Bruce GAMEIRO COSTA
M2 – Image

2021-2022

Représentation et Filtrage Numérique 1D/2D

- EBCOT et JPEG-2000 -

Github : https://github.com/branyoto-etud/filtrage

Table des matières

Abstract 3

Introduction 3

Contribution 3

Présentation 4

I/ JPEG-2000: l'algorithme 4

II/ EBCOT: l'algorithme 5

a) Présentation globale 5

b) Les plans de bits 5

c) Les primitives 6

d) Les passes 7

III/ Exemple 8

a/ Premier plan : n = 5 9

b/ Deuxième plan : n = 4 11

1) Première passe : propagation 11

2) Deuxième passe : affinage 12

4) Troisième passe : Nettoyage 12

IV/ Conclusion 13

a) Avantages et Inconvénients 13

b) Corrections 13

c) Améliorations 13

Abstract

Ce rapport a pour but d'étudier le problème de la compression d'image dans le domaine de l'informatique. Plus précisément, nous allons essayer de voir si l'algorithme JPEG-2000, combinant l'utilisant des ondelettes ainsi que la compression EBCOT, est une proposition envisageable sur le long terme.

Introduction

La compression est un thème très important en informatique, que ce soit pour des textes, des dossiers, ou des images cette problématique à été fortement explorée depuis la création des ordinateurs. En effet les supports de stockages sont limité en taille – bien que de nos jours le soucis est moins sensible due à l'explosion des tailles de disques dur – et l'on veut essayer d'exploiter l'espace le plus efficacement possible.

Dans le domaine de l'image, le soucis est encore plus présent. Maintenant que les smartphones peuvent prendre des photos ayant une résolution de 16M pixels, sans la compression, on ne pourrait stocker que quelques centaine de photos sur notre téléphone en même temps. Heureusement pour nous, des scientifiques se sont penché sur la question et ont proposé différentes solutions.

Le format PNG permet de compresser le photos sans perdre des données. Le format JPG (ou JPEG) permet une meilleure compression au détriment d'informations pouvant sembler moins importantes. Le GIF qui est une compression sans perte mais limitant le nombre de couleurs disponibles.

A ces format, s'ajoute le format que nous allons étudier : le JPEG-2000.

Contribution

La question logique à se poser est, pourquoi utiliser ce nouveau format alors qu'on en a déjà 3 qui fonctionnent très bien ?
Effectivement les 3 formats présenté sont très utilisés et plutôt performant, cependant l'intérêt du format JPEG-2000 est qu'il a un taux de compression bien supérieure au JPG et permet aussi un codage sans perte (comme le PNG) ainsi que plusieurs autre fonctionnalités que nous aborderons plus tard.

Cependant tout a un prix, le prix de cet algorithme est le temps de compression et de décompression qui est plus important que les autres méthodes.

Présentation

I/ JPEG-2000: l'algorithme

La compression au JPEG-2000 se fait en plusieurs étapes successives.

Pour commencer, chaque canal de couleurs doit être transformé en valeur dans l'intervalle [-128, 127].

Une opération non obligatoire possible est de changer la base colorimétrique au profit du modèle YUV qui a l'avantage de posséder des composantes moins liées entre elles.

Ensuite, il est possible de découper l'image en tuiles rectangulaires de même taille (à l'exception des bords qui peuvent avoir une taille différente) ou de garder l'image entière.

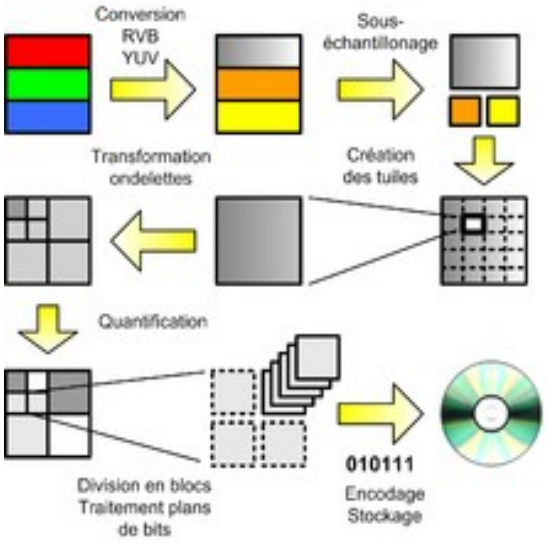


Figure 1: Schéma de l'algorithme JPEG-2000

L'avantage du découpage est que l'algorithme est moins gourmand en mémoire sur les grandes images, mais pour une compression avec perte il est possible de voir un effet de tilling (voir Figure 2).

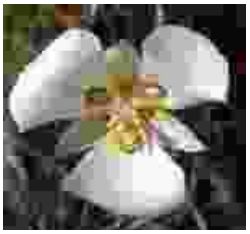


Figure 2: Effet de tiling

Par la suite on applique la transformée en ondelette, les deux types d'ondelettes utilisées sont :

- L'ondelette de Le Gall pour avoir une compression sans perte. Les résultats sont à valeur entière et ne nécessites donc pas de quantification.
- L'ondelette de Daubechies qui produit un fichier plus petit mais qui est plus complexe à implémenter.

Puis on quantifie le résultat tout en éliminant les coefficients proche de zéro qui sont considérés comme peu significatifs et donc pouvant être ignorés.

Et enfin on code le résultat grâce à l'algorithme EBCOT qui sera détaillé dans le prochain point.

II/ EBCOT: l'algorithme

a) Présentation globale

L'algorithme EBCOT, pour Embedded Block Coding with Optimal Truncation, est un codeur entropique par plan de bits. L'image est premièrement découpée en bloc de hauteur 4. Pour chaque plan, le parcours se fait bloc par bloc en appliquant le traitement colonne par colonne (voir Figure 3). Les blocs sont traités de la même façon qu'il soient LL, LH, HL ou HH à l'exception de certains cas qui seront explicités au moment voulut.

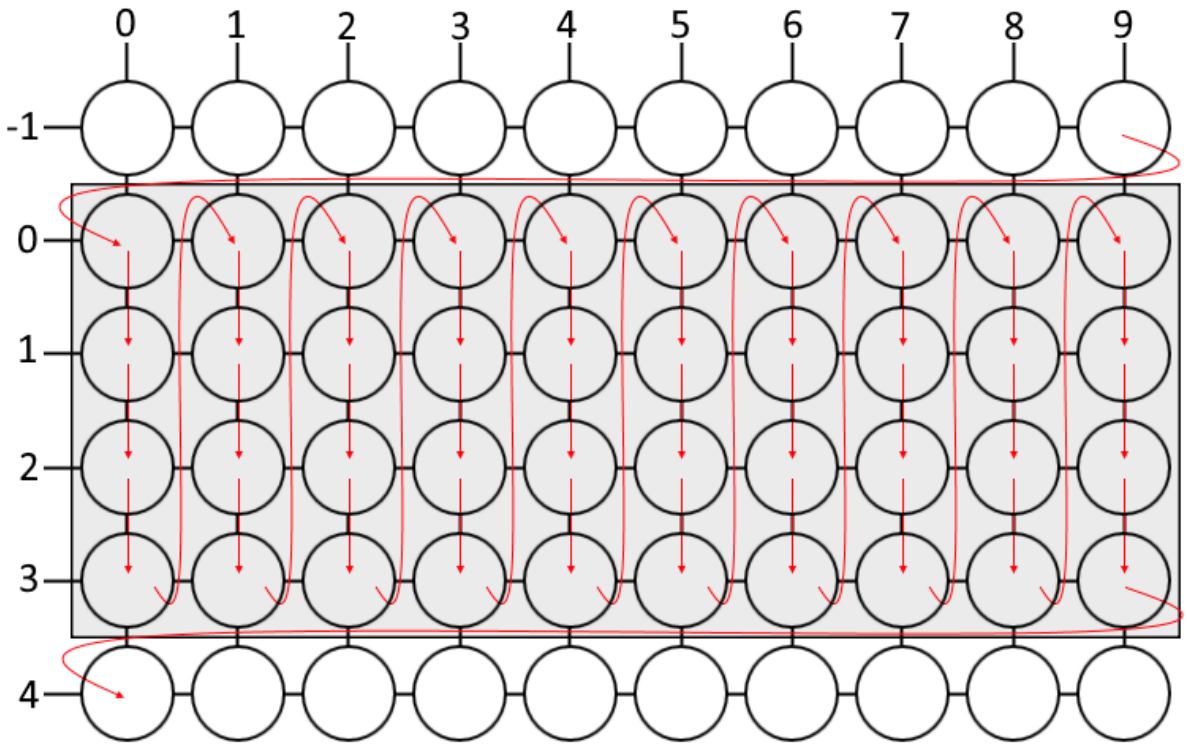


Figure 3: Exemple de parcours d'un bloc

b) Les plans de bits

L'algorithme se base donc sur un système de plans. Chaque plans ne considère que des coefficients dit ''significatifs''. Un coefficient est dit significatif au plan n, si le coefficient en valeur absolue est supérieur à 2^n. Une image possède donc un nombre limité de plans N qui est définit par $N = 1 + \lfloor \log_2(|\text{maximum}|) \rfloor$. Il y a donc N plans allant de 0 à N-1.

Le parcours des plans se fait de celui possédant le moins de coefficient significatifs à celui en ayant le plus. L'intérêt de cet algorithme est que plus on considère de plans de bits, plus le résultat sera proche de l'image réelle. Particulièrement, si tous les plans sont traités on aura une compression sans pertes.

c) Les primitives

Il y a 4 primitives utilisées :

- La primitive *Run-Length* (RL) est très simple, si aucun coefficient n'est significatif, alors on code un 0 sinon on code un 1 suivit de son offset (sur 2 bits) puis la primitive SC. Cette primitive n'est utilisé que si la colonne précédente ne contient pas de coefficients significatifs sur ce plan et que la colonne courante jusqu'à ce qu'un coefficient soit rencontré sur la colonne courante. Dans le cas contraire il faut utiliser la primitive ZC.

- La primitive *Sign Coding* (SC) permet de prédire le signe du coefficient significatif trouvé en fonction de ses **coefficients significatifs voisins** déjà traité (voir Table 1).
  Δ_H (resp. Δ_V) vaut 0 si les voisins horizontaux (resp. verticaux) sont de signe opposé **ou** que le coefficients n'a pas de voisins horizontaux. Sinon Δ_H vaut -1 si au moins un de ses voisins horizontaux est négatifs et 1 si au moins un de ses voisins horizontaux est positif.
  Si le signe prédit est correct, on code 0 suivit du contexte SC (sur 3 bits), sinon on code 1 suivit du contexte SC.

Δ_H	Δ_V	SC*x*	pred.
1	1	SC4	1
1	0	SC3	1
1	-1	SC2	1
0	1	SC1	1
0	0	SC0	1
0	-1	SC1	-1
-1	1	SC2	-1
-1	0	SC3	-1
-1	-1	SC4	-1

Table 1: Sign Coding

- La primitive Zero Coding (ZC) est utilisé à plusieurs moments afin de coder généralement les coefficients non significatifs qui suivent un coefficient significatif.
  Pour cela on code 0 si le coefficient n'est pas significatif et 1 si il l'est suivit du contexte ZC (5 bits).
  Dans le tableau ci-contre, les coefficients K_H, K_V et K_D sont respectivement pour le nombre de coefficients significatifs déjà traités horizontaux, verticaux et diagonaux. Un 'x' représente n'importe quelle valeur de coefficient.

Bandes LL & LH			Bandes HL			Bandes HH		ZC*x*
K_H	K_V	K_D	K_H	K_V	K_D	$K_H + K_V$	K_D	
0	0	0	0	0	0	0	0	ZC0
0	0	1	0	0	1	0	1	ZC1
0	0	≥ 2	0	0	≥ 2	0	≥ 2	ZC2
0	1	x	1	0	x	1	0	ZC3
0	2	x	2	0	x	1	1	ZC4
1	0	0	0	1	0	1	≥ 2	ZC5
1	0	≥ 1	0	1	≥ 1	2	0	ZC6
1	≥ 1	x	≥ 1	1	x	2	≥ 1	ZC7
2	x	x	x	2	x	≥3	x	ZC8

Table 2: Zero Coding

- La primitive Magnitude Refinement (MR) est utilisé pour permettre de connaître la vraie valeur des coefficients. Pour chaque coefficient significatif des plans précédent, on code la valeur du bit à la position n de la représentation binaire du coefficient. Ensuite, on ajoute la valeur du contexte MR (2 bits).
  σ vaut 0 la première fois que le coefficient passe dans la phase d'affinage.

σ	$K_H + K_V$	MR*x*
0	0	MR0
0	≠ 0	MR1
1	x	MR2

Table 3: Magnitude Refinement

d) Les passes

A chaque plans de bits on applique successivement 3 passes[1] sur un bloc avant de passer au suivant:

- La passe de propagation : où l'on va chercher autours des coefficients significatifs du plan précédent et ceux du bloc précédent pour trouver des coefficients significatifs, on va appeler ces coefficients les ‘‘presque significatifs’’. Les coefficients ‘‘presque significatifs’’ ne contiennent pas les coefficients significatifs du plan précédent.
  Ensuite on va utiliser la primitive ZC pour coder chaque coefficients suivit de la primitive SC si le coefficient était significatif.

- La passe d'affinage : où l'on va affiner les coefficients significatifs du plan précédent. Le but de cette étape est que si l'on combine les plans tel que :

  $$2^a + \sum_{i=0}^{N} 2^i . affinage_i$$

  affinage$_i$ est la valeur de l'affinage trouvé au plan i (ou 0 si le coefficient n'était pas significatif au plan i) et a le numéro du plan à partir duquel ces coefficient est significatif.

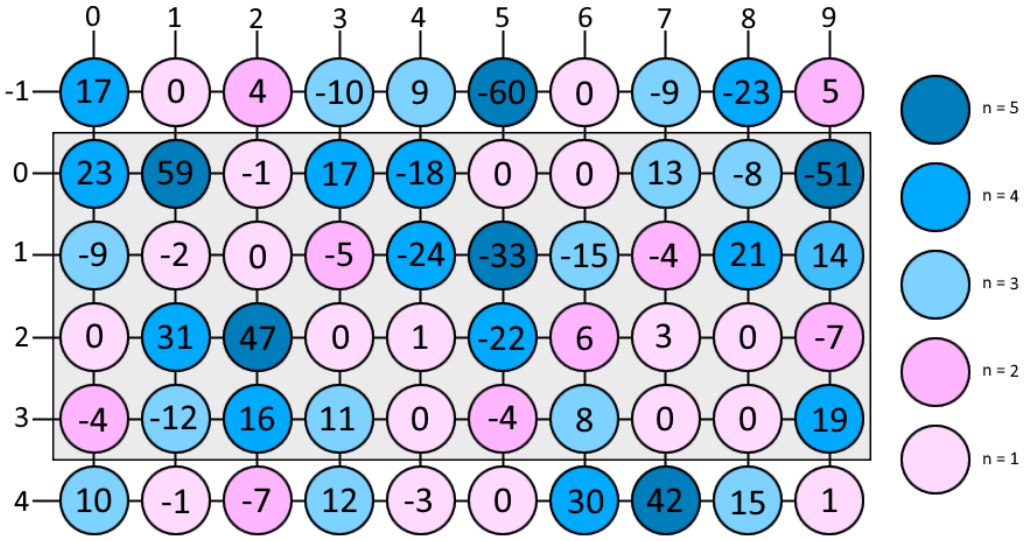
  le résultat obtenu doit être la vraie valeur du coefficient.
  Pour cela on utilise la primitive MR.

- La passe de nettoyage : où l'on va coder les coefficients n'ayant pas été utilisés lors des étapes précédentes. Pour cette étape, on utilise la primitive RL si la colonne précédente **ne contient pas** de coefficient significatif et tant que la colonne actuelle **ne contient pas** de coefficient significatif.
  Si on utilise la primitive RL et qu'aucun coefficient significatif n'est trouvé, alors on code 0, sinon on code 1 suivit de l'offset du coefficient (sur 2 bits) puis on utilise la primitive SC. Le reste de la colonne sera codé avec la primitive ZC.
  Note : Cela implique que la colonne suivante sera aussi codé en utilisant la primitive ZC. Si aucun coefficient significatif n'est trouvé, alors on repassera sur l'utilisation de la primitive RL. Inversement si un coefficient significatif est trouvé, alors la colonne encore après sera toujours en utilisant la primitive ZC.

1 Hormis pour le premier plan de bit traité où seule la dernière peut être appliquée (vu qu'il n'y a pas encore de plan de bit précédent).

III/ Exemple

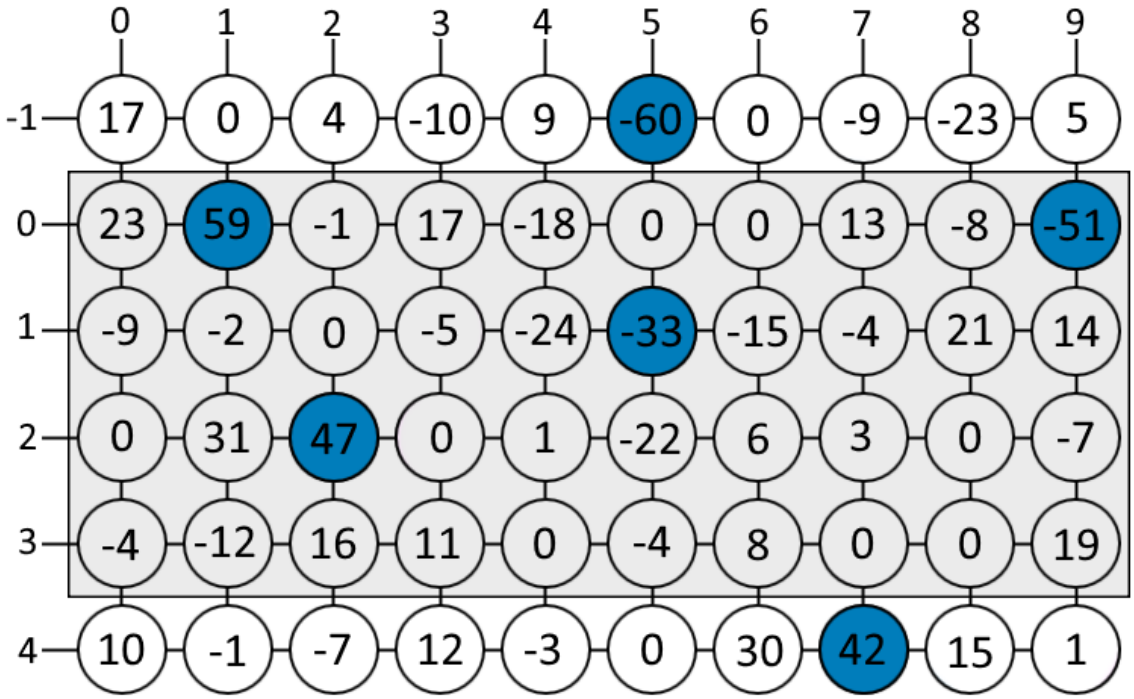
Nous allons maintenant présenter un exemple de l'algorithme de compression. Cet exemple a pour but de visualiser l'algorithme de compression sur des données, cependant les coefficients ne sont pas ceux obtenues par la transformées en ondelettes d'une images, ils sont purement inventé et prévu pour avoir des cas limites afin de pouvoir visualiser au mieux comment fonctionne l'algorithme. Pour cela nous allons nous focaliser sur un fragment d'une image que l'on suppose quantifié et appartenant à la sous-bande LL.



Sur l'image sont visible les différents coefficients significatifs en fonction du plan de bit.

On suppose que le coefficient maximum de l'image est -60. Donc il y aura :
$N = 1 + \lfloor \log_2(|-60|) \rfloor = 6$ plans de bits.

a/ Premier plan : n = 5



Pour le premier plan traité, seule la troisième passe peut être effectuée.

On comment avec la primitive RL. La première colonne ne possède pas de coefficient significatif pour ce plan, donc on va coder **0 RL**.

Dans la seconde colonne, le coefficient (0, 1) est significatif, donc on va coder **1 RL** puis l'offset sur 2 bits **00**. Ensuite on indique le signe en utilisant la primitive SC, le coefficient ne possède pas de voisins significatifs donc le contexte SC0 est utilisé. Le signe prédit est positif, ce qui est vrai donc on code **0 SC0**.
Comme un coefficient significatif a été trouvé, le reste de cette colonne et la colonne suivante seront codé avec la primitive ZC.
Le coefficient (1, 1) n'est pas significatif et il possède uniquement un voisin significatif en vertical (le coefficient 47 n'a pas encore été traité) donc le on va avoir le contexte ZC3. On code donc **0 ZC3**.
Les deux coefficients suivant ne sont pas significatifs et n'ont pas de voisins significatifs, donc on code les deux par **0 ZC0**.

La troisième colonne est codé en utilisant la primitive ZC puisque la colonne précédente possédait un coefficient significatif.
Le coefficient (0, 2) et (1, 2) ne sont pas significatifs et ils possèdent respectivement un voisin significatif horizontal et diagonal donc ils seront codé par **0 ZC5 0 ZC1**.
Le coefficient (2, 2) est significatif et il ne possède aucun voisin significatif donc il est codé par **1 ZC0** puis son signe est codé par **0 SC0** puisqu'il est positif et qu'il ne possède pas de voisins significatif.
Le coefficient (3, 2) n'est pas significatif et il possède un voisin significatif vertical donc on code **0 ZC3**.

La colonne suivante doit encore utilisé la primitive ZC. On obtient le code suivant **0 ZC0 0 ZC1 0 ZC5 0 ZC1**.

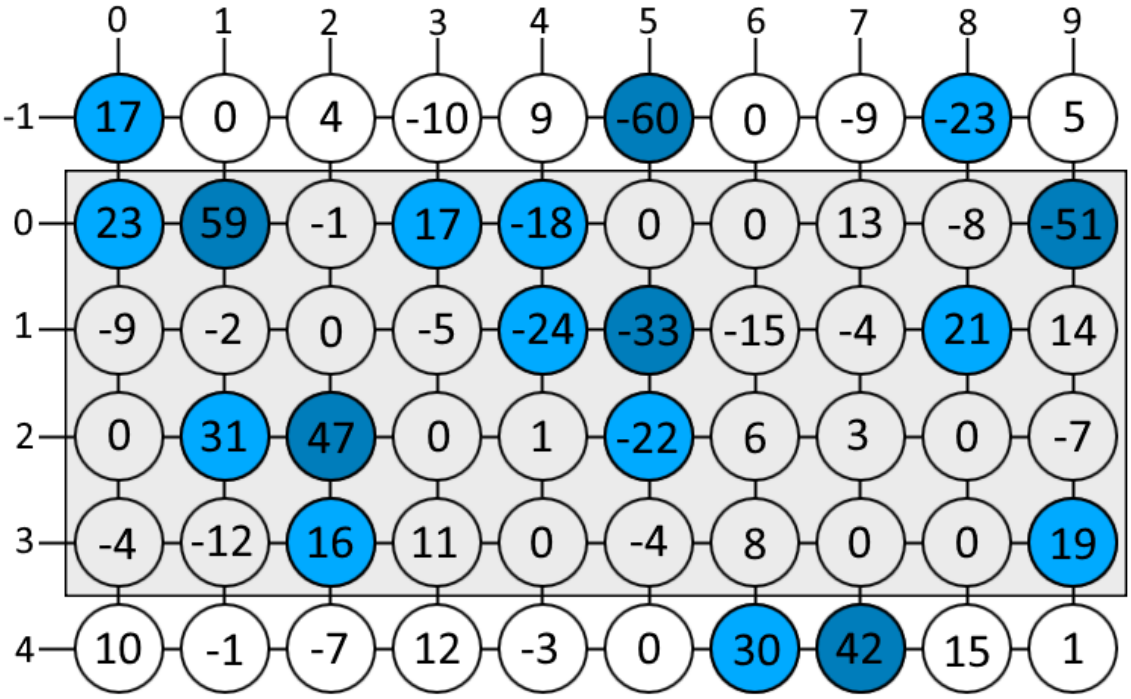
La colonne précédente ne contenait pas de coefficient significatif, donc on peut réutiliser la primitive RL. Cette colonne ne contient pas de coefficient significatif donc on code **0 RL**.

Dans le tableau ci-dessous le résultat de la passe sur toutes les colonnes.

	bits	détails
0	0 RL	Aucun significatif
1	1 RL 00	1 significatif 1ere ligne
	0 SC0	Le signe est correct
	0 ZC3	Le reste de la ligne est encodé avec la primitive ZC
	0 ZC5	
	0 ZC1	
2	0 ZC5	Primitive ZC car la colonne d'avant a un significatif
	0 ZC3	
	1 ZC0	Coefficient significatif
	0 SC4	Le signe est correct
	0 ZC3	
3	0 ZC0	Primitive ZC car la colonne d'avant a un significatif
	0 ZC1	
	0 ZC5	
	0 ZC1	
4	0 RL	Pas de significatif avant

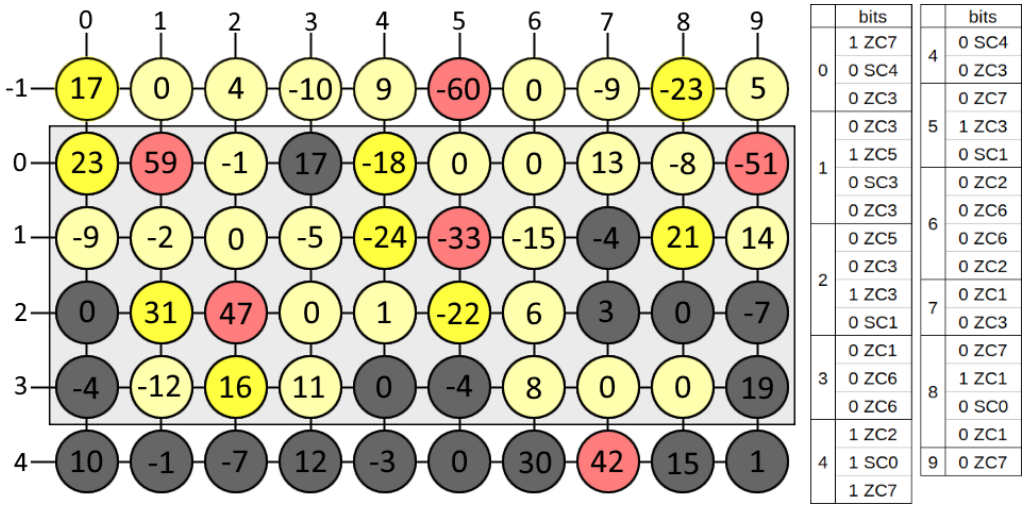
	bits	détails
5	1 RL 01	1 significatif 2eme ligne
	0 SC3	Le signe est correct
	0 ZC3	Le reste de la ligne est encodé avec la primitive ZC
	0 ZC0	
6	0 ZC2	Primitive ZC car la colonne d'avant a un significatif
	0 ZC5	
	0 ZC1	
	0 ZC1	
7	0 RL	Pas de significatif avant
8	0 RL	Pas de significatif avant
9	1 RL 00	1 significatif 1ere ligne
	0 SC2	Le signe est correct
	0 ZC3	Le reste de la ligne est encodé avec la primitive ZC
	0 ZC0	
	0 ZC0	

b/ Deuxième plan : n = 4



Sur ce plan, ainsi que les prochains, il est possible de faire les 3 passes.

1) Première passe : propagation



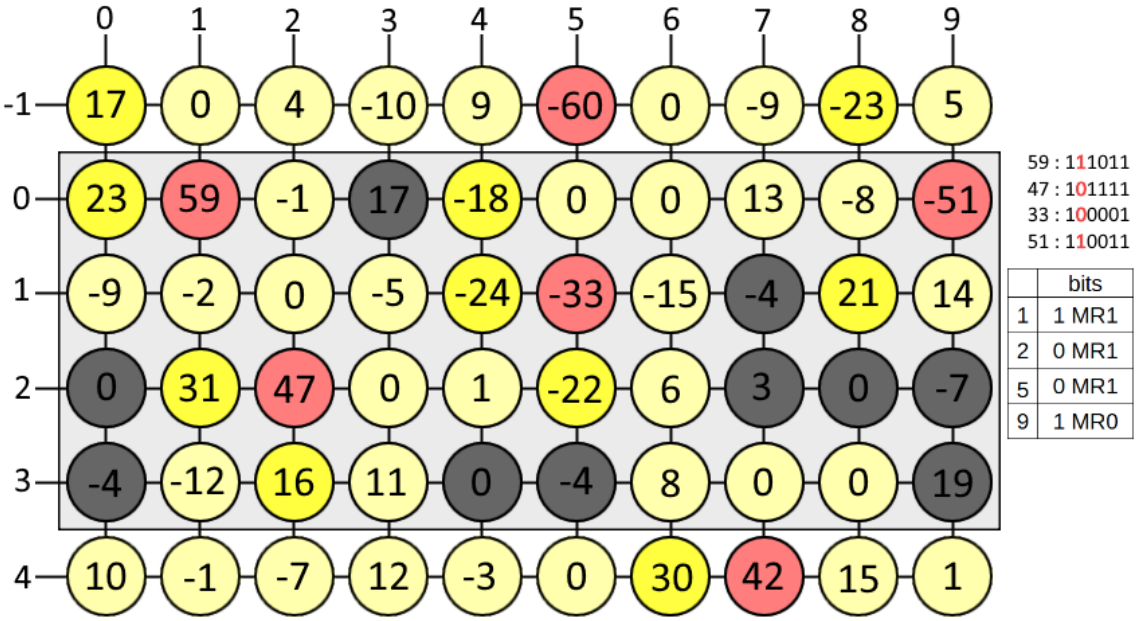
	bits
0	1 ZC7
	0 SC4
	0 ZC3
1	0 ZC3
	1 ZC5
	0 SC3
	0 ZC3
2	0 ZC5
	0 ZC3
	1 ZC3
	0 SC1
3	0 ZC1
	0 ZC6
	0 ZC6
4	1 ZC2
	1 SC0
	1 ZC7

	bits
4	0 SC4
	0 ZC3
5	0 ZC7
	1 ZC3
	0 SC1
6	0 ZC2
	0 ZC6
	0 ZC6
	0 ZC2
7	0 ZC1
	0 ZC3
8	0 ZC7
	1 ZC1
	0 SC0
	0 ZC1
9	0 ZC7

En gris sont indiqué les coefficients non vu par cette passe.
En rouge les cocefficient significatifs des plans précédents.
En jaune clair les coefficients non significatifs trouvé par cette passe et en jaune foncé les coefficients significatifs.

On regarde aussi les coefficients significatifs autour de 42 car bien qu'il ne soit pas dans ce bloc, il est significatif au plan précédent. Demême pour le coefficient -23 car il est significatif au bloc précédent.

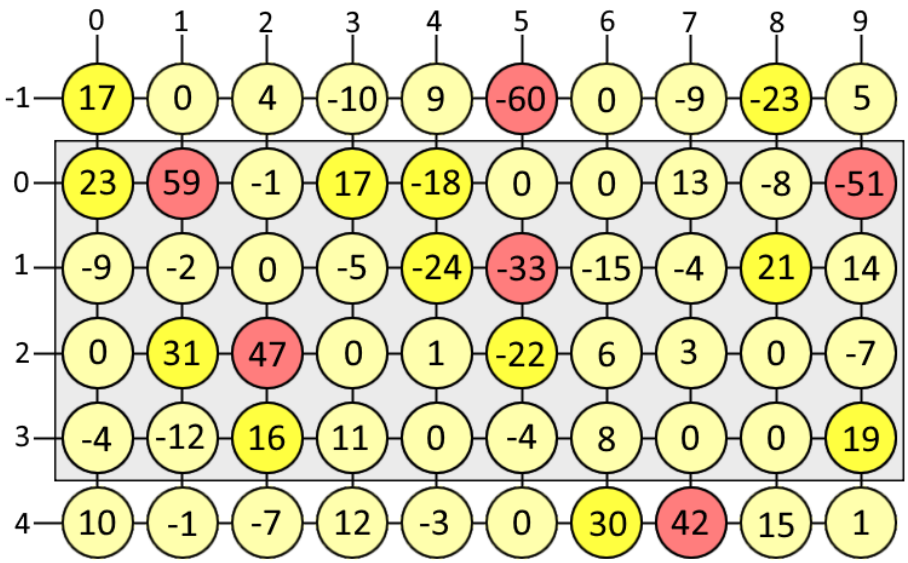
2) Deuxième passe : affinage



	bits
1	1 MR1
2	0 MR1
5	0 MR1
9	1 MR0

Ici on ne va regarder que les coefficients significatifs des plans précédents (rouge) de ce bloc.

4) Troisième passe : Nettoyage



	bits
0	0 RL
3	1 RL 00
	1 SC3
4	0 ZC1
5	0 RL
7	0 RL
8	0 RL
9	1 RL 11
	0 SC0

On voit les coefficient n'ayant pas été vu précédemment (les coefficients gris des images précédentes). On utilise RL pour la colonne 0, 5, 7, 8 et 9 car les colonnes précédentes n'ont pas de coefficient significatifs dans cette passe.

IV/ Conclusion

a) Avantages et Inconvénients

Le format JPEG-2000 propose de nombreux avantages :

- Il a un meilleur taux de compression que le JPEG
- On peut faire de la compression sans pertes si tous les plans de bits sont traités
- On peut choisir son niveau de résolution à partir des plans de bits
- Il supporte nativement les ‘‘Region of Interest’’, qui sont des zones plus détaillés dans l’image (les visages par exemple)
- On peut traiter des grosses images sans soucis en les fragmentant en plus petites images
- En un traitement on peut générer plusieurs résolutions de l’image en sauvegardant l’image après chaque plan de bits
- La résistance aux erreurs est très grande car beaucoup d’indicateurs de segments sont envoyés (codes de contexte, fin de passe, fin de plan de bits, ...)

Cependant le principal défaut est le temps de calcul nécessaire à la compression qui peut exploser lors du traitement de très grosses images. Ce qui a comme effet de bord de surcharger les serveurs et provoque des ralentissements.

Le JPEG-2000 n’est pas supporté par les navigateurs tel que Chrome, Firefox, etc. Et bien que les noms se ressemblent, le JPEG et le JPEG-2000 ne sont pas compatible et ne sont pas facilement convertible.

b) Corrections

Dans le papier : http://d.xav.free.fr/ebcot/#x1-80004.2, lors de la passe de propagation (4.1) le coefficient (0, 6) est pris en compte ce qui est correct. Cependant, lors de la passe de nettoyage (4.3) il est aussi pris en compte alors qu’il ne devrait pas puisqu’il a déjà été vu précédemment.

c) Améliorations

L’algorithme JPEG-2000 est en perpétuelle évolution, depuis sa sortie en 2000 il y a eu 16 extensions. Certaines permettent de faciliter les transferts sur le réseau, d’autres améliorent la sécurité et il y en a qui permettent de compresser des images 3D (utile pour l’imagerie médicale).

Les évolutions du JPEG-2000 sont très conditionnées par le contexte actuel principal de son utilisation : l’imagerie médicale et les vidéos (surveillance, satellites, …).

Mais si l'on reste sur l'algorithme basique présenté ici, une amélioration serait de ne pas ajouter le contexte après chaque bit, l'avantage de cette suppression est que la taille du fichier est divisée par 5 en moyenne. L'inconvénient que cela peut apporter est que la résistance aux erreurs s'en trouve amoindrie.

Une seconde amélioration qui peut être combinée à la première est de remplacer la prédication de signe, donc la primitive SC, par le signe (1 pour positif et -1 pour négatif). Encore une fois la taille du fichier est réduite de 4*N bits (avec N le nombre de coefficients) mais c'est le temps de calcul qui est le plus impacté car on n'a plus besoin de chercher autour des coefficients pour essayer de déterminer leur signe pour ensuite le corriger, on l'indique directement quand on en a besoin.