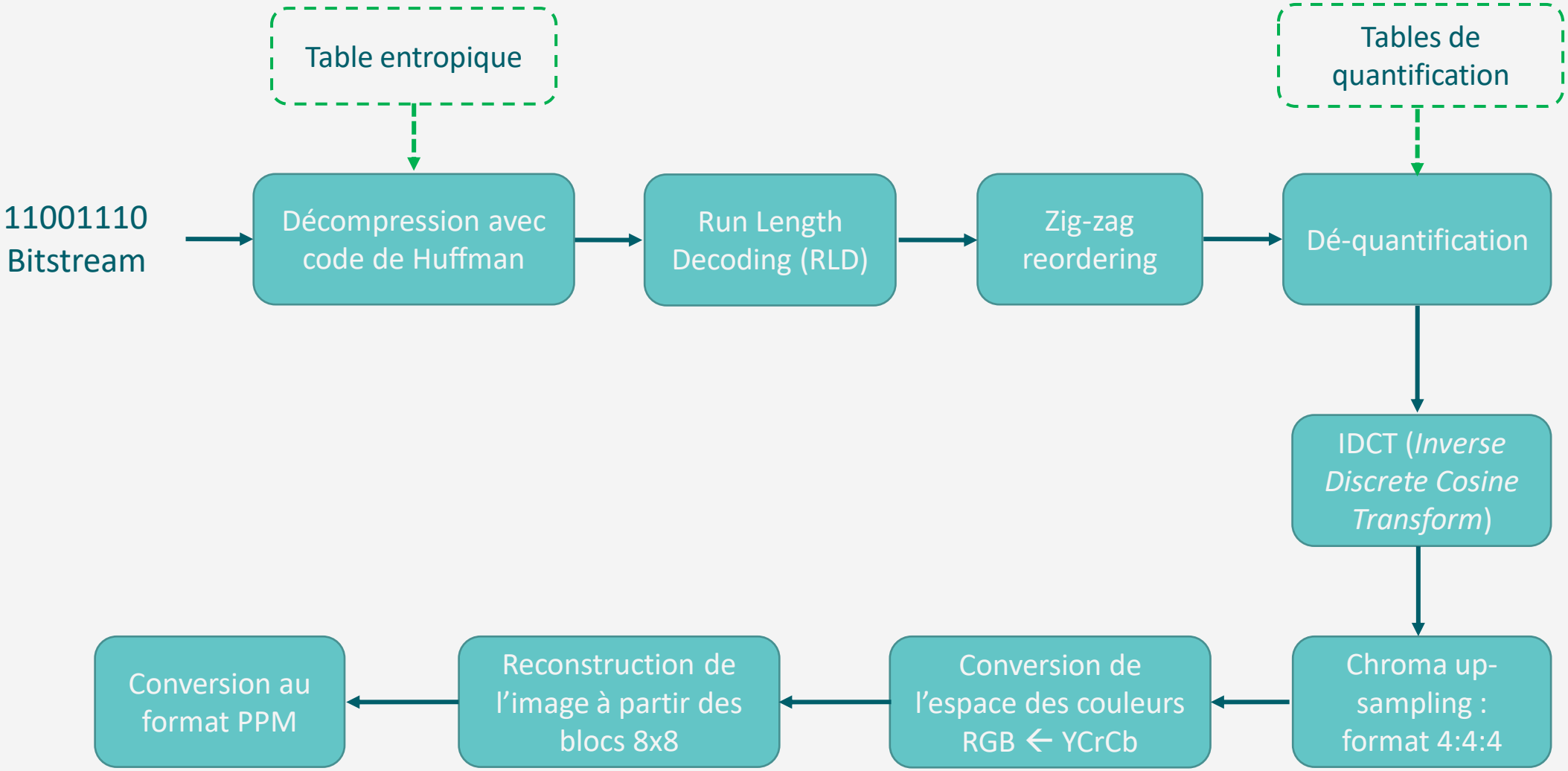


Huffman : code à longueur variable

Spécification de la table entropique

11001110
Bitstream

Processus de décompression



Données d'entrée

- Image PPM (*Portable PixMap)* générée avec un script Python :

mire.ppm

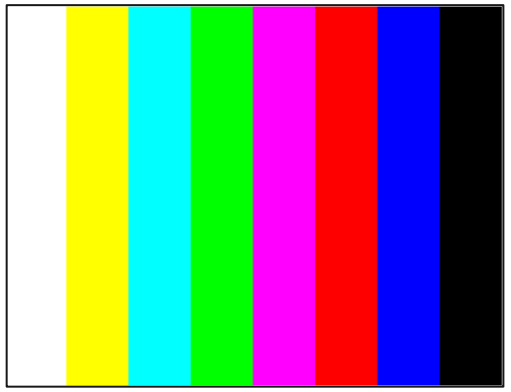
```
1   P3
2   840  640
3   255
4   255
5   255
6   255
7   255
```

Couleurs en ASCII, et codées en RGB.

Format de l'image (W x H)

Valeur max d'un pixel

Valeurs des pixels (entre 0 et 255)



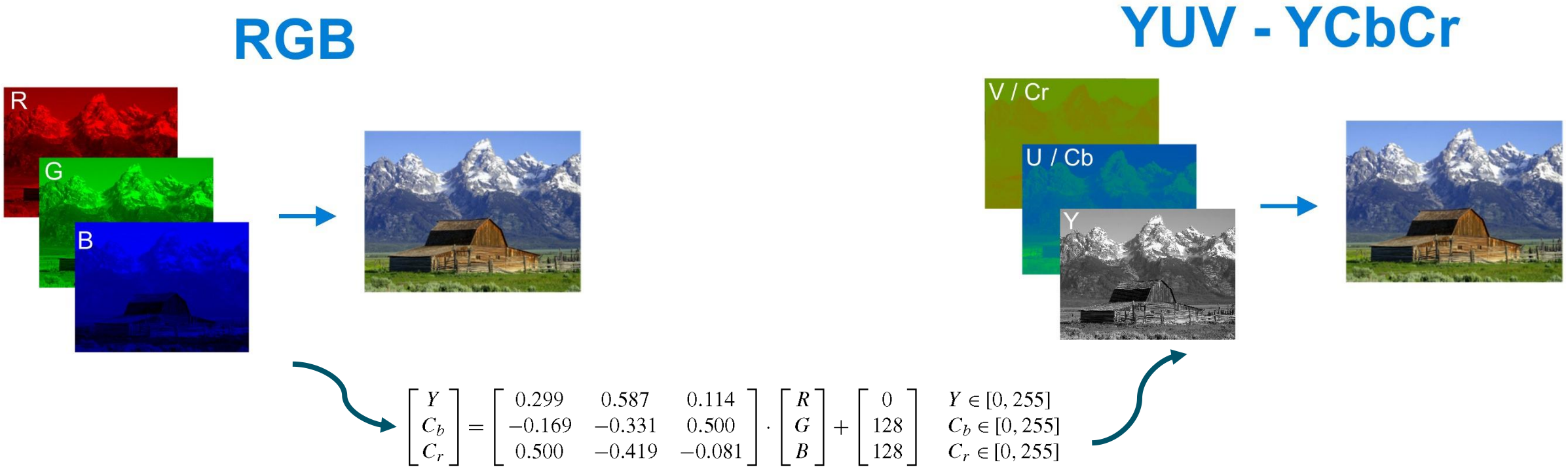
→ Objectif : avoir des images pour lesquelles on maîtrise leurs caractéristiques. La génération « d'imagettes » (sur quelques pixels) nous permettra de tester l'algorithme de compression et pouvoir éventuellement refaire certains calculs à la main, en connaissant parfaitement notre image d'origine décrite sur quelques pixels.

- Lecture d'une image JPEG, PNG, …

- Flux vidéo issue d'une caméra
  - Modèle OV7670 : très bas coût (< 2€)
  - Format de sortie : VGA avec couleurs codées en RGB/YUV

Conversion de l'espace des couleurs (RGB → YCrCb)

- Les meilleurs taux de compression sont obtenus avec des couleurs de type luminance/chrominance : l'œil humain est assez sensible à la luminance (la luminosité) mais peu à la chrominance (la teinte).
- Y est l'information de luminance, et Cb et Cr sont deux informations de chrominance, respectivement le bleu moins Y et le rouge moins Y.

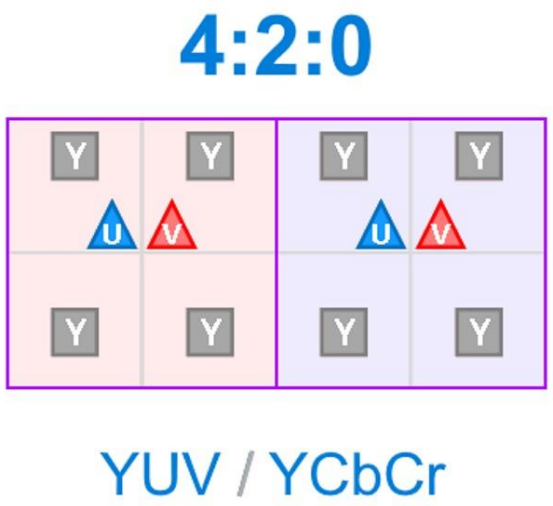


$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} \quad \begin{matrix} Y \in [0, 255] \\ C_b \in [0, 255] \\ C_r \in [0, 255] \end{matrix}$$

Sous échantillonnage de la chrominance

- Chroma subsampling : diminuer la résolution (le nombre d'échantillons) des composantes de couleurs Cb et Cr. On peut se permettre de « dégrader » Cb et Cr, car l'œil humain est beaucoup plus sensible à la luminance (Y).

- Format **4:2:0 :** utilisé pour le grand public : TV, films, DVD, Blu-ray, Blu-ray 4K UHD HDR, jeux vidéo, fichiers vidéo (mp4, divx, avi)….



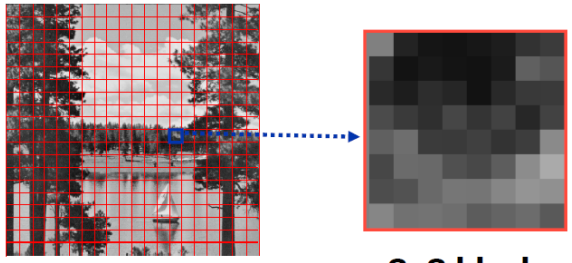
- Résolutions horizontales et verticales des images couleurs (Cb et Cr) divisées par deux.

- La quasi totalité des vidéos que nous regardons, n'ont qu'**un seul pixel de couleur** (un Cb et un Cr) **pour 4 pixels** !

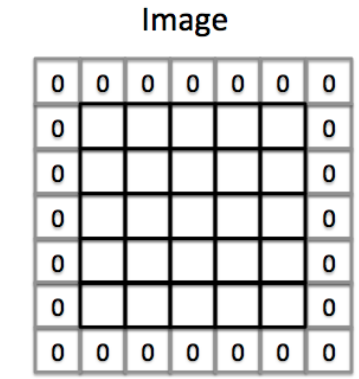
- Bande passante réduite de 50% par rapport au format 4:4:4, et la différence est à peine perceptible par l'œil humain !

Découpage de l'image en blocs (segmentation)

- Principe : subdiviser l'image de départ en sous blocs carrés généralement de taille 8x8 pixels (parfois 16x16)
- Objectifs : réduire les temps et la complexité des calculs qui vont suivre (DCT, …) car on appliquera les transformations sur chaque bloc. On va manipuler des matrices de pixels de taille 8x8.
- *Pourquoi ne pas travailler avec des blocs plus grands ?* Si la taille du bloc est trop grande, les pixels n'ont plus beaucoup de relations entre eux.

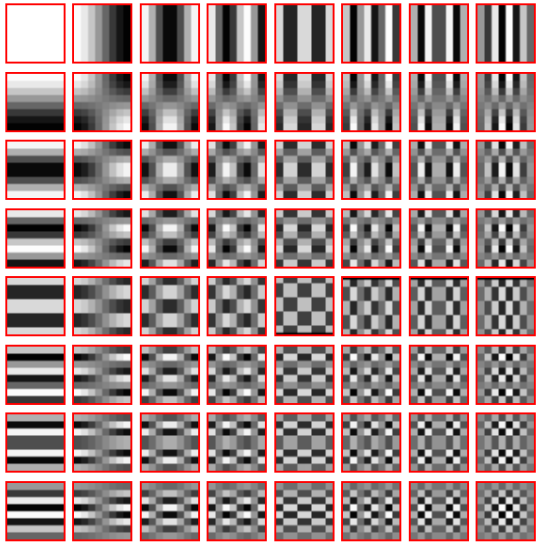


- Problème : il faut faire du **zero-padding** lorsque la taille de l'image n'est pas divisible par la taille du bloc :

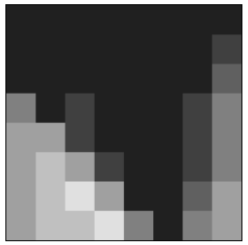


DCT *(Discrete Cosine Transform)*

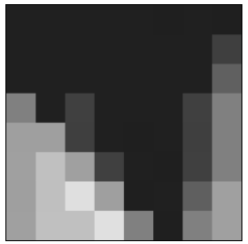
- Passage du domaine spatial en une représentation identique dans le domaine fréquentiel.
- Une image admet une grande continuité entre les valeurs des pixels : les hautes fréquences sont peu communes car elles traduisent des changements rapides d'intensité du pixel. Ainsi, on parvient à représenter l'intégralité de l'information de l'image sur très peu de coefficients, correspondant à des fréquences plutôt basses. La composante continue (valeur moyenne de l'image traitée) ayant une grande importance pour l'œil.
- Les basses fréquences se trouvent en haut à gauche de la matrice, et les hautes fréquences en bas à droite.



Les 64 images de base



Bloc initial (8x8)



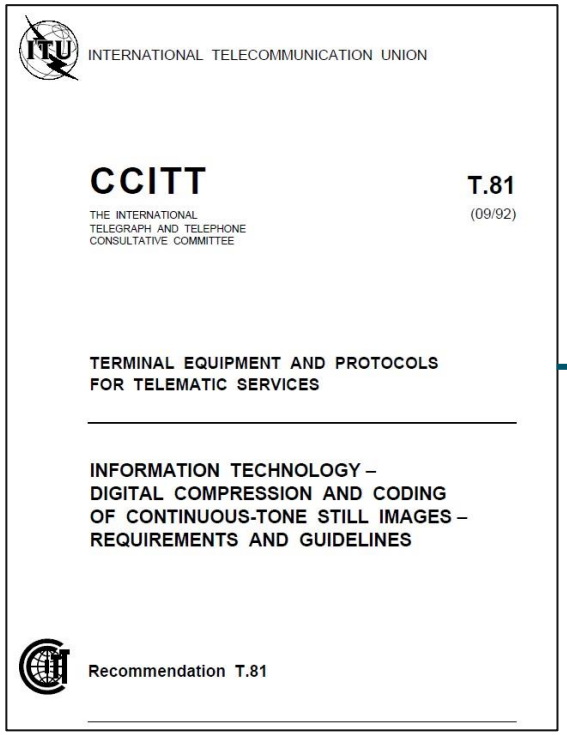
Bloc reconstitué

$$D(i,j) = \frac{1}{\sqrt{2N}} C(i)C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} p(x,y) \cos\left[\frac{(2x+1)i\pi}{2N}\right] \cos\left[\frac{(2y+1)j\pi}{2N}\right]$$

$$C(u) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } u = 0 \\ 1 & \text{if } u > 0 \end{cases}$$

Quantification / seuillage

- Phase non conservatrice du processus de compression : en moyennant une diminution de la précision de l'image, ont réussi à réduire le nombre de bits nécessaires au stockage.
- Diminution des coefficients issus de la DCT en les divisant par un nombre (*quantum*), fixé par une table de quantification (matrice 8 x 8).



INTERNATIONAL TELECOMMUNICATION UNION

CCITT

THE INTERNATIONAL TELEGRAPH AND TELEPHONE CONSULTATIVE COMMITTEE

T.81 (09/92)

TERMINAL EQUIPMENT AND PROTOCOLS FOR TELEMATIC SERVICES

INFORMATION TECHNOLOGY – DIGITAL COMPRESSION AND CODING OF CONTINUOUS-TONE STILL IMAGES – REQUIREMENTS AND GUIDELINES

Recommendation T.81

Page 143

Table K.1 – Luminance quantization table

Y

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Table K.2 – Chrominance quantization table

Cr,Cb

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

$$D = \begin{bmatrix} 162.3 & 40.6 & 20.0 & 72.3 & 30.3 & 12.5 & -19.7 & -11.5 \\ 30.5 & 108.4 & 10.5 & 32.3 & 27.7 & -15.5 & 18.4 & -2.0 \\ -94.1 & -60.1 & 12.3 & -43.4 & -31.3 & 6.1 & -3.3 & 7.1 \\ -38.6 & -83.4 & -5.4 & -22.2 & -13.5 & 15.5 & -1.3 & 3.5 \\ -31.3 & 17.9 & -5.5 & -12.4 & 14.3 & -6.0 & 11.5 & -6.0 \\ -0.9 & -11.8 & 12.8 & 0.2 & 28.1 & 12.6 & 8.4 & 2.9 \\ 4.6 & -2.4 & 12.2 & 6.6 & -18.7 & -12.8 & 7.7 & 12.0 \\ -10.0 & 11.2 & 7.8 & -16.3 & 21.5 & 0.0 & 5.9 & 10.7 \end{bmatrix}$$

$$C = \begin{bmatrix} 10 & 4 & 2 & 5 & 1 & 0 & 0 & 0 \\ 3 & 9 & 3 & 2 & 1 & 0 & 0 & 0 \\ -7 & -5 & 3 & -2 & -1 & 0 & 0 & 0 \\ -3 & -5 & 0 & -1 & 0 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$C_{i,j} = round\left(\frac{DCT_{i,j}}{Q_{i,j}}\right)$$

→ Nombreux coefficients égaux à 0. Les coefficients DCT de petite valeur (moitié inférieure droite) sont divisés par des coefficients de quantifications élevés. L'arrondi nous donne une valeur nulle.

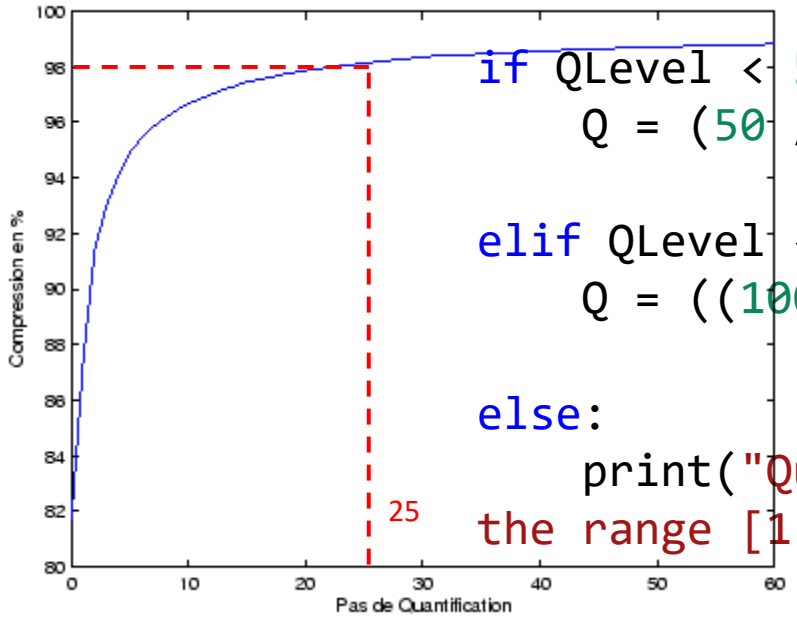
Quantification / seuillage

- On choisit au début le niveau de quantification (entre 1 et 99) :

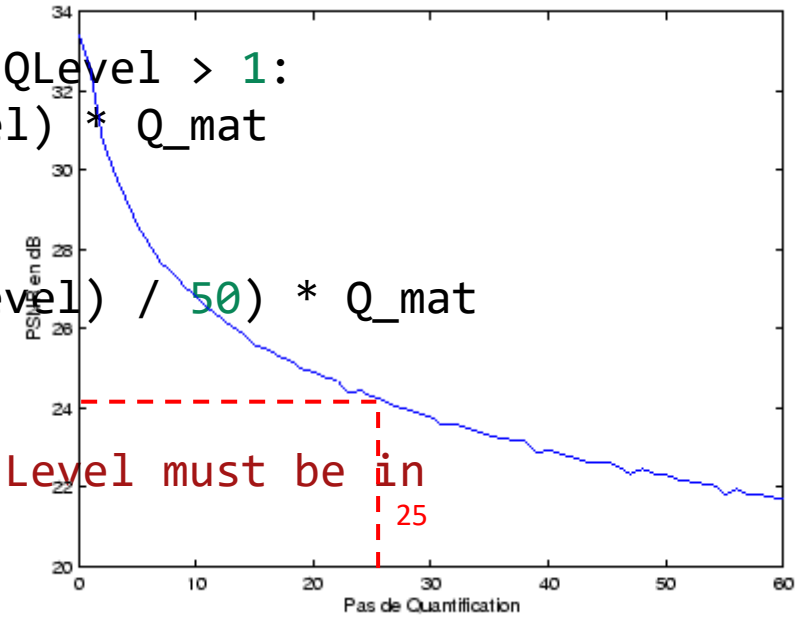
```
if QLevel < 50 and QLevel > 1:
    Q = (50 / QLevel) * Q_mat

elif QLevel < 100:
    Q = ((100 - QLevel) / 50) * Q_mat

else:
    print("Quality Level must be in
the range [1..99]")
```



Taux de Compression en fonction du Pas de Quantification

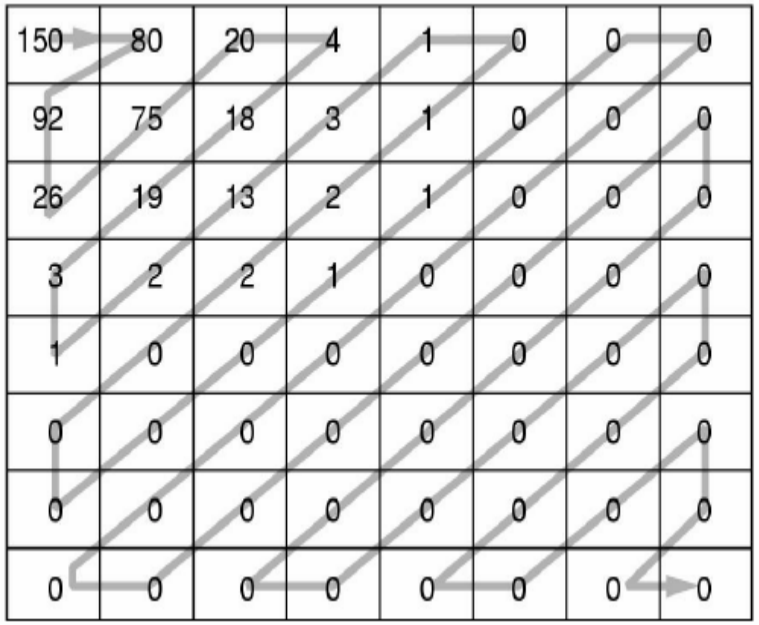


Rapport Signal sur Bruit (PSNR : Peak Signal To Noise Ratio) en fonction du Pas de Quantification

- On estime qu'il ne faut pas dépasser un facteur de 25, après quoi l'image est trop dégradée : le gain en terme de nombre de bits utilisés pour coder l'image devient négligeable.

Zig-zag scanning

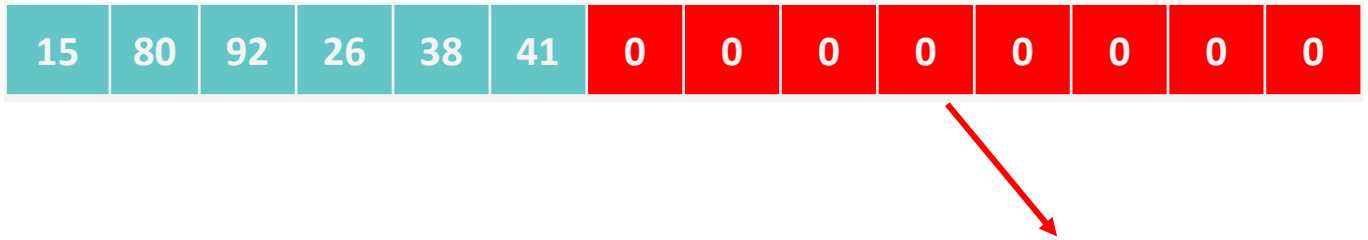
- Nous avons réussi à générer beaucoup de zéros !
- L'information à coder est redondante, ce qui va nous permettre d'obtenir un codage plus efficace. Le zig-zag tire parti de cette redondance.
- L'objectif est de « sérialiser » un maximum de coefficients nuls afin d'améliorer le taux de compression. On condense l'information au début :



Zig-zag →

150, 80, 92, 26, 75, 20, 4, 18, 19, 3, 1, 2, 13, 3, 1, 0, 1, 2, 2, 0, 0, 0, 0, 0, …

Run Length Encoding (*RLE*)



- Peu avantageux au début, voir même contre-productif, car on rajoute de nouveaux éléments, mais ensuite on condense beaucoup l'information dès qu'il y a une longue série de zéros à la suite.

Codage de Huffman

- Ce codage attribut des mots (une séquence binaire) à chaque couple rencontré en sortie du RLE.
- Le code est déterminé à partir d'une estimation des probabilités d'apparition des symboles de source, un code court étant associé aux symboles de source les plus fréquents.

```
(15, 3),(80, 2),(0, 8),(38, 8),(41, 1),(0, 8),(15, 1),(0, 8),(14, 8),(0, 8),(14, 9)
```

Symbole de source	Probabilité d'apparition du symbole	Mot de code	Longueur du mot de code
(0, 8)	**0,364**	**'11'**	**2**
(14, 9)	0,091	'100'	3
(15, 1)	0,091	'010'	3
(14, 8)	0,091	'011'	3
(38, 8)	0,091	'000'	3
(41, 1)	0,091	'001'	3
(15, 3)	0,091	'1010'	4
(80, 2)	0,091	'1011'	4

Encodé → 10101011110000011101011011111100

En écrivant ce flux dans un ficher binaire, on peut calculer le taux de compression en fonction de la taille de notre image d'origine en bits.

Problèmes à résoudre

- Décodage : *comment retrouver les "frontières" d'un bloc dans le flux binaire qui est de longueur variable ?* Pour chaque bloc, on propose d'ajouter un marqueur de fin qui est une séquence qui ne se reproduit pas dans le flux binaire (par exemple « EOB » pour *End-of-Block*).

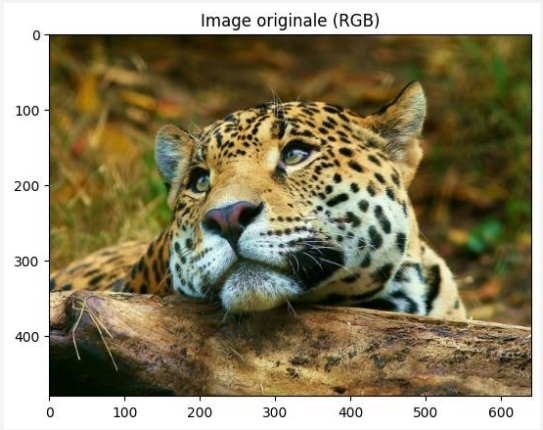
Lorsqu'on avance ou recule dans un film avec la télécommande, on balaye très rapidement les marqueurs de fin de bloc contenus dans le film.

- Deux stratégies peuvent êtres adoptées, pour transmettre la table des fréquences du code entropique de manière :
  - **dynamique** : avec une table différente en fonction des séquences de la vidéo,
  - **statique :** on transmet la table des fréquences une seule fois avec les données compressés. Il faut déterminer qu'elles sont les paires (issues de l'opération de *Run Length Encoding*) les plus fréquentes pour différentes vidéos. Cela nécessite une étude statistique de vidéos ayant des contextes et une dynamique différentes. On tolèrera alors une certaine erreur en approximant notre table des fréquences calculées sur la base de la moyenne des paires les plus observées.

Reste à faire

- Décompresser l'information encodée avec Huffman (la décompression fonctionne sans passer par cette étape).
- Optimiser le code de Huffman.
- Améliorer le taux de compression en déterminant le facteur de quantification optimal et/ou en jouant sur d'autres paramètres.
- De manière générale, réfléchir à des manières d'améliorer l'algorithme (en terme de performance et de complexité).
- Appliquer l'algorithme réalisé de compression/décompression dans un contexte vidéo.
- Réfléchir à une implémentation sur FPGA et à la mise en place de la caméra OV7670 pour obtenir un flux vidéo.

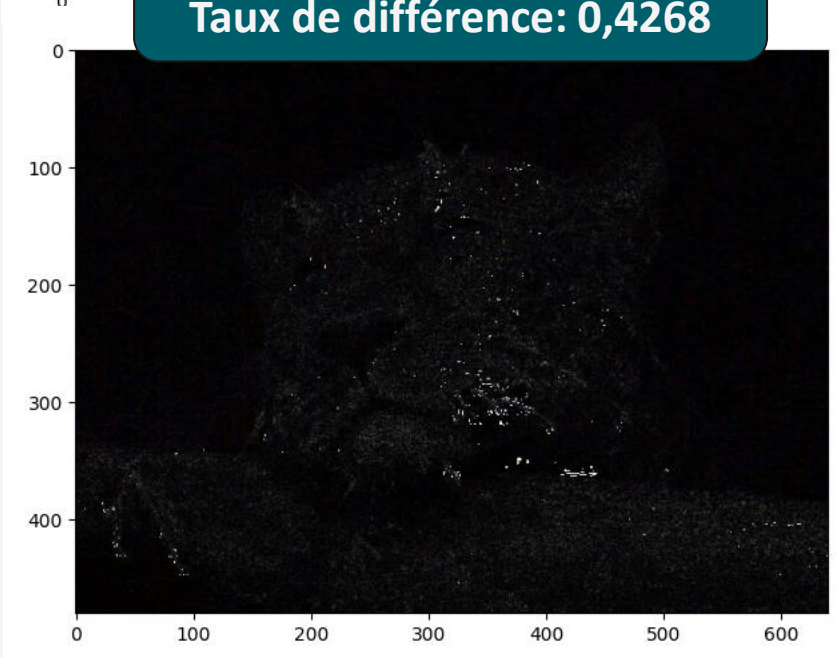
Résultats obtenus (*QLevel = 20*)



Y

Cr

Taux de différence: 0,4268



Démonstration & Questions