Signal, image et multimedia

Nouvelle approche d'accélération du codage fractal d'images

Sofia Douda*, Amer Abdelhakim El Imrani**, Mohammed Limouri**

* Département de Mathématiques et Informatique
Faculté des Sciences et Techniques, Settat, Maroc
** Laboratoire de Conception et Systèmes
Faculté des Sciences, Rabat, Maroc
sofia_douda@yahoo.fr

RÉSUMÉ. La compression fractale d'images permet un décodage rapide et une indépendance de la résolution mais souffre d'une lenteur dans le codage. Le présent travail présente une approche visant à réduire le temps de calcul en utilisant deux dictionnaires et une approximation de l'image en deux étapes (AP2D). L'approche AP2D peut être appliquée aux méthodes de classification ou aux méthodes de réduction du cardinal du dictionnaire et ainsi réduire davantage le temps de codage. Les résultats expérimentaux ont montré que AP2D appliquée à une recherche exhaustive a atteint un gain de temps de plus de 72%. De même AP2D appliquée à la classification de Fisher a permis une réduction de temps de codage de plus de 65%. La qualité de l'image n'a pas été altérée par cette approche et le taux de compression a légèrement augmenté.

ABSTRACT. The Fractal image compression has the advantage of presenting fast decoding and independent resolution but it suffers of slow encoding phase. In the present study, we propose to reduce the computational complexity by using two domain pools instead of one domain pool and encoding an image in two steps (AP2D approach). AP2D could be applied to classification methods or domain pool reduction methods leading to more reduction in encoding phase. Indeed, experimental results showed that AP2D speed up the encoding time. The time reduction obtained reached a percentage of more than 65% when AP2D was applied to Fisher classification and more than 72% when AP2D was applied to exhaustive search. The image quality was not altered by this approach while the compression ratio was slightly enhanced.

MOTS-CLÉS : Compression fractale d'images, IFS, Codage/décodage, temps de calcul.

KEYWORDS: Fractal image compression, IFS, Coding/decoding, time computation.

1. Introduction

La compression fractale d'images a été proposée pour la première fois par Barnsley [1]. Cette méthode, basée sur le théorème du collage [2], montre qu'il est possible de coder des images fractales à l'aide de quelques transformations contractantes définissant un système de fonctions itérées (SFI). Barnsley proposa un algorithme pour construire, à partir d'une image donnée, un ensemble de transformations contractantes la représentant. Les signaux naturels ne possédant pas forcément la propriété d'auto-transformabilité globale, Jacquin [9] proposa de rechercher des auto-transformabilités locales ou partielles ce qui a conduit au premier algorithme de compression par SFI et à la notion de système de fonctions itérées locales (SFIL).

Pour coder une image réelle par fractales, on effectue un partitionnement adapté de l'image où chaque partie élémentaire (bloc cible) est mise en correspondance avec une autre partie d'échelle différente (bloc source) recherchée dans toute l'image [9]. Les blocs peuvent être de taille variable, de forme carrée (partition en quadtree [5]) ou de forme rectangle (partition H-V [6]) ou triangulaire [3]. Chaque correspondance est décrite par une transformation affine locale et l'union de ces transformations (SFIL) forme le code de l'image. Le calcul du SFIL pendant le processus de codage est très long car pour chaque bloc cible, le bloc correspondant est recherché parmi tous les blocs sources. L'ensemble de ces blocs source est appelé dictionnaire. Les principales approches de réduction du temps de calcul des SFIL se basent sur la classification des blocs. Dans ce schéma de classification, les blocs cible et source sont groupés en un nombre fini de classes selon leurs caractéristiques communes. Durant la phase du codage, seul les blocs source ayant la même classe que le bloc cible, seront comparés à ce dernier. Jacquin [10] a proposé un schéma de classification basé sur les caractéristiques discrètes des blocs. Seul trois classes de blocs sont distinguées : blocs homogènes, blocs avec contour et blocs texturés. Dans la méthode de classification de Fisher (CF) [7], un bloc (cible ou source) est divisé en quatre quadrants. Pour chaque quadrant, la moyenne et la variance sont calculées. Selon une certaine combinaison de ces valeurs, 72 classes sont construites. Une autre méthode de classification des blocs, basée sur la transformée en cosinus discrète (TCD), consiste à classer les blocs en trois classes : bloc homogène, bloc avec contour diagonal/sous-diagonal et bloc avec contour horizontal/vertical [4]. Elle est basée seulement sur le calcul de deux coefficients de la TCD à savoir les coefficients horizontaux et verticaux de plus basses fréquences. D'autres approches de réduction du temps de calcul se basent sur la diminution du cardinal du dictionnaire par l'élimination d'un certain nombre de blocs source du dictionnaire par exemple ceux à faible variance [12] ou ceux à entropie élevée [8].

Dans ces approches de réduction de temps de codage citées, un seul dictionnaire est utilisé pour chercher les similarités entre blocs cible et source. Ce dictionnaire est

construit avec un pas de chevauchement de blocs source fixé au départ à P. Dans le but d'accélérer davantage le codage fractal d'images, nous avons conçu dans notre travail, une approche qui utilise deux dictionnaires et une approximation de l'image en deux étapes.

2. Principe de base de la compression fractale

La plupart des algorithmes de compression fractale sont des variantes de l'algorithme de Jacquin [10], [11]. Ils construisent, pour une image donnée I, un ensemble de transformations (simples) localement contractantes (SFIL) W_i qui, itéré sur une image de départ quelconque, donne une approximation de cette image (attracteur du SFIL : $A = W_i (A)$). La construction des transformations s'inspire du théorème du collage. En effet, elle minimise la distance entre l'image originale I et sa transformée par le SFIL $W_i(I)$ au lieu de celle entre I et l'attracteur du SFIL A.

Pour définir les transformations formant W_i, il faut préciser leurs domaines de départ et d'arrivée. Les domaines d'arrivées forment une partition $\{R_1, R_2, \ldots, R_N\}$ de l'image, ils sont appelés blocs cible et sont de tailles B x B. Les domaines de départ sont également des blocs de l'image de taille 2B x 2B, $\{D_1, D_2, \ldots, D_M\}$ qui peuvent se chevaucher et sont appelés blocs source et l'ensemble des blocs source est appelé dictionnaire. Chaque bloc cible est ensuite mis en correspondance avec un autre bloc transformé $W_i(D_j)$ lui ressemblant, au sens d'une mesure d'erreur sur les niveaux de gris. La complexité de la recherche du meilleur bloc source est linéaire en M, le nombre d'éléments du dictionnaire. Le code fractal est formé des paramètres des transformations (position du bloc source et coefficients de la fonction affine agissant sur les niveaux de gris). La distance utilisée dans le calcul des erreurs de collage est la distance euclidienne.

La transformation w_i des blocs source comprend une transformation géométrique qui déforme le support des blocs D_j et une transformation massique qui agit sur la luminance des pixels des blocs D_j. Chaque w_i exprime (code) la manière de passer d'un bloc D_j de l'image à un autre bloc R_i de taille plus petite lui ressemblant. La transformation affine d'un niveau de gris z=f(x,y) d'un pixel de position (x,y) de l'image A est du type de l'équation (1).

$$w_i \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_i & b_i & 0 \\ c_i & d_i & 0 \\ 0 & 0 & s_i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \\ o_i \end{bmatrix} \quad (1)$$

Pour la compression, le nombre de transformations géométriques possibles est limité à 8 isométries applicables aux blocs carrés (identité, rotation de $\pi/2$, π, $3\pi/2$, symétrie horizontale, verticale et les 2 symétries diagonales). Ainsi, les coefficients a_i, b_i, c_i, d_i,

qui décrivent la transformation géométrique, ne peuvent prendre qu'un nombre fini de valeurs. Le vecteur (e_i, f_i) permet de translater le support du bloc source pour le faire coïncider avec celui du bloc cible. Ces deux transformations constituent la transformation spatiale définie par Jacquin. Les coefficients o_i et s_i concernent respectivement une différence de moyenne des niveaux de gris et une variation de contraste entre blocs cible et blocs source.

Le calcul d'une transformation w_i, constituant le code SFIL d'une image A, se fait par minimisation de l'erreur quadratique entre R_i et $w_i(D_j)$. Cette erreur s'écrit :

$$R = \frac{1}{n}\sum_{i=1}^{n}(sD_j(k) + o - R_i(k))^2 \tag{2}$$

n étant le nombre total de pixels dans le bloc cible. La minimisation de cette erreur, en fonction du facteur d'échelle s (contraste) et du rehaussement de niveau de gris o (offset), est assez immédiate puisqu'il suffit d'annuler les dérivées partielles en s et o. Ainsi, les valeurs optimales de s et o sont données par les formules suivantes :

$$s = \frac{n\sum_{i=1}^{n} d_i r_i - \sum_{i=1}^{n} d_i \sum_{i=1}^{n} r_i}{n\sum_{i=1}^{n} d_i^2 - (\sum_{i=1}^{n} d_i)^2} \tag{3}$$

$$o = \frac{1}{n}(\sum_{i=1}^{n} r_i - s\sum_{i=1}^{n} d_i) \tag{4}$$

Où les r_1, r_2, …, r_n sont les valeurs des pixels de R_i et d_1, d_2, …, d_n celles des pixels de Dj.

L'erreur R se calcule ainsi par :

$$R = \frac{1}{n}\left[\sum_{i=1}^{n} b_i^2 + s(s\sum_{i=1}^{n} a_i^2 - 2\sum_{i=1}^{n} a_i b_i + 2o\sum_{i=1}^{n} a_i) + o(no - 2\sum_{i=1}^{n} b_i)\right] \tag{5}$$

Pour le décodage, il suffit alors de commencer par n'importe quelle image initiale A_0 et d'appliquer les transformations w_i plusieurs fois (6 à 10 le plus souvent). L'algorithme tend alors vers le point fixe Ã qui constitue l'approximation fractale de l'image initiale A.

3. Algorithme de base de codage fractal d'images

Dans le partitionnement quadtree utilisé, les blocs d'image sont carrés (de préférence de côté une puissance de deux). Lorsqu'une recherche est effectuée pour un bloc cible donné, on teste l'erreur minimale trouvée. Si cette erreur est supérieure à un certain seuil fixé, le bloc est divisé en quatre carrés de côtés égaux, et une recherche est effectuée pour chacun des sous-blocs obtenus, jusqu'à ce que l'erreur soit inférieure au seuil, ou que les blocs soient devenus trop petits pour être divisés à nouveau. Ce type partitionnement est simple à utiliser et fournit des résultats corrects par rapport à un partitionnement carré où tous les blocs sont de taille carrée fixée. L'algorithme principal de codage de compression fractale d'images basée sur le partitionnement quadtree [5] peut être résumé comme suit :

- Construire le dictionnaire D(P) de pas P.
- Fixer un seuil d'erreur T.
- Partitionner l'image en blocs cible R_i jusqu'à une profondeur maximale.
- Pour chaque block R_i non traité faire:
    - Pour chaque bloc D_j du dictionnaire D:
    - Calculer les coefficients s et o de la transformation w_i tel que la distance $r_{ij}=d_{L2}(w_i(D_j), R_i)$ soit minimale.
    - Retenir le bloc Dj et les coefficients s et o tel que la distance r_{ij} correspondante est minimale.
    - Si cette distance est inférieure au seuil T:
        - Encoder la référence du bloc source et les coefficients de la transformation.
    - Sinon:
        - Si la taille du bloc est inférieure à une profondeur minimale
            -Subdiviser le bloc cible en quatre sous-blocs cible et les ajouter dans l'ensemble des blocs cible à traiter.
- Stocker la référence du meilleur bloc source et les coefficients s et o correspondants.

Dans cet algorithme, d_{L2} désigne la distance euclidienne.

4. Réduction du temps de calcul par l'approche proposée

Dans une recherche exhaustive (RE) ou dans les méthodes de classification utilisées pour l'accélération du codage fractal d'images, en l'occurrence CF, un seul dictionnaire est utilisé. Ce dictionnaire [D(P)], de pas de chevauchement de blocs source fixé à P, permet de générer les blocs source qui vont être comparés aux blocs cible. Pour réduire le temps de calcul, nous avons proposé une approche qui utilise deux dictionnaires de pas de chevauchement différents et une approximation de l'image en deux étapes (AP2D). AP2D peut être utilisée avec RE ou appliquée aux méthodes de classification ou aux méthodes de restriction de la taille du dictionnaire. Dans cette approche, nous manipulons deux dictionnaires D(P') et D(P). Le pas P' doit être choisi supérieur à P pour générer un dictionnaire de cardinal inférieur au cardinal de D(P) dans le but de chercher les blocs cible qui peuvent être bien approximés avec D(P') avant d'utiliser D(P). En effet, le nombre de comparaisons entre blocs cible et blocs source utilisant D(P') est inférieur au nombre de comparaisons utilisant D(P) (puisque P'> P). Nous avons opté pour P'=2P pour avoir une bonne réduction du temps de codage sans dégradation remarquable de la qualité de l'image. Dans la première étape de AP2D, nous obtenons une approximation de l'image en comparant les blocs cible avec les blocs source du dictionnaire D(2P). Quant aux blocs cible mal approchés résultant de cette première approximation, ils sont comparés aux blocs source du deuxième dictionnaire D(P) en omettant de ce dernier les blocs source du dictionnaire D(2P). De cette manière, nous diminuons le nombre de comparaisons entre blocs cible et source, et par conséquent le temps de codage est aussi réduit.

En effet, si nous considérons une image de taille NxN et des blocs cible de taille nxn. Le nombre de blocs cible, N_{BC}, est donné par :

$$N_{BC} = \left(\frac{N}{n}\right)^2 \tag{6}$$

Le nombre de comparaisons, NC_{RE}, entre blocs cible et blocs source appartenant au dictionnaire D(P), dans RE est donné par :

$$NC_{RE} = N_{BC} \times \left(\frac{N-2n+1}{p}\right)^2 \tag{7}$$

D'autre part, le nombre de comparaisons, NC_{AP2D}, entre blocs cible et source dans AP2D, est donné par :

$$NC_{AP2D} = \underbrace{N_{BC} \times \left(\frac{N-2n+1}{2p}\right)^2}_{(a)} + \underbrace{N_{BM} \times \left(\left(\frac{N-2n+1}{p}\right)^2 - \left(\frac{N-2n+1}{2p}\right)^2\right)}_{(b)} \tag{8}$$

N_{BM} est le nombre de blocs cible mal approchés issus de la première approximation de l'image avec le dictionnaire D(2P). Le terme (a) correspond au nombre de comparaisons dans la première approximation et le terme (b) correspond au nombre de comparaisons des blocs cible mal approchés avec les blocs source issus de D(P) en omettant les blocs source issus de D(2P) puisqu'ils étaient déjà comparés avec ces derniers. Du fait que $N_{BC}=N_{BA}+N_{BM}$ (N_{BA} est le nombre de blocs cible bien approchés lors de la première approximation) et après simplification, (8) devient :

$$NC_{AP2D} = \frac{N_{BA}}{4} \times \left(\frac{N - 2n + 1}{p} \right)^2 + N_{BM} \times \left(\frac{N - 2n + 1}{p} \right)^2 \qquad (9)$$

D'après (7) et (9), nous déduisons que NC_{AP2D} est inférieur à NC_{RE}.

5. Algorithme de l'approche proposée AP2D

L'algorithme de l'approche peut se résumer en trois étapes :

Etape 1 :

- Fixer un seuil d'erreur T ;
- Choisir un pas P de chevauchement de blocs source ;
- Construire le dictionnaire D(2P) ;
- Eliminer les blocs ayant des caractéristiques données (variance faible, forte entropie,…) si on applique AP2D à une méthode de restriction du dictionnaire ;
- Déterminer la classe de chacun des blocs si on applique AP2D à une méthode de classification ;
- Partitionner l'image en bloc cible R_i jusqu'à une profondeur minimale et initialiser une liste chaînée L à vide ;
- Pour chaque bloc R_i restant à traiter :
    - Déterminer la classe de R_i si on utilise une méthode de classification ;
    - Chercher le bloc source D_j de même classe que le bloc R_i tel que la distance $d_{ij}= d_{L2}(w_i(D_j), R_i)$ soit minimale ;
    - Retenir le bloc source et les coefficients de w_i tels que d_{ij} correspondante est minimale ;
    - Si d_{ij} est inférieur à T :

-Insérer en fin de la liste L, la référence du bloc cible R_i et le bloc source correspondant et les coefficients de w_i ;

- Sinon :

- Si la taille du bloc R_i est inférieure à la profondeur maximale :

- Diviser le bloc R_i en quatre sous-blocs, supprimer le nœud contenant R_i et le remplacer par quatre nœuds correspondants aux quatre sous-blocs créés. ;

- Sinon {là on a un bloc cible mal approché}

- Insérer en fin de la liste L, la référence du bloc cible R_i et le meilleur bloc source trouvé et les coefficients de la transformation w_i ;

Etape 2 :

- Construire le deuxième dictionnaire D(P) et éliminer les blocs source appartenant à D(2P) et posons ainsi D'(P)= D(P)-D(2P) ;

- Eliminer les blocs ayant des caractéristiques données si on utilise une méthode de restriction du dictionnaire ;

- Déterminer la classe de chacun des blocs restants si on utilise une méthode de classification ;

- Pour chaque nœud de la liste correspondant à un R_i, faire

- Si R_i est mal approché

- Soit $d'_{ij} = \min d_{L2}(w_i(D_j), R_i)$ lors de la première approximation ;

- Chercher le bloc source Dj du dictionnaire D'(P) de même classe que le bloc R_i tel que la distance $d_{ij} = d_{L2}(w_i(D_j), R_i)$ soit minimale ;

- Si $d_{ij} < d'_{ij}$ alors remplacer les anciennes informations par les nouvelles informations concernant le bloc source trouvé ;

Etape 3 :

- Stocker le code de l'image en parcourant la liste chaînée L de début vers la fin.

6. Résultats expérimentaux

Les différents tests sont effectués sur les trois images représentées sur la figure 1 avec 8 bpp sur un AMD Athlon XP 2000 à 1,6 Ghz et à 256 MO de RAM. Pour évaluer les performances de l'approche proposée, nous avons utilisé le temps de codage, la qualité de l'image et le taux de compression. La qualité de l'image est mesurée en utilisant le rapport signal sur bruit de crête noté PSNR (Peak Signal to Noise Ratio). Le

PSNR entre l'image originale, de taille n x n, composée de pixels A(i, j) et l'image décodée composée de pixels B(i, j) est donné par :

$$PSNR = 10x \log(\frac{255^2}{MSE}) \quad (10)$$

Où

$$MSE = \frac{\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (A(i, j) - B(i, j))^2}{n \times n} \quad (11)$$

Le taux de compression, TC, est égal à la taille de l'image originale divisée par la taille de l'image compressée.

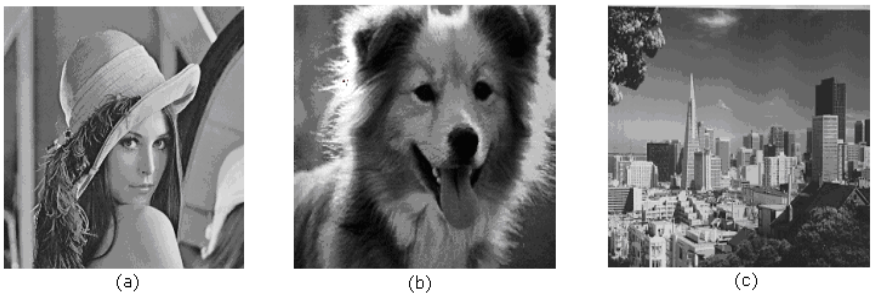

Figure 1. *Images, de taille 256x256, de : Lenna (a), Collie (b) et San256 (c).*

Une comparaison du temps de calcul, du PSNR et du TC, entre AP2D et RE a été établie dans le cas d'un partitionnement quadtree. Les résultats de cette comparaison sont représentés sur le tableau 1. Le gain en temps de calcul obtenu, pour les images de test, est supérieur à 72,60%. Pour l'image de Lenna, l'application de RE avec un pas égal à 1 a généré un PSNR de 31.57dB, un temps de codage de 670.83s et un taux de compression de 9.87 alors que AP2D a généré un PSNR de 31.31dB, un temps de codage de 168.14s et un taux de compression de 10.32. Par ailleurs, cette réduction n'a pas été obtenue au prix d'une perte de la qualité de l'image. Par exemple pour l'image de Lenna, la qualité n'a pas été altérée (Figure 2) et la chute du PSNR (ΔPSNR) n'a pas dépassé 0.4 dB. De plus, le TC a augmenté légèrement (Tableau 1).

La comparaison entre l'application de AP2D à CF (AP2D-CF) avec CF est représentée dans le tableau 2 dans le cas d'un partitionnement quadtree. Les temps de codage obtenus sont plus courts que ceux obtenus par CF seule. Les gains de temps obtenus, pour les images de tests, sont supérieurs à 65.60%. Ce résultat suggère que

l'application d'AP2D à d'autres méthodes de classification serait aussi efficace à réduire le temps de codage. Il n'y a pas de perte de la qualité de l'image comme le montre la figure 3.

Image	RE				AP2D			Gain en temps	ΔPSNR
	Pas	Temps (s)	PSNR	TC	Temps (s)	PSNR	TC		
Lenna	4	36,95	30,92	10,46	11,02	30,52	10,73	70,18%	0,4
	3	71,66	31,07	10,41	18,72	30,70	10,58	73,88%	0,37
	2	152,43	31,31	10,32	41,21	30,92	10,46	72,97%	0,39
	1	670,83	31,57	9,87	168,14	31,31	10,32	74,94%	0,26
Collie	4	30,77	32,72	13,98	8,70	32,60	13,73	71,73%	0,12
	3	58,25	32,74	14,14	14,39	32,63	13,84	75,30%	0,11
	2	118,62	32,77	14,14	30,24	32,72	13,98	74,50%	0,05
	1	526,15	32,92	13,74	121,98	32,77	14,14	76,81%	0,15
San256	4	43,14	30,13	8,25	13,28	29,44	8,75	68,78%	0,69
	3	85,19	30,34	8,00	22,78	29,72	8,51	73,26%	0,62
	2	181,37	30,81	7,78	49,70	30,13	8,25	72,60%	0,68
	1	821,90	31,25	7,50	215,21	30,81	7,78	73,81%	0,44

Tableau 1. *Comparaison entre AP2D et RE dans le cas d'une partition quadtree.*

Image	CF				AP2D-CF			Gain en temps	ΔPSNR
	Pas	Temps(s)	PSNR	TC	Temps(s)	PSNR	TC		
Lenna	4	1,03	30,09	9,62	0,36	29,12	9,83	65,05%	0,97
	3	1,86	30,21	9,77	0,58	29,46	9,75	68,81%	0,75
	2	3,81	30,58	9,67	1,23	30,09	9,62	67,72%	0,49
	1	15,00	31,04	9,48	4,44	30,58	9,67	70,4%	0,46
Collie	4	1,11	32,58	12,03	0,34	32,21	11,80	69,37%	0,37
	3	1,95	32,66	12,21	0,56	32,34	11,87	71,28%	0,32
	2	3,82	32,77	12,40	1,07	32,58	12,03	72,99%	0,19
	1	15,54	32,80	12,56	4,12	32,77	12,40	73,48%	0,03
San256	4	1,31	28,78	7,99	0,44	27,49	8,49	66,41%	1,29
	3	2,59	28,97	7,78	0,73	27,96	8,23	71,82%	1,01
	2	5,14	29,68	7,57	1,54	28,78	7,99	70,03%	0,9
	1	22,11	30,34	7,26	6,33	29,68	7,57	71,37%	0,66

Tableau 2. *Comparaison entre AP2D-CF et CF dans le cas d'une partition quadtree.*

Le tableau 3 montre que le nombre de comparaisons a diminué de manière importante par AP2D-CF. Cette diminution est la conséquence de l'utilisation de deux dictionnaires D(2P) et D'(P) [D'(P) = D(P)-D(2P)], au lieu de D(P) seul. En effet, dans la première approximation, le cardinal du dictionnaire D(2P) est inférieur au cardinal de D(P). Les blocs mal approchés, comparés aux blocs source de D'(P), résultant de cette première approximation sont en nombre réduit.

Image	Nombre de comparaisons Entre blocs cible et source		
	P	CF	AP2D-CF
Lenna	4	418652	144065
	3	688502	240109
	2	1 521 161	513815
	1	5 735 113	1836449
Collie	4	392878	124143
	3	6 62028	2 06407
	2	1 420230	512 251
	1	5 168830	1 694 572
San256	4	6 04327	215978
	3	1 050985	369545
	2	2 352518	1 421847
	1	9 166638	5099213

Tableau 3. *Effet de AP2D-CF sur le nombre de comparaisons entre blocs cible et source.*

a. *PSNR :31,31dB, Temps de codage : 152,43s*

b. *PSNR : 30,92dB, Temps de codage : 41,21s*

Figure 2. *Image reconstruite de Lenna 256x256 par RE (a) et AP2D (b) cas d'une partition quadtree.*

a. *PSNR=30,58dB, Temps de codage : 3,81s*

b. *PSNR=30,09, Temps de codage : 1,23s*

Figure 3. *Image reconstruite de Lenna 256x256 par CF (a) et AP2D-CF (b) cas d'une partition quadtree.*

Dans le tableau 4 est illustrée la comparaison entre AP2D et RE dans le cas d'une partition carrée. Le gain de temps pour les images de tests a atteint 71,32%. Une augmentation du TC, variant de 1,13 à 1,51, a été enregistrée. La qualité de l'image n'a

pas été altéré (figure 4) bien qu'on a noté une diminution du PSNR variant de 0,29 à 0,59.

L'AP2D-CF a aussi réduit le temps de codage dans le cas d'une partition carrée (Tableau 5). Cette réduction a atteint 68,35% avec une augmentation du TC variant de 1,13 à 1,53 et une diminution de PSNR variant de 0,42 à 0,79. La qualité de l'image n'a été altérée que légèrement (figure 5).

Image	RE				AP2D			Gain en Temps	ΔPSNR
	P	Temps(s)	PSNR	TC	Temps(s)	PSNR	TC		
Lenna	4	18,58	27,62	18,94	6,33	27,15	20,45	65,93%	0,47
	3	36,89	27,83	18,26	11,06	27,24	19,66	70,01%	0,59
	2	74,72	28,02	17,63	24,63	27,62	18,94	67,03%	0,4
	1	310,81	28,31	16,50	98,89	28,02	17,63	68,18%	0,29
Collie	4	18,55	31,27	18,94	5,86	30,87	20,45	68,41%	0,4
	3	36,95	31,47	18,26	10,09	31,01	19,66	72,69%	0,46
	2	74,56	31,66	17,63	22,14	31,27	18,94	70,30%	0,39
	1	309,27	31,96	16,50	88,69	31,66	17,63	71,32%	0,3
San256	4	18,56	25,06	18,94	6,95	24,69	20,45	62,56%	0,37
	3	36,72	25,15	18,26	12,22	24,74	19,66	66,72%	0,41
	2	74,48	25,38	17,63	27,36	25,06	18,94	63,27%	0,32
	1	310,31	25,79	16,50	111,84	25,38	17,63	63,96%	0,41

Tableau 4. *Comparaison entre AP2D et RE dans le cas d'une partition carrée avec une taille de blocs de 8x8.*

a. *PSNR : 28,02dB, Temps de codage : 74,72s*

b. *PSNR : 27,62dB, Temps de codage : 24,63s*

Figure 4. *Image de Lenna 256x256 reconstruite par RE (a) et AP2D (b) dans le cas d'une partition carrée de taille de blocs 8x8.*

Image	CF				AP2D-CF			Gain en temps	ΔPSNR
	P	Temps (s)	PSNR	TC	Temps (s)	PSNR	TC		
Lenna	4	0,50	26,55	18,94	0,20	25,87	20,47	60,00%	0,68
	3	0,94	26,81	18,26	0,31	26,02	19,67	67,02%	0,79
	2	1,83	27,23	17,63	0,66	26,55	18,94	63,93%	0,68
	1	7,42	27,67	16,50	2,48	27,23	17,63	66,58%	0,44
Collie	4	0,55	30,50	18,95	0,19	29,80	20,47	63,93%	0,7
	3	1,05	30,82	18,27	0,34	29,89	19,67	67,62%	0,93
	2	2,09	31,05	17,64	0,69	30,50	18,95	66,96%	0,55
	1	8,34	31,47	16,50	2,64	31,05	17,64	68,35%	0,42
San256	4	0,56	24,00	18,94	0,23	23,50	20,52	58,93%	0,5
	3	1,08	24,24	18,26	0,39	23,64	19,68	63,88%	0,6
	2	2,13	24,59	17,63	0,83	24,00	18,94	61,03%	0,59
	1	8,58	25,02	16,50	3,25	24,59	17,63	62,12%	0,43

Tableau 5. *Comparaison entre AP2D-CF et CF dans le cas d'une partition carrée avec une taille de blocs de 8x8.*

a. PSNR=27,67dB,
Temps de codage :7,42s

b. PSNR=27,23dB,
Temps de codage : 2,48s

Figure 5. *Image de Lenna 256x256 reconstruite par CF (a) et AP2D-CF (b) dans le cas d'une partition carrée de taille de blocs 8x8.*

7. Conclusion

Nous avons présenté une nouvelle approche (approche AP2D) pour réduire le temps de codage fractal d'images en utilisant deux dictionnaires et une approximation d'images en deux étapes. Le premier dictionnaire, construit avec un pas égal à 2P, sert à une première approximation de l'image. Les blocs mal approchés, résultant de cette dernière, sont comparés aux blocs sources appartenant au dictionnaire construit avec un pas égal à P. Nous avons comparé les résultats obtenus avec AP2D à ceux obtenus par les approches utilisant un seul dictionnaire. La comparaison, en terme de temps de calcul, montre que AP2D appliquée à la classification de Fisher (CF) a permis une réduction de plus de 65% par rapport à CF et de 72% par rapport à une recherche exhaustive. Cette réduction en temps serait la conséquence de la réduction du nombre de comparaisons générées par AP2D. La qualité de l'image n'a été altérée que légèrement par cette réduction du temps. AP2D a de plus permis de préserver le taux de compression et même de l'augmenter légèrement.

8. Bibliographie

[1] Barnsley M. F., Sloan A. D., A better way to compress images, *BYTE magazine*, 1988, pp. 215-223.

[2] Barnsley M. F., Fractal every where, *New-york: Academic Press*, California, 1988.

[3] Davoine F., Antonini M., Chassery J. M., Barland M., Fractal Image compression Based on Delaunay Triangulation and Vector Quantization. IEEE Trans. *image Processing*, 1996, Vol. 5, N° 2, p. 338-346.

[4] Duh D. J., Jeng J. H., Chen S.Y. DCT based simple classification scheme for fractal image compression, *Image and vision computing* 23 , 2005 P. 1115-1121.

[5] Fisher Y., Fractal encoding with quadtree, Chapter 3 in *Fractal Image Compression: Theory and Applications to Digital Images*, Yuval Fisher, Ed, New York: Springer-Verlag, 1995, pp. 55-77.

[6] Fisher Y., Menlove S., Fractal encoding with HV partitions, Chapter 6 in *Fractal Image Compression: Theory and Applications to Digital Images*, Yuval Fisher, Ed, New York: Springer-Verlag, 1995, pp. 119-136.

[7] Fisher Y., *Fractal Image Compression : Theory and Application,* Springer-verlag, 1995 , New York: Springer-Verlag, 1995, 341 P.

[8] Hassaballah M., Makky M.M., Mahdy Y., *A Fast Fractal Image Compression Method Based Entropy, Electronic Letters on Computer Vision and Image Analysis 5(1):30-40, 2005.*

[9] Jacquin A. E., A fractal theory of iterated Markov operators on spaces of measures with applications to digital image coding, PhD Thesis, Georgia Institute of Technology, 1989.

[10] Jacquin A. E., Image coding based on a fractal theory of iterated contractive image transformations, *IEEE Trans. Image Processing*, 1992, Vol. 1, N°.1, pp.18-30.

[11] Jacquin A. E., Fractal image coding : A review, in *Proceedings of IEEE*, 1993, Vol. 81, N°.10, pp.1451-1465.

[12] Saupe D., Lean Domain Pools for Fractal Image Compression. *Proceedings IS&T/SPIE1996 Symposium on Electronic Imaging: Science & Technology Still Image Compression* II, Vol. 2669, Jane 1996.

9. Annexe

Dans cette partie nous avons appliqué notre approche à deux dictionnaires (AP2D) à la méthode de Saupe (MS) [12], qui fait partie des méthodes de réduction de la taille du dictionnaire par élimination d'un certain nombre de blocs source. Dans MS, les blocs

source ayant une faible variance, pour un seuil fixé, sont éliminés du dictionnaire. Dans les tests, nous avons utilisé l'écart-type au lieu de la variance. Le tableau 1 présente une comparaison entre MS et AP2D appliquée à MS (AP2D-MS). Le gain de temps obtenu pour l'image de Lenna a atteint 73,12% sans dégradation de la qualité de l'image comme le montre la figure 1 ainsi que la faible diminution, de 0,39, du PSNR. Le taux de compression a quant à lui légèrement augmenté.

MS				AP2D-MS				
Seuil d'écart type	Temps (s)	PSNR	TC	Temps (s)	PSNR	TC	Gain en temps	ΔPSNR
40	29,39	31,12	9,50	8,06	30,75	9,56	72,57%	0,37
30	55,88	31,30	9,92	14,89	30,92	10,00	73,35%	0,38
20	81,81	31,33	10,23	22,16	31,02	10,29	72,91%	0,31
10	107,16	31,34	10,30	28,80	30,95	10,46	73,12	0.39

Tableau 1 : C*omparaison entre MS et AP2D-MS pour l'image de Lenna 256x256.*

Nous avons, de plus, hybridé CF et MS (CF-MS) et nous avons appliqué AP2D à CF-MS (AP2D-CF-MS). Le résultat de ces approches est illustré dans le tableau 2. Le gain obtenu pour l'image de Lenna a atteint 68,81%. La qualité de l'image n'a cependant pas été altérée (figure 2) (ΔPSNR variant de 0,51 à 0,77).

MS-CF				AP2D-CF-MS				
Seuil écart-type	Temps (s)	PSNR	TC	Temps (s)	PSNR	TC	Gain en temps	ΔPSNR
40	1,00	30,05	8,87	0,38	29,28	9,07	62,00%	0,77
30	1,70	30,41	9,25	0,56	29,68	9,34	67,06%	0,73
20	2,34	30,59	9,54	0,75	30,03	9,45	67,95%	0,56
10	2,95	30,61	9,65	0,92	30,10	9,62	68,81%	0,51

Tableau 2 : C*omparaison entre MS-CF et AP2D-CF--MS pour l'image de Lenna 256x256.*

a. Temps de codage : 29,39s,
PSNR : 31,12dB

b. Temps de codage : 8,06s,
PSNR : 30,75dB

Figure 1. *Image de Lenna 256x256 reconstruite par MS (a) et AP2D-MS dans le cas d'une partition quadtree.*

a. Temps de codage : 1s,
PSNR : 30,05dB

b. Temps de codage : 0,38s,
PSNR : 29,28dB

Figure 2. *Image de Lenna 256x256 reconstruite par MS-CF (a) et AP2D-CF-MS dans le cas d'une partition quadtree.*