SOUTENANCE : Conception FPGA d’un compresseur vidéo

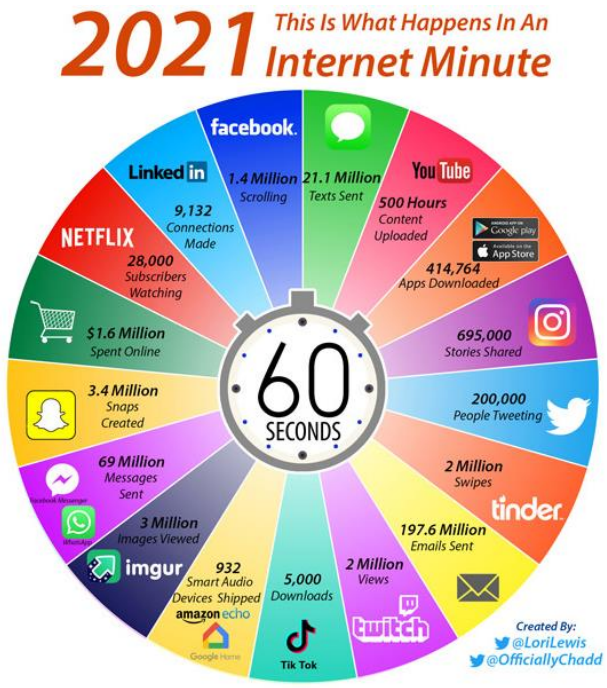
Vincent Robin : FIPA 3A, Promotion 2023, Option SE

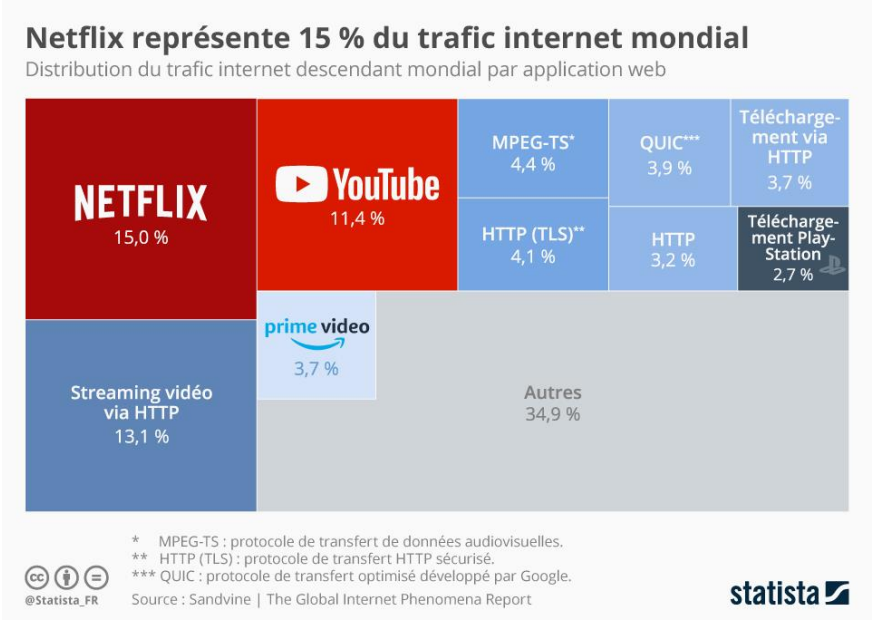
Février 2023

ENSTA BRETAGNE

Contexte du projet

/ Explosion de la consommation de vidéos sur internet, consommatrices de bande passante :





TB

1 **Terraoctets** : 300h de vidéo de bonne qualité

PB

2 **Petaoctets** : Toutes les bibliothèques universitaires américaines

20 Petaoctets : Quantité de données traitées quotidiennement par Google

200 Petaoctets : Tous les documents imprimés

EB

5 **Exaoctets** : Tous les mots jamais prononcés par des êtres humains

15 Exaoctets : total des données détenues par Google en 2013

ZB

180 **Zettaoctets** d'ici 2025 = 5 allers-retours sur la lune de pile de CD !!

→ **L'élaboration de normes de compression vidéo efficaces revêt une importance cruciale !**

Objectifs du projet :

- Elaborer un modèle de référence de compression/décompression intelligible dans le langage de son choix,

- offrir la possibilité de générer des images synthétiques,

- évaluer les performances de l'algorithme de compression/décompression,

- proposer des pistes d'amélioration,

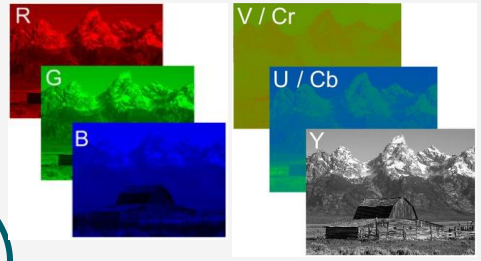
- commencer les travaux sur une séquence vidéo,

- appréhender la programmation VHDL d'un tel compresseur.

Génération d'une image simple au format PPM

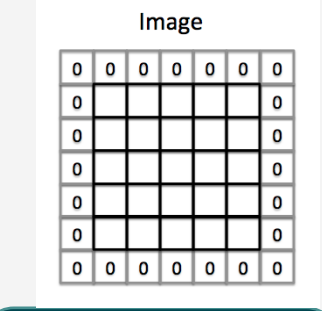
Lecture d'une image en couleurs (RGB) au format .png, .jpg, …

Flux vidéo (format VGA : 480x640)

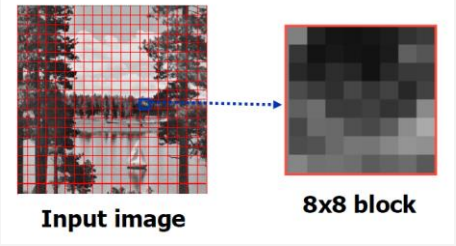


Conversion RGB → YCrCB

Chroma sub-sampling → format 4:2:0



Zero-padding (si nécessaire)

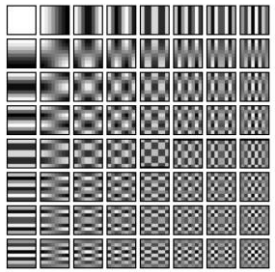


Découpage en blocs 8x8 pour chaque canal (Y,Cr,Cb)

COMPRESSION

$$\tau = 1 - \frac{t_c}{t_u}$$

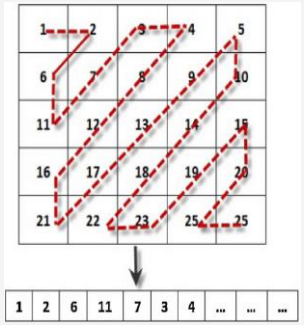
DCT (*Discrete Cosine Transform*)



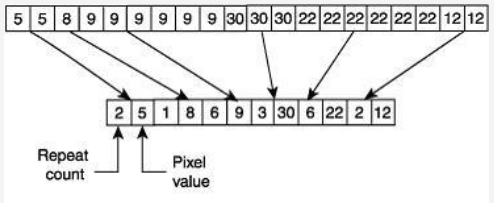
Quantification

Tables de quantification

Zig-zag



Run Length Encoding (RLE)

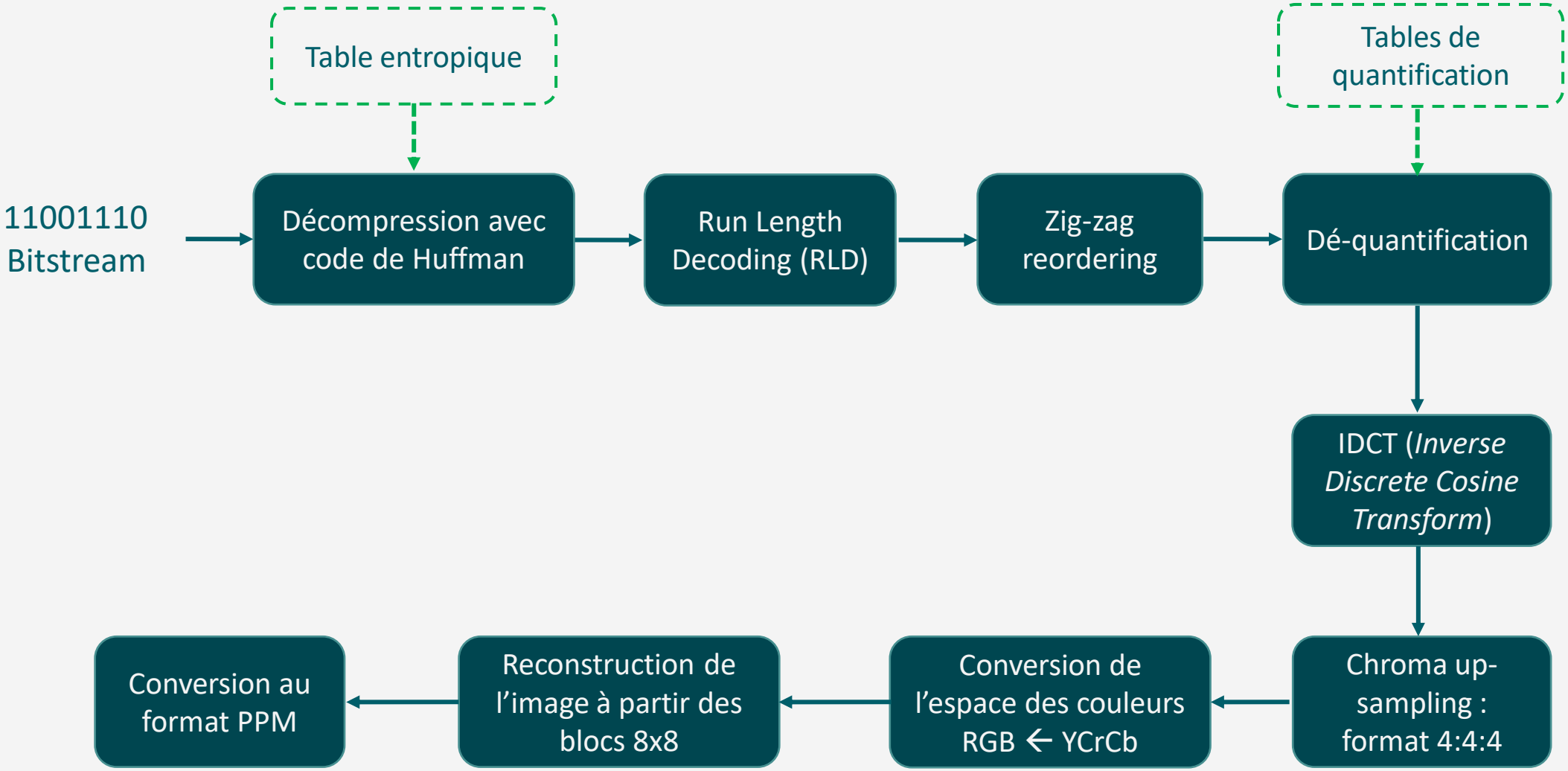


Huffman : code à longueur variable

Spécification de la table entropique

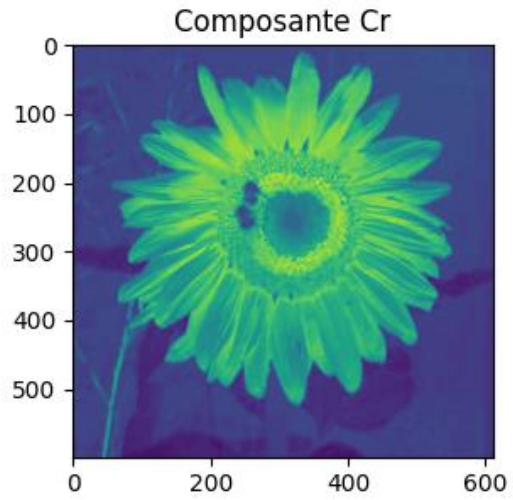
11001110
Bitstream

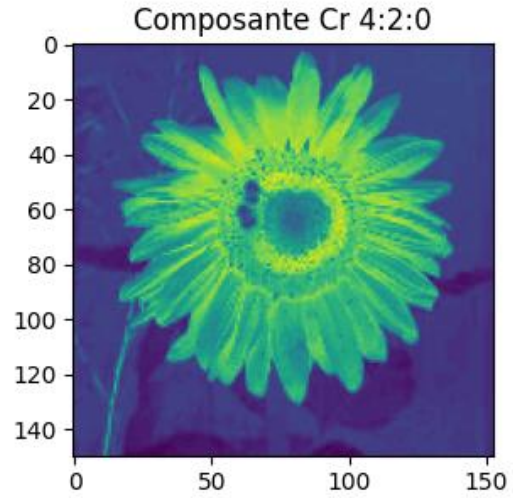
Processus de décompression



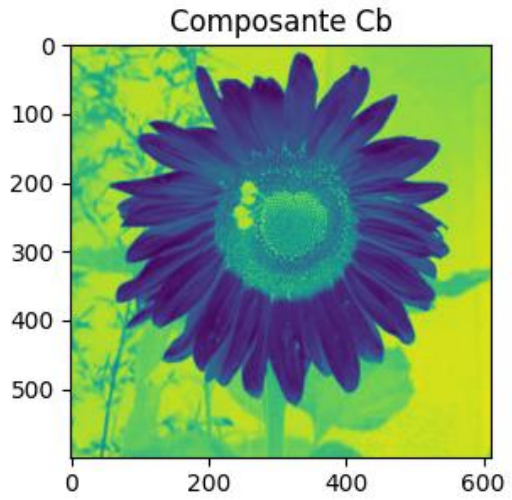
Sous échantillonnage de la chrominance

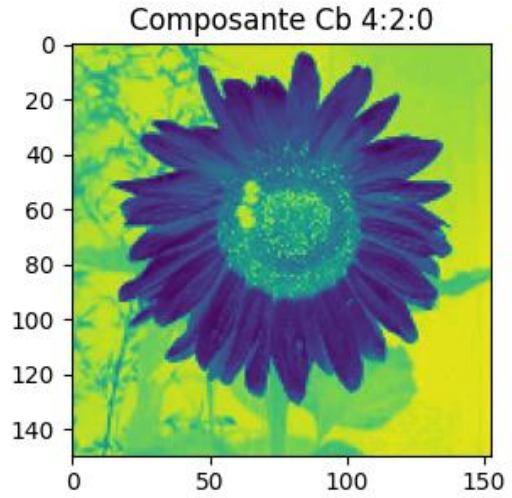
4:2:0





Voyez-vous une différence ?

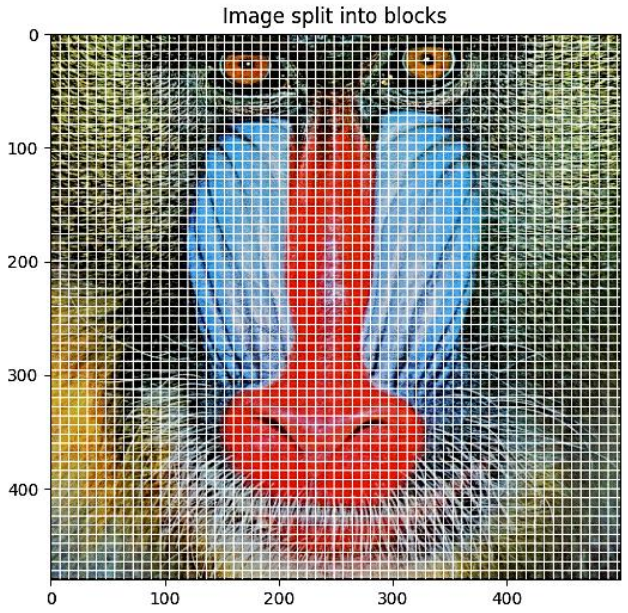


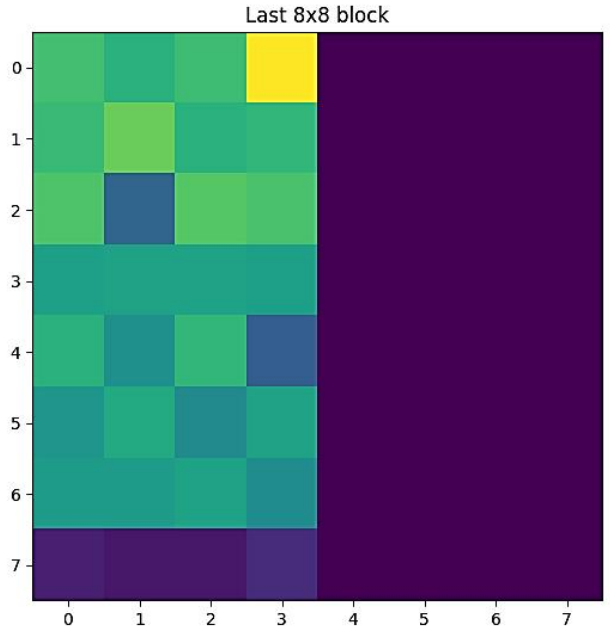


4:2:0

Découpage de l'image en blocs de taille 8*8

- Pourquoi ne pas travailler avec des blocs plus grands ?
  - Raisons historiques : on travaille depuis longtemps sur des blocs 8*8 de 64 pixels
  - Rapidité de calcul et précision
  - Corrélation entre pixels : 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 :

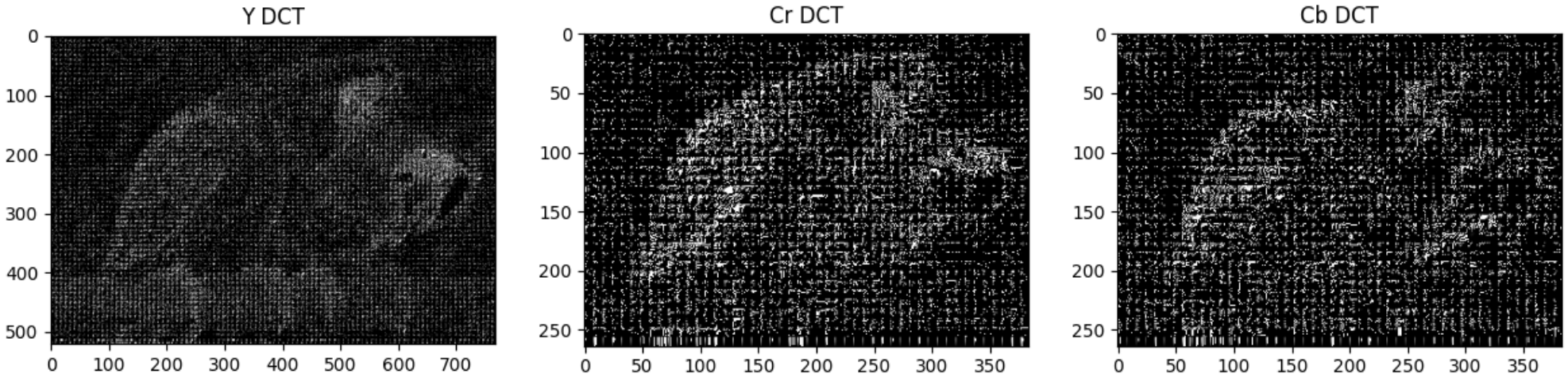





```
Height: 480, Width: 500
Block size:  8
Number of blocks in height (nbh):  60
Number of blocks in width (nbh):  63
Height of padded image:  480
Width of padded image:  504
⟶ Zero padding is required!
```

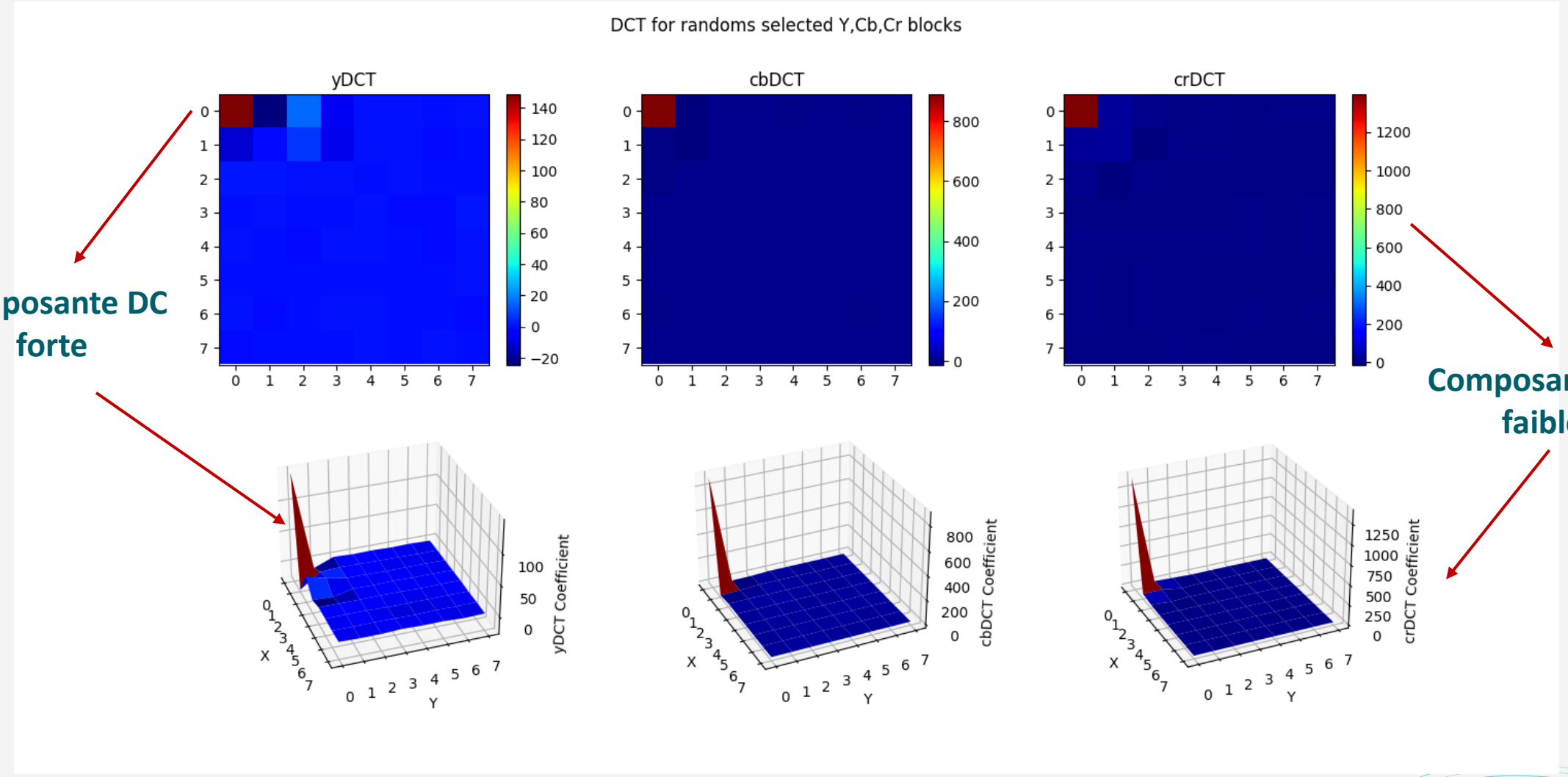
DCT *(Discrete Cosine Transform)*

- Le principe de cette approche est de projeter les données de l'image initiale dans un espace où l'information pertinente se retrouve concentrée dans un nombre limité de coefficients à fortes valeurs.
- La transformée est appliquée indépendamment à la composante de luminance et aux 2 composantes de chrominance :



- 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 DC (valeur moyenne de l'image traitée) ayant une grande importance pour l'œil.

DCT *(Discrete Cosine Transform)*



Composante DC forte

Composantes AC faibles

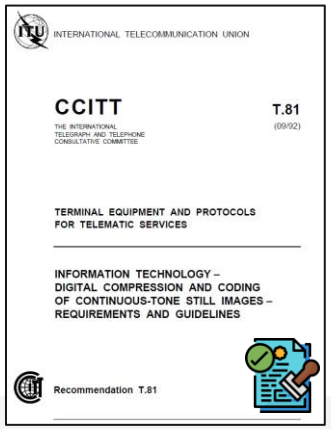
Quantification

- **La transformation DCT ne comprime pas, au contraire, puisque les coefficients transformés sont réels, contrairement aux valeurs entières originales. La quantification est nécessaire au moins pour cette raison !**
- 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

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

Table K.1 – Luminance quantization table

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



Tables de quantification (pour Y et Cr,Cb)

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

$$=$$

$$E_{i,j} = D_{i,j} - C_{i,j} * Q_{i,j}$$

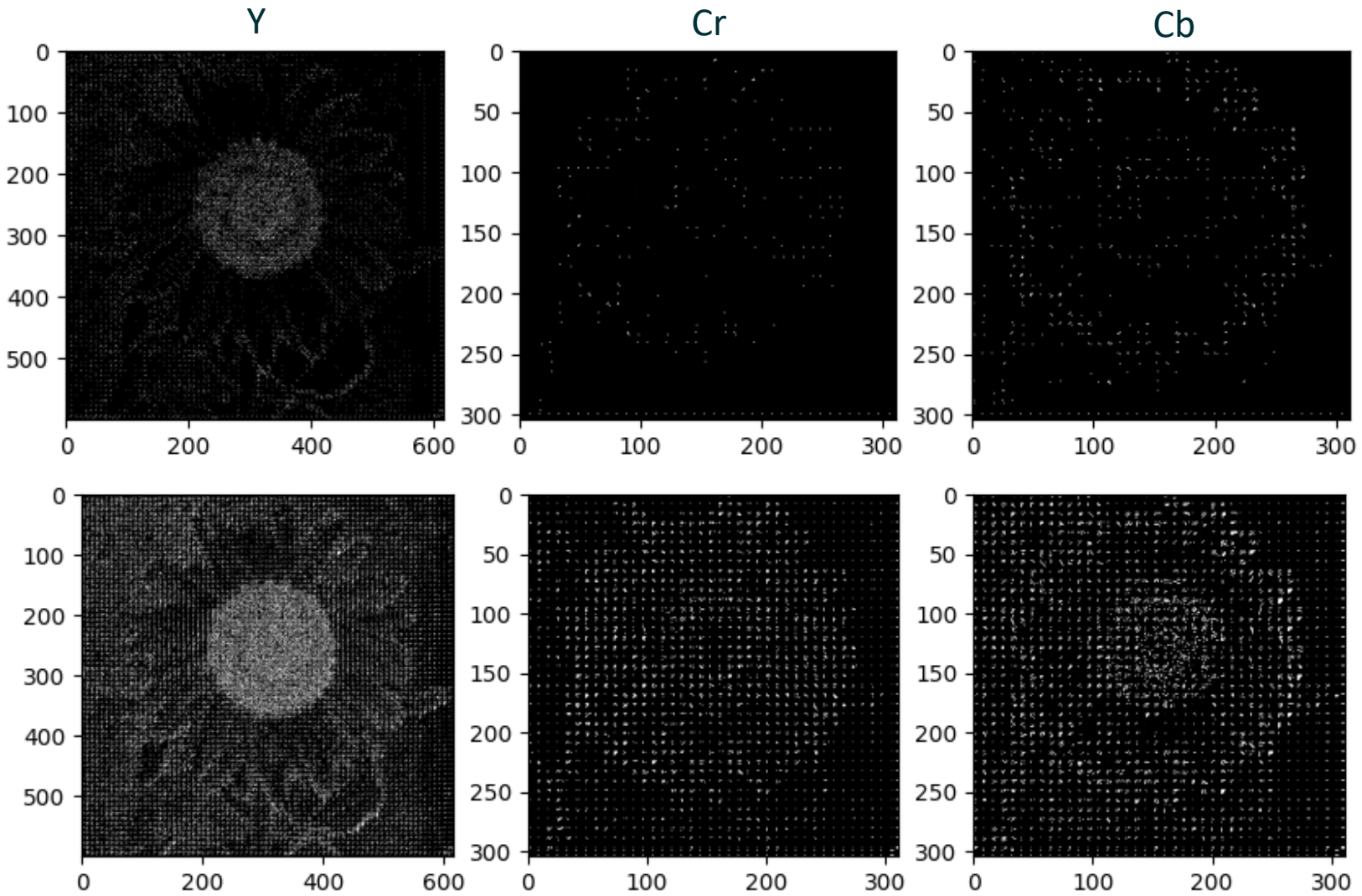
$$C = \begin{bmatrix} 10 & 4 & 2 & 5 & 1 & 0 & 0 & 0 \\ 3 & 9 & 3 & 2 & 1 & 0 & 0 & 0 \\ -7 & -5 & 3 & -2 & -1 & 0 & 0 & 0 \\ -3 & -5 & 0 & -1 & 0 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Les coefficients DCT faibles sont divisés par des coefficients de quantifications élevés : beaucoup de zéros apparaissent.
L'arrondi introduit une erreur de quantification $E_{i,j}$

Quantification

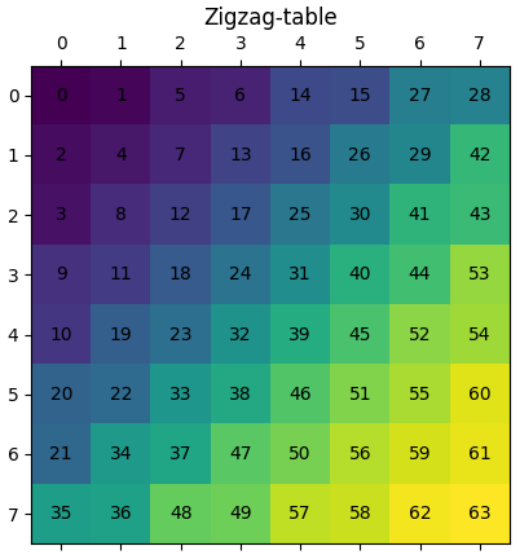
QLevel = 10

QLevel = 80

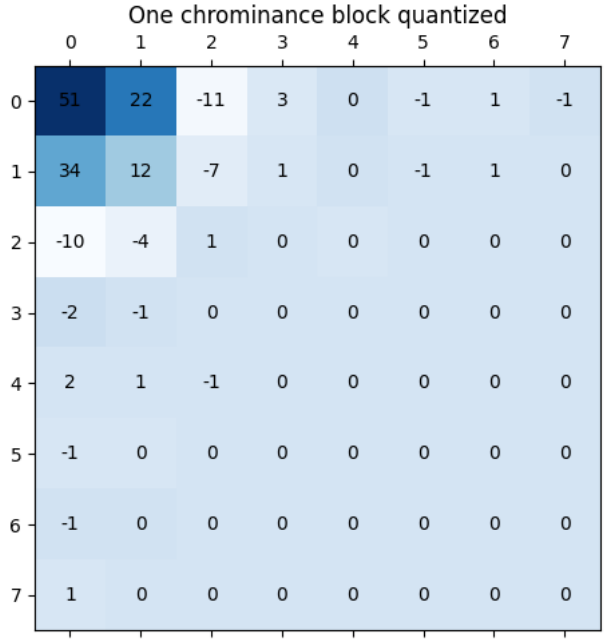


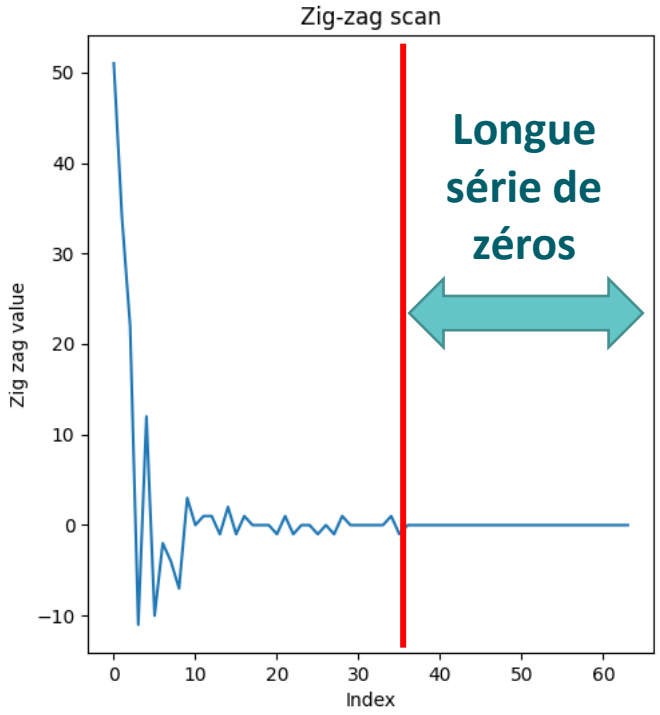
Relecture en zig-zag

- 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 :

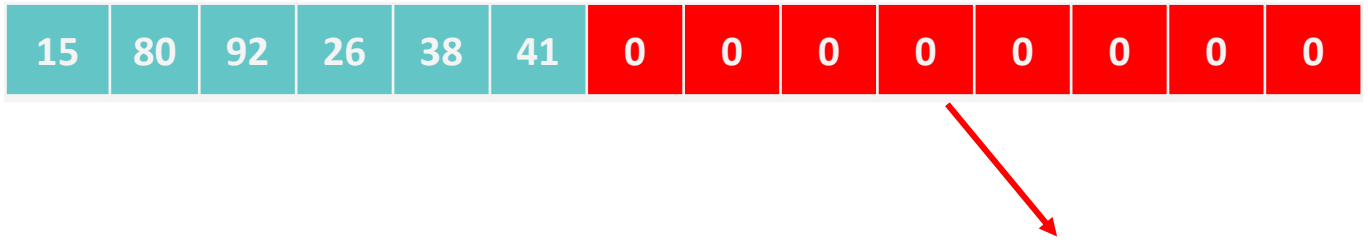


Relecture du bloc 8*8 en diagonal selon une séquence en zig-zag





Run-Length Encoding (*RLE*)



(15, 1) (80, 1) (92, 1) (26, 1) (38, 1) (41, 1) **(0, 8)**

- 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

Deux possibilités s'offrent à nous :

- Adresser le dictionnaire d'Huffman de manière dynamique → **1 table = 1 image**. Le code de Huffman est optimal, mais on doit transmettre les dictionnaires à chaque fois au décompresseur.
- Utiliser les tables de Huffman du standard JPEG → **1 table = toutes les images**. Le code de Huffman n'est pas optimal car il n'est pas propre à l'image, mais on n'a pas besoin de transmettre les tables au décompresseur.

Table K.3 – Table for luminance DC coefficient differences

Category	Code length	Code word
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

Table K.5 – Table for luminance AC coefficients (sheet 1 of 4)

Run/Size	Code length	Code word
0/0	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
0/5	5	11010
0/6	7	1111000
0/7	8	11111000
0/8	10	1111110110
0/9	16	1111111110000010
0/A	16	1111111110000011
1/1	4	1100
1/2	5	11011
1/3	7	1111001

Codage de Huffman

Huffman analysis:

Luminance Y				Chrominance Cr				Chrominance Cb			
Symbol	Proba	Code	Length	Symbol	Proba	Code	Length	Symbol	Proba	Code	Length
(-1, 0)	0.09	0011	4	(-1, 0)	0.091	011	3	(-1, 0)	0.117	0001	3
(1, 0)	0.089	1111	4	(1, 0)	0.09	010	3	(1, 0)	0.115	1111	3
(-2, 0)	0.048	1110	4	(2, 0)	0.048	0000	4	(-2, 0)	0.047	1110	4
(2, 0)	0.047	0010	4	(-2, 0)	0.048	00100	5	(2, 0)	0.041	0011	5
(3, 0)	0.039	01100	5	(1, 1)	0.034	11001	5	(1, 1)	0.038	10100	5
(-3, 0)	0.039	11000	5	(-3, 0)	0.032	11010	5	(-1, 1)	0.038	10001	5
(-4, 0)	0.026	01000	5	(3, 0)	0.028	00011	5	(3, 0)	0.024	01111	5
(-1, 1)	0.024	11001	5	(-4, 0)	0.019	001011	6	(1, 2)	0.021	011100	6
(1, 1)	0.024	01101	5	(4, 0)	0.018	110111	6	(-1, 2)	0.021	110010	6
Total	0.454			Total	0.440			Total	0.486		
Entropy			6.8	Entropy			7.2	Entropy			6.7

Comment déterminer le taux de compression ?

- Avant compression : on sait qu'un pixel est codé sur 8 bits (valeur comprise entre 0 et 255) et qu'il y a 3 canaux (Rouge, Vert, Bleu)

$$N_{bits} = N_{pixels} * 8\ bits$$

$$avec, \qquad N_{pixels} = Hauteur\ * Largeur\ * 3$$

- Après compression : le nombre de bits total est égal à la taille du fichier **binaire** contenant les données comprimées + la taille des dictionnaires de Huffman pour les signaux Y, Cr, Cb

- **Taux de compression :**

$$\tau = \frac{N_{bits}\ avant\ compression}{N_{bits}\ \ après\ compression}$$

Décompression

La décompression réalise de manière stricte, en sens inverse, la suite des opérations effectuées par le codeur.

Le décodeur doit avoir connaissance des paramètres choisis par le codeur :

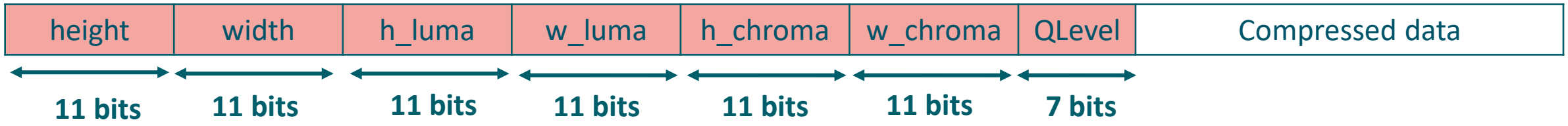
- Le **niveau de qualité** utilisé à la compression,
- le **code binaire à longueur variable** (VLC) : transmission du dictionnaire de Huffman.

Il doit aussi connaître les paramètres utiles pour la reconstruction de l'image finale :

- la hauteur (*height*) et la largeur (*width*) de l'image d'origine (exprimée en termes de pixels),
- La hauteur et la largeur de la composante de luminance (*h_luma* et *w_luma*) et des deux composantes de chrominance (*h_chroma* et *w_chroma*)

Comment adresser ces informations au décodeur ?

→ Création d'un en-tête dans le fichier binaire, avant les données compressés :



Evaluation des performances : métriques utilisées

Mean squared error (MSE) :

L'erreur quadratique moyenne entre les deux images est la somme de la différence au carré entre l'image d'origine Y_i et l'image reconstruite $\widehat{Y_i}$.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{Y_i})^2$$

Plus cette valeur est petite, plus les images sont similaires.

Signal to Noise Ratio (SNR) :

Le rapport signal à bruit s'exprime en décibels (dB) par :

$$SNR_{dB} = 10 * log_{10}\left(\frac{P_{signal}}{P_{bruit}}\right)$$

La puissance du bruit correspond au MSE.

Plus les images sont différentes, plus le SNR tend vers 0.

Entropie

Elle mesure la quantité d'information dans une source S, c'est-à-dire, l'information moyenne par symbole de la source. Elle s'exprime en bits/symbole :

$$H(S) = -p_i * log_2(p_i)$$

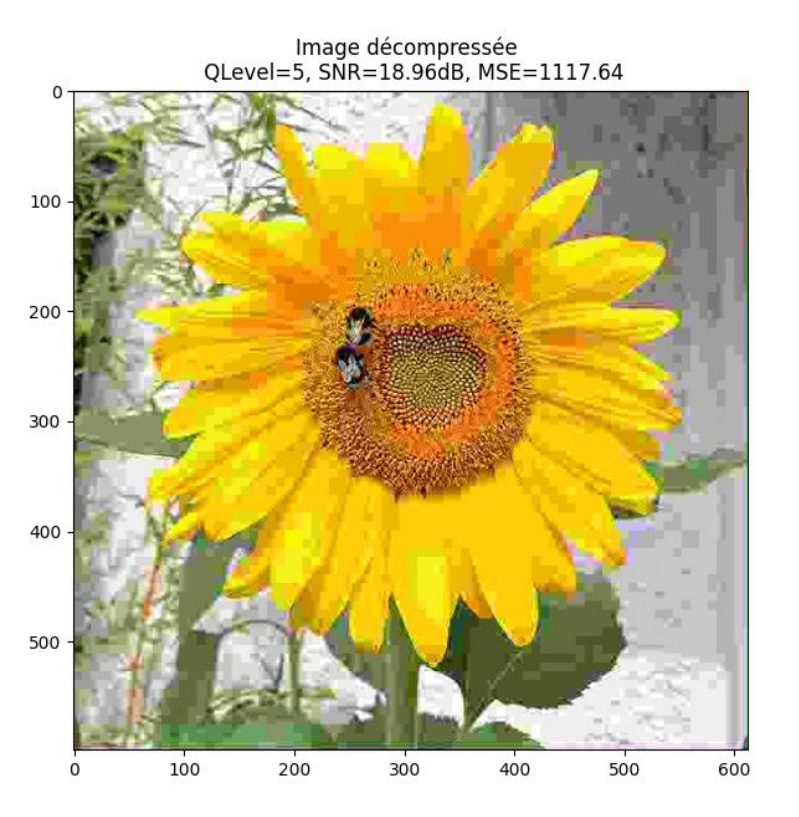
Taux de compression

$$\tau = \frac{N_{bits}\ avant\ compression}{N_{bits}\ \ après\ compression}$$

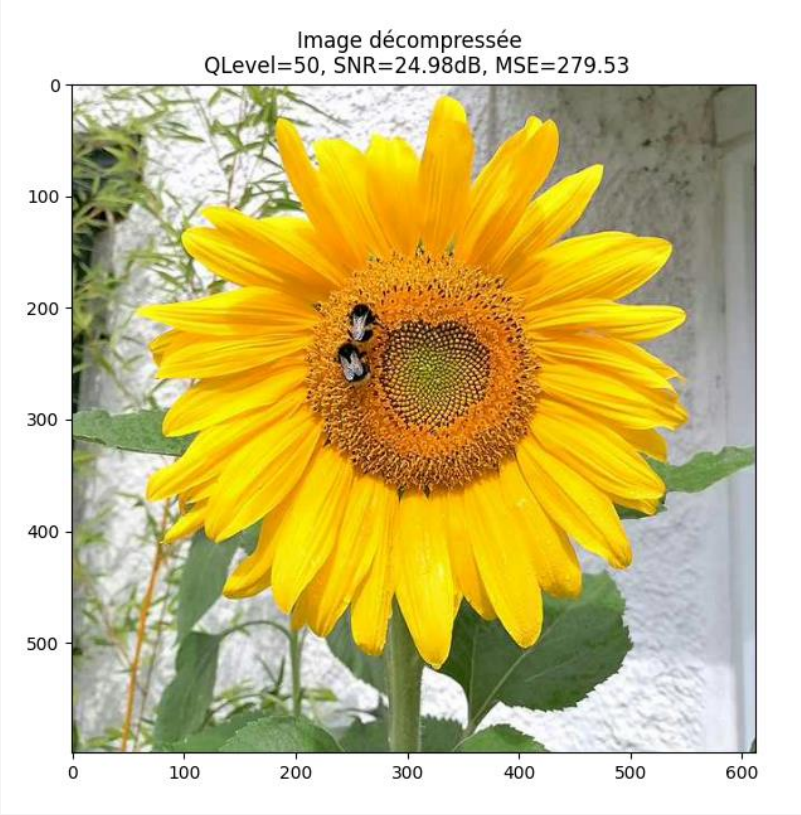
Temps d'exécution

(à la décompression)

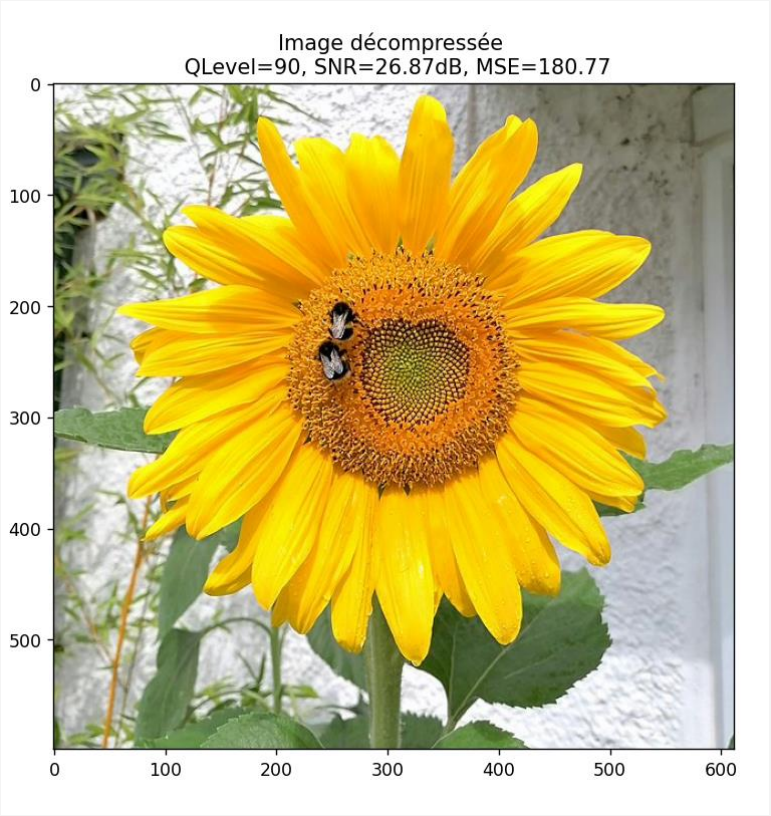
Résultats obtenus en fonction du niveau de quantification



QLevel = 5 : « effets de blocs »



QLevel = 50 : peu de différences visibles par l'œil humain

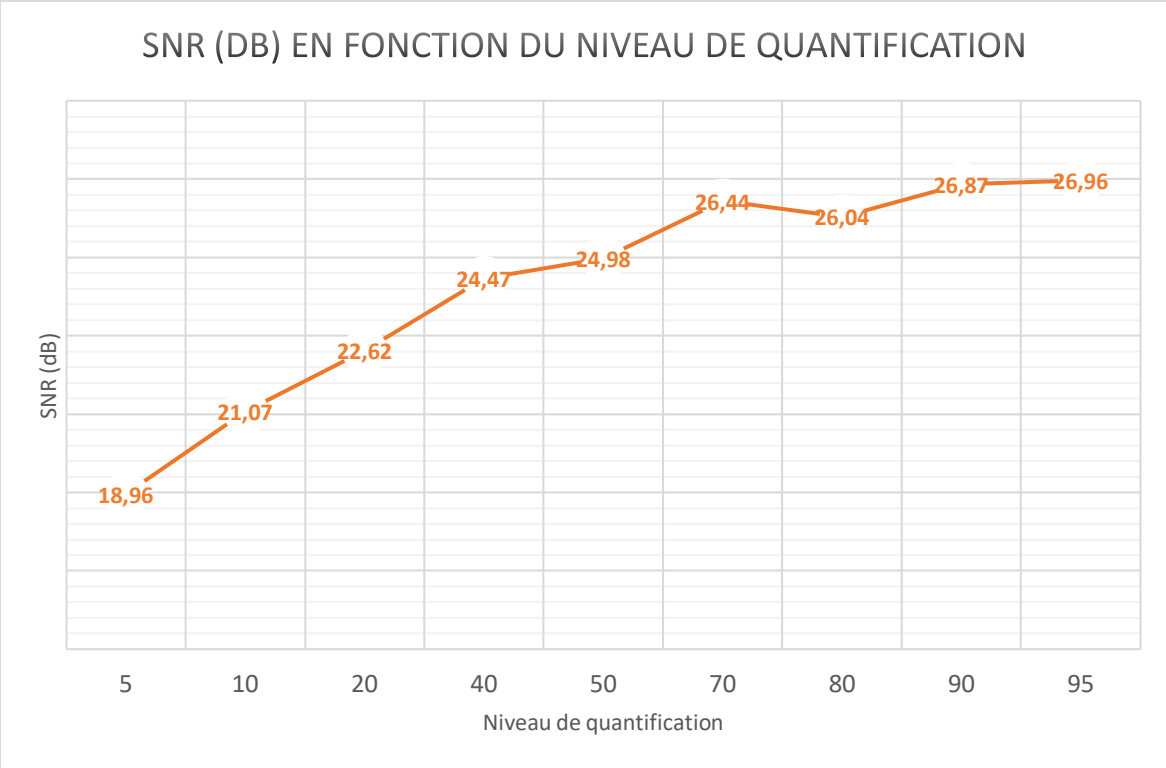


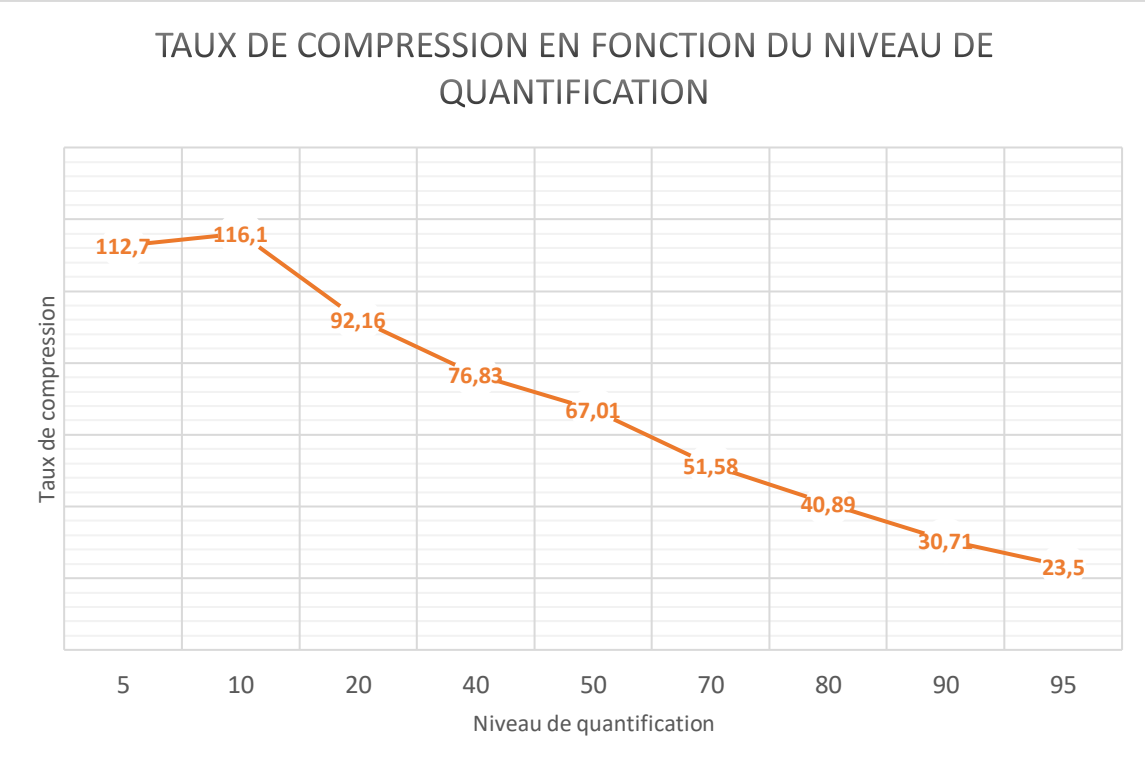
QLevel = 90 : aucunes différences visibles

Résultats obtenus en fonction du niveau de quantification

QLevel =	5	10	20	40	50	70	80	90	95
MSE	1117.64	687.77	480.47	314.19	279.53	199.40	218.72	180.77	176.96
SNR (dB)	18.96	21.07	22.62	24.47	24.98	26.44	26.04	26.87	26.96
Entropie (bits/symbole)	Y=7 Cr= 5.3 Cb= 6 moy=6.1	Y=6,5 Cr= 4.7 Cb= 5.9 moy=5.7	Y=6.4 Cr= 5.5 Cb= 6.2 moy=6.0	Y=6 Cr= 6.2 Cb=6.4 moy=6.2	Y=6 Cr= 6.3 Cb=6.5 moy=6.3	Y=6.2 Cr= 6.5 Cb=6.7 moy=6.5	Y=5.8 Cr= 6.6 Cb=6.7 moy=6.4	Y=6.8 Cr= 7.2 Cb=6.7 moy=6.9	Y=7.6 Cr= 7.4 Cb=6.9 moy=7.3
Taux de compression	112.70	116.10	92.16	76.83	67,01	51.58	40.89	30.71	23.5
Temps d'exécution (décompression)	7.24	8.18	12.07s	13.23s	16.59s	24.60s	34.83	70.56	155.98

Résultats obtenus en fonction du niveau de quantification





Programmation en VHDL

- Quelques briques VHDL ont déjà été faites : DCT, RLE, ZigZag
- Codes des testbenches fournis
- Définition des types et des fonctions dans des packages

PACKAGE

```
library ieee;
use ieee.std_logic_1164.all;

package pkg is
    -- CONSTANT
    constant BLOCK_SIZE: natural := 8;
    -- TYPES
    type array_1D        is array (0 to BLOCK_SIZE-1) of integer;
    type one_block       is array (0 to BLOCK_SIZE-1) of array_1D;
    type block_flatten   is array (integer range <>) of integer;
    -- FUNCTION
    function zig_zag (img : one_block) return block_flatten;

end package;


package body pkg is

    function zig_zag (img : one_block) return block_flatten is

     -- body of zigzag function

    end zig_zag;

end package body;
```

TXT

STIMULIS

```
10 4 2 5 1 0 0 0
3 9 1 2 1 0 0 0
-7 -5 1 -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
```

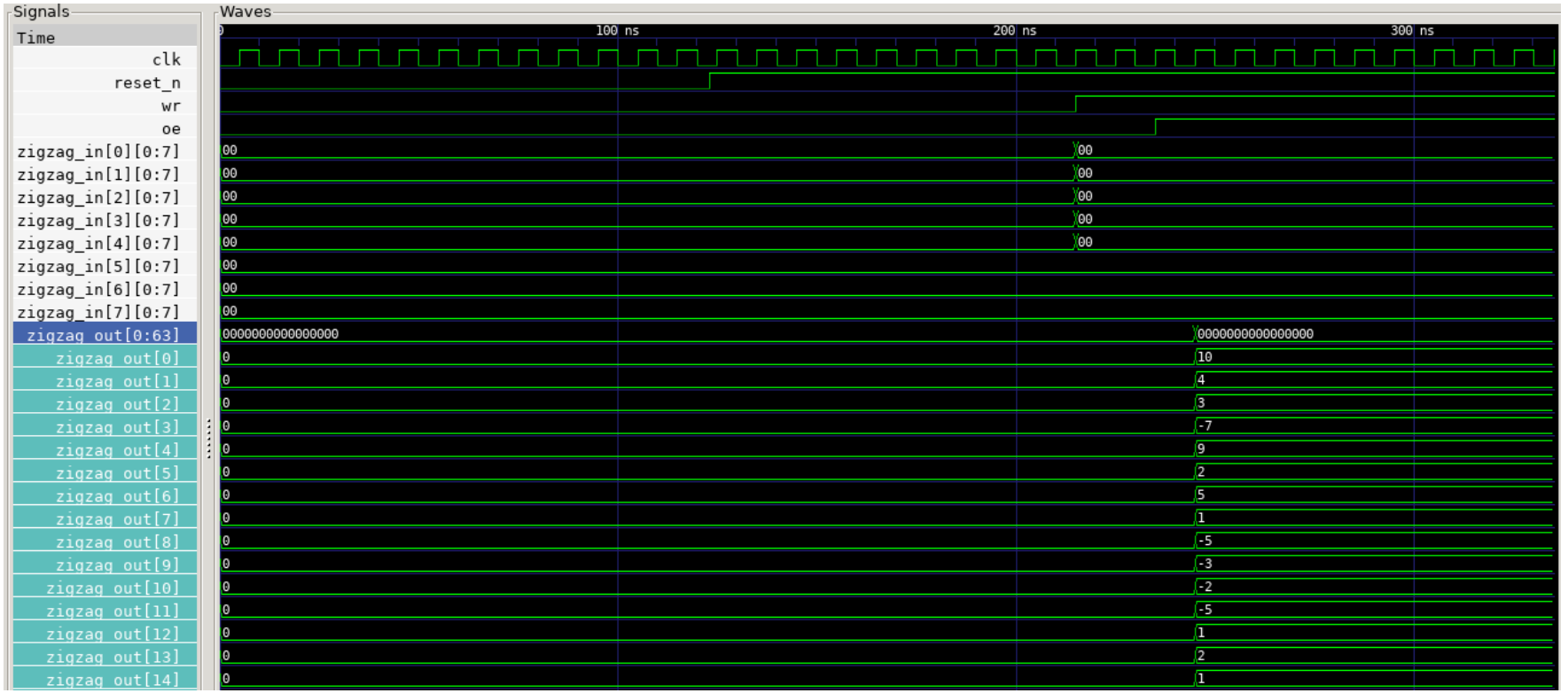
TESTBENCH

```
stim : process

    file INFILE:  text open read_mode is "input.txt";
    variable input_block: one_block;
    variable INPUT_LINE: line;
    variable nb_lines: natural := 0;

  begin
    report "running testbench for zigzag_tb(arch)";
    oe ← '0';
    wr ← '0';
    report "waiting for asynchronous reset";
    wait until reset_n = '1';
    wait_cycles(10);
    while not endfile(INFILE) loop
        nb_lines := nb_lines + 1;
        for i in 0 to (number_of_inputs-1) loop
            readline(INFILE, INPUT_LINE);
            for j in 0 to (number_of_inputs-1) loop
                read(INPUT_LINE,input_block(i)(j));
            end loop;
        end loop;
    end loop;
    file_close(INFILE);
    wr ← '1';
    zigzag_in ⩽ input_block;
    wait_cycles(2);
    oe ← '1';
    wait_cycles(10);
    report "end of simulation";
    running ← false;
    wait;
  end process;

end bhv;
```

Programmation en VHDL

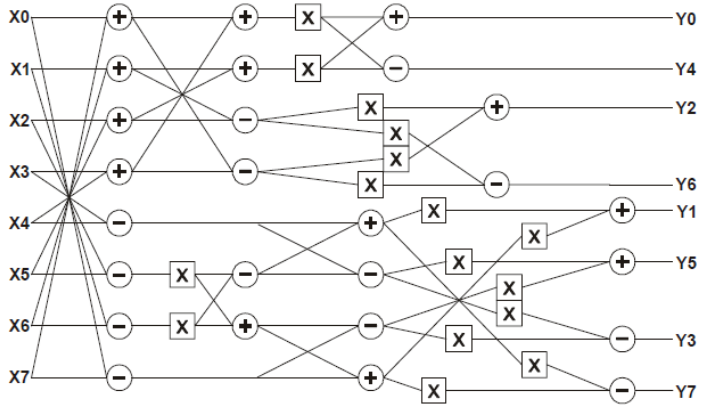
- Environnement logiciel simple (*GHDL + GTKWave*),
- Scripts Bash de compilation/élaboration

```
echo "⇒ cleaning"
rm -rf work*.cf *.o

echo "⇒ analysis package"
ghdl -a --std=08 pkg.vhd

echo "⇒ analysis design file"
ghdl -a --std=08 rle.vhd

echo "⇒ analysis test bench"
ghdl -a --std=08 rle_tb.vhd

echo "⇒ elaboration"
ghdl -e --std=08 rle_tb

echo "⇒ run"
ghdl -r --std=08 rle_tb
--wave=rle_tb.ghw

echo "⇒ result analysis"
gtkwave rle_tb.ghw rle_tb.sav
```

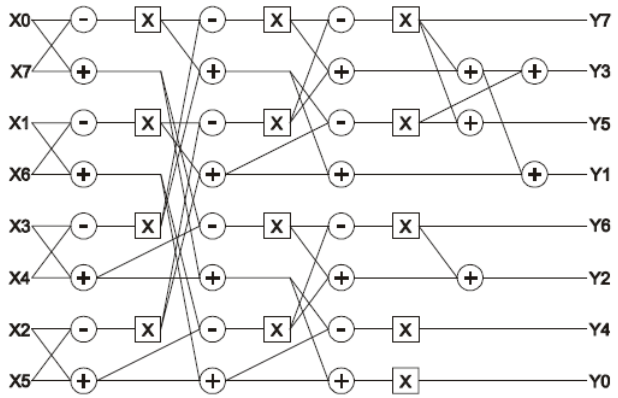


Programmation en VHDL

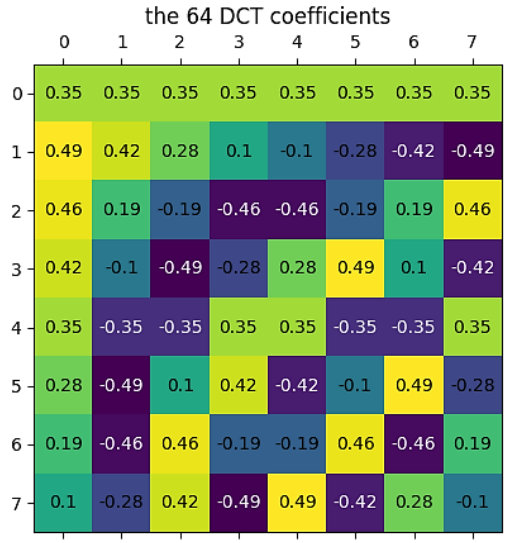
- Pour la DCT, des algorithmes de calculs spécifiques existent (*Fast-DCT algorithms*) pour être implémentés en hardware sur FPGA :



Chen's algorithm flowgraph



Lee's algorithm flowgraph

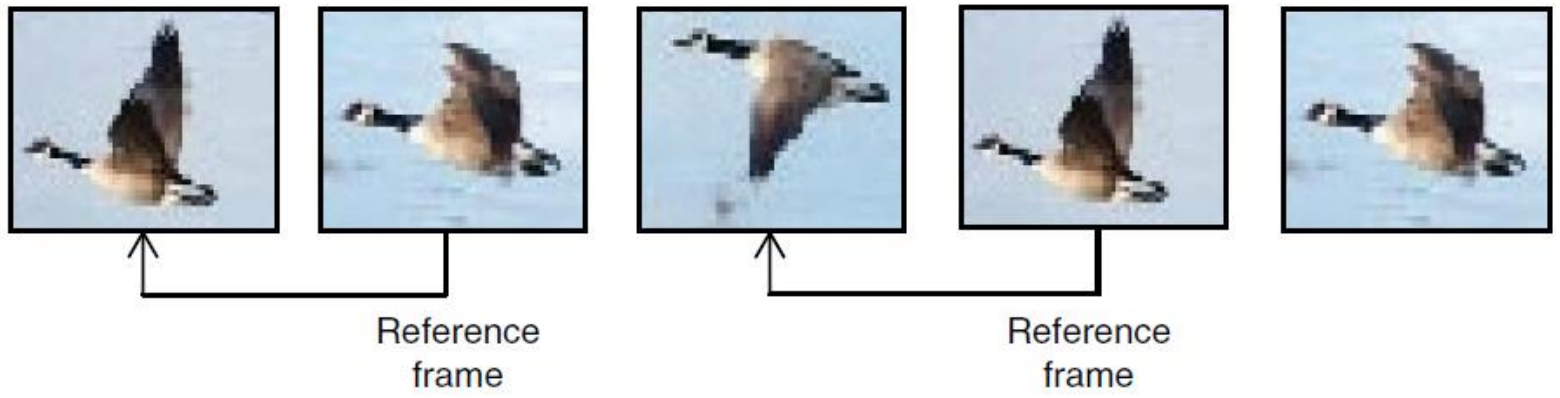


	X	+ / -
DCT-1D	64	56
DCT-2D	1024	896
Chen (1D)	16	26
Lees (1D)	13	29
Loeffler (1D)	11	29

Le nombre de multiplications exécutées peut être réduit de 64 à 32 grâce à la symétrie de la matrice de la DCT, mais le nombre d'additions et de soustractions augmente simultanément.

Compression vidéo

- Pour une **image**, la redondance est plutôt spatiale et est due à la corrélation entre les pixels voisins.
- Pour une **vidéo,** en plus de la redondance spatiale inhérente à chaque trame, la redondance temporelle, due à la corrélation entre trames voisines est également utilisée.



- Les différentes images d'une séquence vidéo sont codées en mode prédictif par rapport aux autres images. Les données à transmettre sont alors constituées des informations nécessaires pour réaliser la prédiction par compensation de mouvement.
- Pour pouvoir visualiser une image il est nécessaire de décoder toutes les images précédentes.

Image de référence F_n

Image de référence reconstruite F'_{n-1}

Image résiduelle F_n - F'_{n-1} (sans compensation de mouvement)

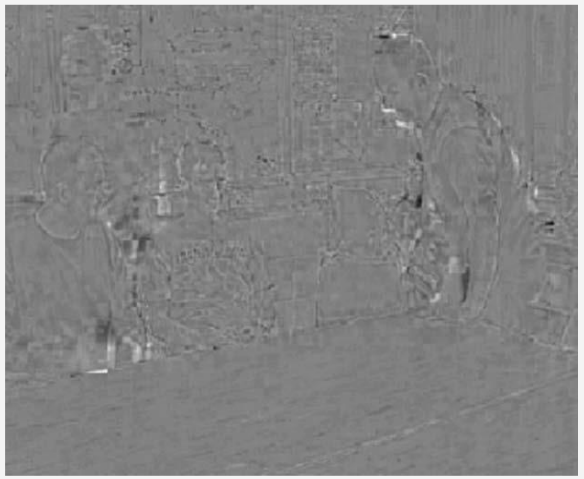


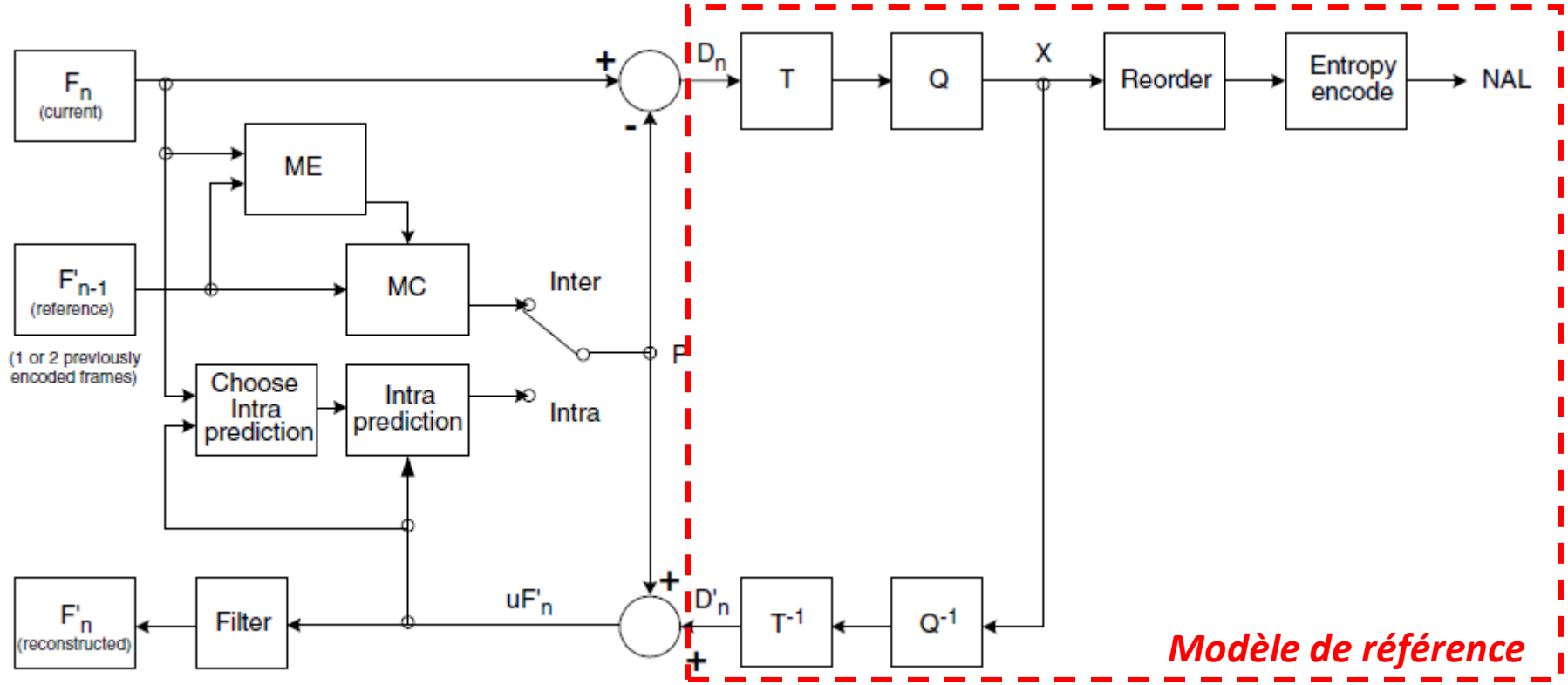
Image résiduelle F_n - F'_{n-1} (avec compensation de mouvement)

16*16 vecteurs de mouvement

Image F_n reconstruite

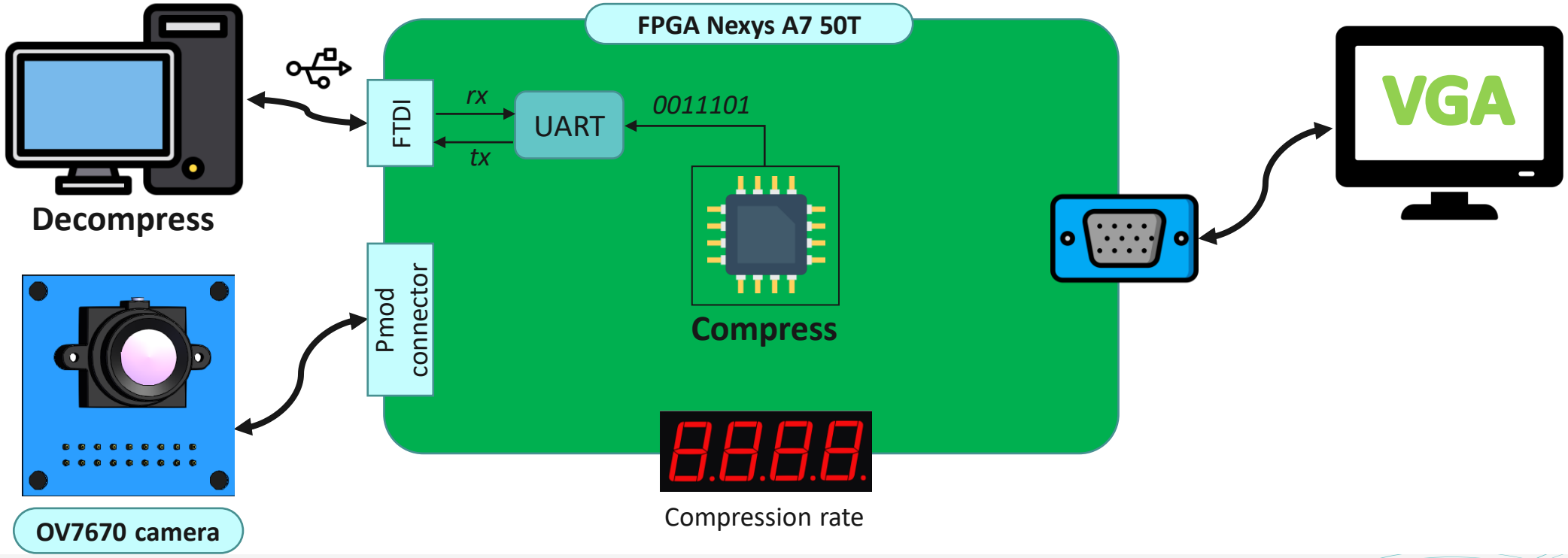
Compression vidéo : exemple d'un codeur H.264

- Les schémas de compression vidéo utilisent toujours plus ou moins la même architecture. Ils reposent sur des algorithmes hybrides de **prédiction-transformation**. Ces algorithmes associent une estimation de mouvement, une prédiction temporelle compensée en mouvement, un codage par transformation, une quantification et un codage entropique.



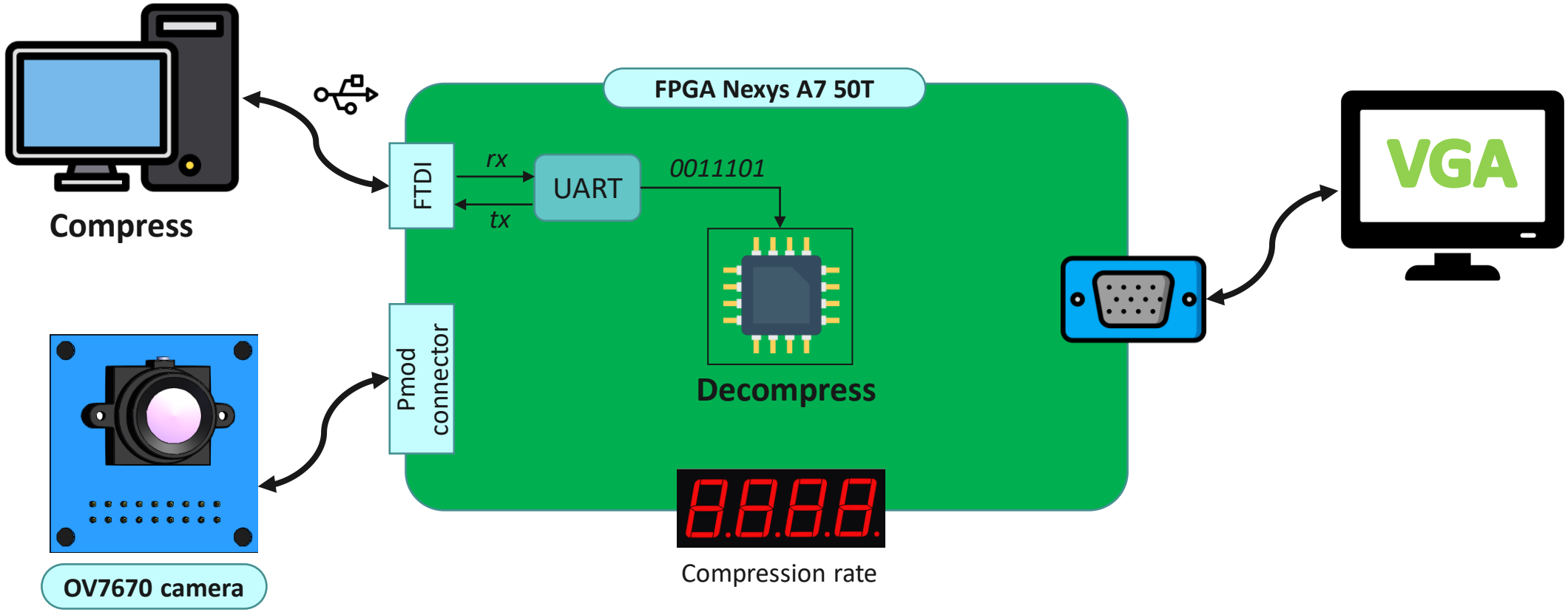
Implémentation FPGA

- L'idée est de se baser sur le modèle de référence pour élaborer via la HLS (*High Level Synthesis*) un compresseur/décompresseur vidéo.
- Proposition d'implémentation n°1 : Une carte FPGA (Nexys A7 50T) capte une vidéo avec la caméra (OV670), l'affiche sur un écran via la liaison VGA, compresse l'information et la renvoie par liaison UART au PC qui fera le décodage.



Implémentation FPGA

- Proposition d'implémentation n°2 : Un PC compresse une vidéo, envoie l'information encodée à une carte FPGA (via l'UART). Le FPGA décompresse les données et réaffiche la vidéo sur un écran PC via la liaison VGA.



Démonstration

```
python3 -B compression.py image.jpg -q 80
```

Compresseur

QLevel à définir

Image d'origine (RGB)

4 fichiers : données compressées (.bin)
+ 3 tables de Huffman (pour Y,Cr,Cb)

100010000

Métriques : Taux de compression / Entropie

```
python3 -B decompression.py
```

Décompresseur

?

Métriques : MSE, SNR, Temps d'éxecution

Perspectives, conclusion & questions

Bibliographie

/ **Pages Internet**

- *Jpeg : Colorspace Transform, Subsampling, DCT and Quantisation — Multimedia Codec Excercises 1.0 documentation*. https://www.hdm-stuttgart.de/%7Emaucher/Python/MMCodecs/html/jpegUpToQuant.html
- *YUV 420, YCbCr 422, RGB 444, c'est quoi le chroma subsampling ?* (2018, 24 mai). L'Atelier du câble. https://www.latelierducable.com/tv-televiseur/yuv-420-ycbcr-422-rgb-444-cest-quoi-le-chroma-subsampling/

/ **Livres**

- Bhaskaran, V., & Konstantinides, K. (2013). *Image and Video Compression Standards : Algorithms and Architectures* (The Springer International Series in Engineering and Computer Science, 334) (Softcover reprint of the original 1st ed. 1995). Springer.
- Richardson, I. E. (2003). *H.264 and MPEG-4 Video Compression : Video Coding for Next-generation Multimedia* (1re éd.). Wiley.
- Richardson, I. E. (2023). *Video Codec Design : Developing Image And Video Compression Systems* (1st éd.). WILEY INDIA.
- Waggoner, B. (2009). *Compression for Great Video and Audio, Second Edition : Master Tips and Common Sense (DV Expert)* (2e éd.). Focal Press.
- Yun Q. Shi, & Huifang Sun. (2017). *Image and Video Compression for Multimedia Engineering : Fundamentals, Algorithms, and Standards, Second Edition*. CRC Press.