Compression embarquée temps réel pour caméras rapides

Romuald Mosqueron, Julien Dubois, Michel Paindavoine

Compression embarquée temps réel pour caméras rapides

R.Mosqueron & J.Dubois & M.Paindavoine
Laboratoire Le2i - UMR CNRS 5158, Université de Bourgogne
Aile des Sciences de l'Ingénieur, BP 47870 - 21078 Dijon Cedex,France
Tel: +33 3 80 39 68 50 Fax: +33 3 80 39 59 69
romuald.mosqueron@u-bourgogne.fr

Résumé : Les caméras rapides sont de puissants outils pour étudier, par exemple, la dynamique des fluides ou le déplacement des pièces mécaniques lors d'un processus de fabrication. Nous décrivons dans ce papier, un nouveau type de caméra rapide possédant un fonctionnement original. En effet, outre le fait qu'elle utilise comme d'autres caméras, la grande flexibilité des capteurs CMOS en termes d'acquisition (ROI), elle est novatrice au niveau du transfert des données. Celles-ci pouvant être à la fois traitées et/ou compressées en temps réel au sein même de la caméra. Le transfert peut s'effectuer alors à l'aide d'une simple connection série de type USB 2.0. On réalise ainsi l'économie d'une mémoire embarquée, les données étant directement stockées sur la mémoire d'un PC standard, ce qui permet d'utiliser l'intégralité de ses capacités (grande taille mémoire, évolution constante). En parallèle au développement matérielle de la caméra, nous présenterons les algorithmes de compression intégrés au sein de la caméra, notamment un algorithme nous permettant d'utiliser la caméra à sa plus grande résolution (1280 x 1024 pixels) et avec une fréquence image de 500 par seconde. Son taux de compression est de 20, avec un PSNR supérieur à 30.
Mots-clés : Capteur CMOS, FPGA, Compression d'image , Vidéo rapide.

1 INTRODUCTION

Durant les 15 dernières années, notre laboratoire a travaillé dans le secteur des systèmes de vision rapide [FAUVET, 1990, BOUFFAULT, 1997] et a obtenu des résultats pour des applications biologiques comme l'analyses de contraction des cellules en temps réel [BOUFFAULT, 1994] et l'analyse du mouvement humain [PAINDAVOINE, 1999]. Toutes ces applications ont été développées sur la base de capteurs Fairchild à technologie CCD en mode "binning" et de technologie FPGA de chez Xilinx [XILINX]. Le but de nos systèmes était de produire à bas coût, des caméras rapides (500 images par seconde) utilisant des capteurs standards CCD, avec un module de pré-traitement connecté à un PC.
Durant les 5 dernières années, l'utilisation des capteurs CMOS à la place des capteurs CCDs a facilité le développement des caméras rapides industrielles en offrant des sorties numériques rapides, une flexibilité des modes d'acquisition dont l'acquisition partielle (régions du capteur) et un faible coût de fabrication. L'utilisation de capteurs ayant des résolutions et des fréquences images de plus en plus importantes implique que le flot de données est par conséquent de plus en plus important. En raison de son volume et des caractéristiques des éléments de traitements et de stockage (fréquences de fonctionnement, bande passante), le transfert et le traitement d'un tel flot ne peut alors s'effectuer facilement. Une solution consiste à stocker temporairement ce flot dans une RAM locale rapide. Etant donné la taille limitée de la RAM, le temps d'enregistrement est restreint à quelques secondes (CamRecord de chez Imasys par exemple). Nous proposons une solution alternative permettant un enregistrement continu. Nous avons implanté au sein du FPGA, une compression d'images en temps réel pour réduire ce flot de données. Plusieurs types de compression ont été considérées qui peuvent se regrouper dans deux catégories : les compressions avec pertes ou sans perte. Pour répondre aux besoins d'un maximum d'applications, nous avons étudier différents algorithmes appartenant aux deux catégories. Trois algorithmes ont été comparé : codage par longueur de plage (run-length encoding), codage par bloc et codage par ondelettes.
Pour obtenir un flot de 500 images par seconde au format de 1280×1024 pixels et en raison de l'architecture de notre système (sans mémoire interne, liaison USB2), un taux de compression de 20 est nécessaire. L'association de deux algorithmes de compression, un codage par ondelettes suivit d'un codage par bloc, nous permet d'atteindre cet objectif avec un PSNR supérieur à 30 dB. Ces performances ont été obtenues à partir de l'architecture décrite dans cet article. Celui-ci s'organise de la manière suivante : la description de la caméra rapide est faite en Section 2, les études des algorithmes de compression sont présentés en Section 3, l'implantation et l'expérimentation de ces algorithmes sont décrits dans la Section 4. Enfin, les conclusions et les perspectives seront exposées en Section 5.

2 DESCRIPTION DE LA CAMÉRA RAPIDE

La caméra rapide que nous avons développée utilise un capteur rapide de chez Micron[MICRON] pour l'acquisition des images et un FPGA de chez Xilinx pour le traitement d'images. Dans cette section, le capteur CMOS et le composant FPGA seront décrit respectivement dans la

première et la deuxième partie. Enfin la troisième partie présentera l'architecture globale de notre caméra.

2.1 Acquisition d'image avec un capteur CMOS

Dans le contexte de l'imagerie rapide, les capteurs CMOS présentent quelques avantages par rapport aux capteurs CCD que nous allons récapituler ici :

– Accès à une région de pixels,
– Amplification intra-pixel et convertisseur A/N intégré,
– Pas d'effet "blooming",
– Basse puissance,

En tenant compte de ces avantages, nous avons utilisé un capteur rapide CMOS MT9M413 de chez Micron en l'implantant dans le schéma de notre caméra rapide. Ce capteur délivre des images 10 bits couleurs ou monochromes avec une résolution de 1.3 megapixels ($1,280H \times 1,024V$) à 500 images par seconde.

2.2 Traitement d'images avec un composant FPGA

Le capteur rapide utilisé en mode pipeline délivre un flot de 500 images par seconde soit plus de 5 Gigabits par seconde. Pour pouvoir traiter ce flot de données, il est nécessaire d'introduire à l'intérieur de la caméra un élément de calcul (processeur, FPGA) capable de traiter ces données en temps réel. Plusieurs solutions sont envisageables et l'une d'elles est l'utilisation d'un FPGA.

2.2.1 Les avantages du FPGA pour le traitement d'images en temps réel

La majeure partie des traitements d'images de bas niveau peut être séparée en deux types de tâches. Le premier type de tâches est assimilable à des opérations à coefficient fixe qui sont effectuées identiquement sur chaque pixel de l'image. Le deuxième type regroupe des traitements utilisant le voisinage immédiat de chaque pixel de l'image (convolution). Dans ce cas, le résultat généré pour chaque pixel est lié à une fenêtre de pixels centrés autour de celui-ci. Ces opérations prouvent qu'il y a un degré élevé de répétition du traitement à travers l'image entière. Ce type de traitement se prête à une implantation basée sur un modèle en pipeline sur un FPGA qui peut exécuter les mêmes opérations mathématiques sur un flot de données.

Les FPGAs, tel que la série Virtex II de chez Xilinx, fournissent un grand nombre de blocs logiques où chaque bloc contient plusieurs bascules et LUTs capables de mettre en application de nombreuses fonctions logiques. En outre, des ressources dédiées pour la multiplication et le stockage mémoire sont disponibles et permettent d'améliorer l'exécution et les performances. De telles ressources matérielles permettent l'implantation des traitements d'images aux débits très élevés, avec des taux atteignant plusieurs centaines de MHz. Grâce au mode pipeline, ces tâches peuvent être directement exécutées sur le flot de don-nées provenant de la caméra sans incorporer de délai de traitement supplémentaire. Les possibles "goulots d'étranglements" du flot de données, qui limitent les performances, peuvent être ainsi réduits et dans certains cas supprimés. Ainsi, les tâches plus complexes telles que la convolution peuvent être implantées avec succès sur les FPGAs. Le procédé global de convolution est une multiplication de matrice et exige plusieurs multiplications afin d'être exécutée pour chaque pixel. Le nombre exact de multiplieurs nécessaires dépend de la taille de la fenêtre utilisée pour la convolution. Pour une fenêtre 3x3, 9 multiplieurs sont nécessaires, et pour une fenêtre 5x5, 25 multiplieurs sont requis, etc... Les FPGAs considérés précédemment possèdent un grand nombre de multiplieurs. Par exem-ple, avec le Virtex-II-1000 (un million de portes logiques), 40 multiplieurs sont disponibles et dans la version 8000 (huit millions de portes), on augmente ce nombre à 168.

2.2.2 Caractéristiques principales du FPGA utilisé

Pour le traitement d'un tel débit et d'une telle résolution, nous avons choisi un FPGA VIRTEX-II XC2V3000 de Xilinx avec les spécifications suivantes :

– 3,000,000 portes (14,336 cellules),
– 96 multiplieurs 18-bit×18-bit,
– 1,728 Kbits de RAM double port (soit 18Kbits),
– 720 entrées/sorties.

2.3 Système caméra rapide

La caméra rapide est composée de trois cartes comme l'illustre la Figure 1.

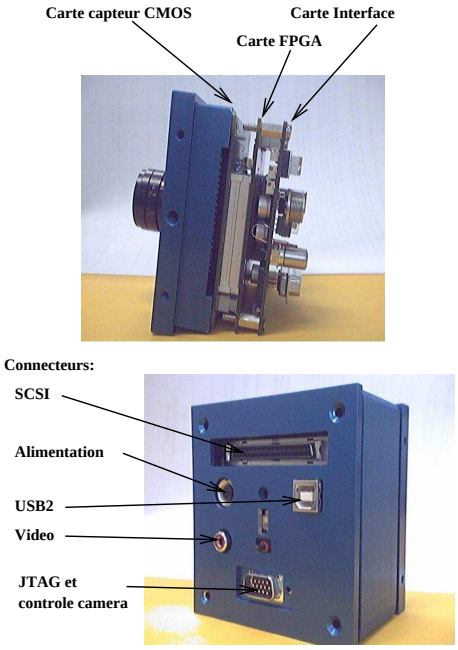

FIG. 1 – *Système caméra rapide*

La Figure 2 représente le schéma bloc de la caméra. La première carte contient le capteur CMOS, elle même connectée à la carte FPGA. Cette seconde carte a trois fonctions :

– mettre en forme les signaux de contrôle du capteur CMOS,
– réaliser la compression d'images en temps réel,

– permet un traitement d'images comme le suivi de mouvement, la détection de contours ou d'autres applications.

La troisième carte nommée carte d'interface contrôle le transfert en temps réel des images entre la caméra et l'ordinateur en utilisant le protocole USB2. Elle alimente aussi le dispositif.

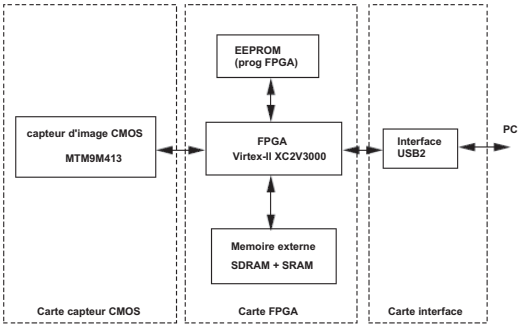

FIG. 2 – *Schéma bloc de la caméra rapide*

3 DESCRIPTION DE L'ALGORITHME DE COMPRESSION

Le capteur rapide de technologie CMOS délivre des images avec une résolution de 1024×1280 pixels à 500 images par seconde soit un taux de transfert de 655 Mpixels/s. Chaque pixel étant codé sur 10 bits, le taux de transfert est de 655 Mpixels/s$\times$ 10 bits $= 6.55$ Gbits/s.
Comme décrit précédemment, les données sont envoyées depuis notre caméra rapide vers un PC via une liaison USB2 avec un taux de transfert de 340Mbits/s à 480Mbits/s. Pour envoyer des images entières en temps réel, il est nécessaire de compresser les données. Dans notre cas , le taux de compression est de $\frac{6.55Gbits/s}{340Mbits/s} = 19.5$.
Deux approches principales sont utilisées dans les algorithmes de compression : compression à faibles per-tes et compression sans perte. Le but principal de la compression sans perte est de minimiser le nombre de bits nécessaire pour représenter les échantillons de l'image originale sans aucune perte de données. Tous les bits de chaque échantillons doivent être reconstruits parfaitement durant la décompression. Quelques algorithmes sans perte connus basés sur la compression "sans erreurs" ont été considérés : le codage Huffman [HUFFMAN, 1952] ou le codage LZW[LEMPEV, 1977]. Ces algorithmes sont particulièrement utiles dans l'archivage d'images tel que le stockage de données médicales ou légales. Dans ce cas précis, le taux de compression est faible (compris entre 2 :1 et 3 :1). Les algorithmes de compression à faibles pertes génèrent des taux de compression plus importants, typiquement de 10 :1 à 100 :1 voire plus. En général, plus le taux de compression est important, moins la qualité d'image est bonne. Quelques méthodes connues, développées pour des applications multimédia, ont été présentées : JPEG, JPEG2000, MPEG2, MPEG4... Ces méthodes sont basées sur des algorithmes spatio-temporels et utilisent différentes approches, telles que le codage prédictif, le codage par transformée (transformée de Fourier, transformée en cosinus discrète), et le codage par ondelettes.
Le choix de notre méthode de compression est basé sur deux contraintes principales. La première concerne la considération "temps-réel". Comment compresser 500 images/s en temps réel ? La deuxième contrainte est liée à la qualité d'image : notre but est, dans ce cas, de compresser et de décompresser des images de manière à obtenir un PSNR[1] plus élevé que 30dB. Pour l'aspect temps-réel, la compression est réalisée avec le composant FPGA, et la décompression est réalisée par le PC après que la séquence d'image ait été enregistrée.
Comme le montre la figure 3, nous combinons la compression sans pertes et la compression à faibles pertes. La compression sans perte, basée sur l'algorithme de Huffman, génère un taux de compression proche de 2 :1. Donc la compression à faibles pertes doit générer un taux de compression proche de 10 :1. Pour atteindre cet objectif, nous avons étudié 3 algorithmes de compression qui sont compatibles avec une implantation sur FPGA : Codage par bloc, Codage par longueur de plage à une dimension (One-dimensional run-length coding), codage par ondelettes utilisant le "lifting-scheme". Nous décrivons ces algorithmes dans les parties suivantes.

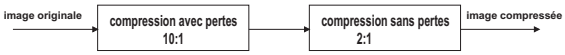

FIG. 3 – *Synoptique de compression*

3.1 Codage par bloc (Block coding)

Cette méthode de compression consiste à traiter l'image avec une fenêtre de $n \times n$ pixels. En accord avec les spécifications FPGA décrites dans le chapitre précédent, il est possible de stocker dans ce composant 8 lignes de 1280 pixels chacune, donc nous avons choisi $n = 8$. Pour chacune des fenêtre de $n \times n$ pixels, nous testons l'uniformité des pixels. Si les pixels ne sont pas uniformes, nous divisons cette fenêtre en sous-fenêtre de 4×4 et 2×2 pixels et nous retestons l'uniformité dans ces nouvelles fenêtres. Dans la figure 4, nous donnons un exemple de cette méthode. Si nous considérons les pixels $P(1,1), P(1,2), P(2,1)$ et $P(2,2)$ dans la fenêtre de 2×2 pixels, nous pouvons effectuer les opérations suivantes :
$P_{moy} = \frac{P(1,1)+P(1,2)+P(2,1)+P(2,2)}{4}$
$if P(i,j) \leq P_{moy}$ alors $Diff(i,j) = P_{moy} - P(i,j)$ et

[1]PSNR (Peak Signal to Noise Ratio) est calculé comme suit : $PSNR = 10log_{10}\frac{(2^B-1)^2}{MSE}$ où B représente le nombre de bits de l'image originale. MSE (Mean Square Error) est calculé de la manière suivante $MSE = \frac{1}{Npixels}\sum_{x,y}(f(x,y) - g(x,y))^2$ où Npixels est le nombre de pixels de l'image, f(x,y) et g(x,y) sont respectivement les niveaux de gris de l'image originale et de l'image traitée aux coordonnées x et y. Les images reconstruites sont obtenues après compression-décompression

$Sign = 0$
$if P(i,j) \geq P_{moy}$ alors $Diff(i,j) = P(i,j) - P_{moy}$ et $Sign = 1$ avec $i, j = 1..2$
le code résultant est :
P_{moy},$Diff(1,1)$,$Diff(1,2)$,$Diff(2,1)$,$Diff(2,2)$, $Sign(1,1)$,$Sign(1,2)$,$Sign(2,1)$,$Sign(2,2)$.
Comme chacun des pixels originaux $P(1,1), P(1,2)$, $P(2,1)$ et $P(2,2)$ sont codés sur 10 bits, la sous-fenêtre originale de 2×2 pixels contient 40 bits. Si nous codons P_{moy} sur 10 bits, $Diff$ sur 2 bits et le $Sign$ sur un bit, la taille du code obtenu est $(1 \times 10) + (4 \times 2) + (4 \times 1) = 22$ bits et le taux de compression théorique est de $\frac{40}{22} = 1.81$.

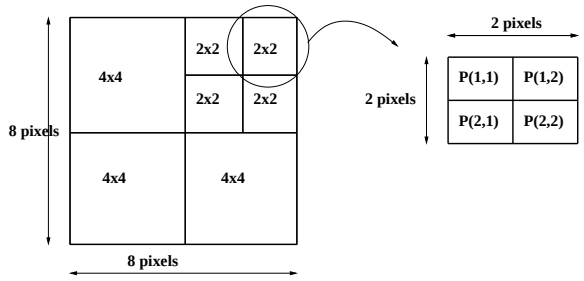

FIG. 4 – *principe du codage par bloc*

3.2 Codage par longueur de plage à une dimension (One-dimensional run-length coding)

Dans cette méthode nous considérons les variations entre pixels voisins d'une même ligne d'image. Si les variations entre pixels voisins sont petites, nous fusionnons ces pixels sur le même segment avec une valeur unique de référence de niveau de gris. La figure 5 est une illustration de cette méthode. Pour chaque pixel, nous exécutons les tests suivants :
Si $r_j - e \leq g_i \leq r_j + e$ alors pixel(i) confondus avec r_j sinon $r_j = g_i$
avec g_i le pixel en niveau de gris courant, r_j le niveau de gris de référence du j^{eme} segment, et e l'échelle d'erreur. Le code obtenu est : $r_1, n_1, r_2, n_2, ..., r_{N_{seg}}, n_{N_{seg}}$ avec n_j la taille du j^{eme} segment et N_{seg} le numéro du segment détecté sur la ligne actuelle.
Si nous codons le pixel de référence (r_j) sur 10 bits et la taille du segment (n_j) sur 5 bits, le taux de compression pour une ligne de n pixels (ici 1280) est $\frac{10 \times n}{\sum_{j=1}^{N_{seg}} (10+5)}$.
Ce taux est variable en fonction du contenu de l'image ; Si l'image a beaucoup de variations, alors le taux de compression baisse.

3.3 Codage par ondelettes utilisant le lifting-scheme (Wavelet coding using lifting-scheme)

Cette approche de compression utilise la théorie des ondelettes qui a d'abord été introduite par Grossman et Morlet[GROSSMAN, 1984] en 1984 pour étudier les signaux de réflexions sismiques des applications géophysiques, et elle a été ensuite appliquée à l'étude du son et de l'image. Beaucoup d'auteurs proposent différentes fonctions par ondelettes, et certaines d'entres

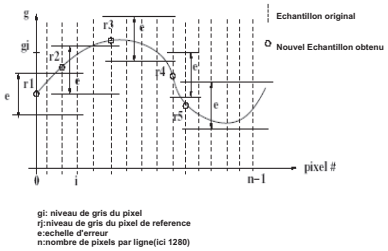

FIG. 5 – *Principe du codage par longueur de plage*

elles ont des applications très intéressantes pour l'analyse d'image en multirésolution[MALLAT, 1989].
L'avantage du codage de la transformée en ondelettes pour la compression d'images réside dans le fait que les coefficients d'ondelettes décorrèlent les pixels de l'image et ainsi ils peuvent être codés plus efficacement que les pixels originaux. La figure 6 est une illustration de la transformée en ondelettes à une dimension avec 3 niveaux de décomposition. L'histogramme de l'image originale montre que la distribution des niveaux de gris est relativement large (échelle de 0 à 255) tandis que l'histogramme des coefficients d'ondelettes est plus fin et centré sur 0. En utilisant cette propriété, des coefficients d'ondelettes peuvent être codés avec une plus grande efficacité que les pixels de l'image originale.
Dans l'objectif d'implanter la transformée en ondelettes avec une compatibilité des contraintes matériel-les, nous avons utilisé une approche en lifting-scheme proposé par Sweldens [SWELDENS, 1995] en 1995. Cette méthode d'implantation de la transformée en ondelettes est décrite en Figure 7, où nous considérons les pixels de l'image d'origine comme un mode de flot de données (dans une représentation 1D). L'approche du Lifting-scheme à une dimension(LS_1D) est décomposée en trois blocs principaux : Split, Predict and Update. Le bloc Split sépare les pixels en deux signaux : les pixels pairs et impairs. Les blocs Predict and Update sont de simples filtres FIR du premier ordre permettant d'obtenir deux sorties : les images de détails (coefficients d'ondelettes) et les images d'approximation utilisées pour le prochain étage LS_1D comme montré en Figure 8. Dans cette figure, un algorithme pyramidale est décrit où trois blocs LS-1D sont mis en cascade et cela nous donne une transformée en ondelettes avec trois niveaux de coefficients.

4 IMPLANTATION ET RÉSULTATS EXPÉRIMENTAUX

Les 3 algorithmes de compression ont été validés de manière logicielle à partir de matlab 6.5[2]. Au vue des performances, nous avons couplé deux algorithmes, le codage en ondelettes est réalisé au préalable pour augmenter les performances de la compression par blocs. Actuellement ce logiciel sert à la reconstruction des images et aux tests des performances. Les résultats obtenus sont

[2]Matlab 6.5 is from Mathworks, Inc.

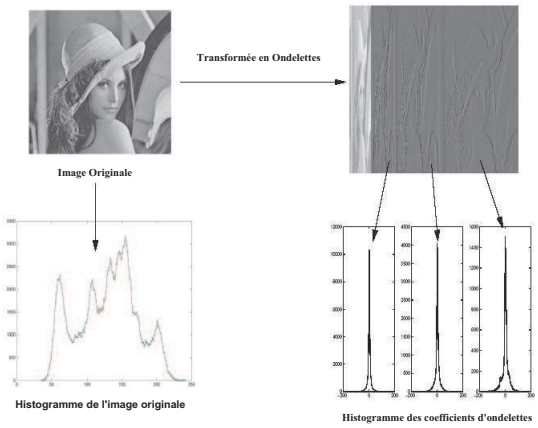

FIG. 6 – *Transformée en ondelettes*

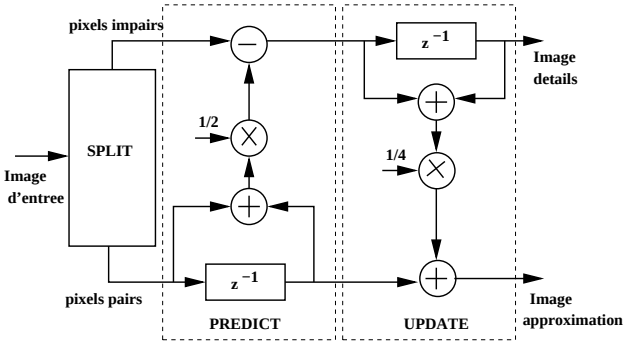

FIG. 7 – *Lifting Scheme 1D (LS1D)*

illustrés sur les figures 9 et 10 qui sont des images d'une taille de 512∗512. Sur la Figure 9, il apparaît que l'algorithme de codage par bloc donne un taux de compression de 2.5 avec un PSNR de 25.4 dB. Pour la même image, l'algorithme de codage par longueur de plage donne un taux de compression de 2.27 avec un PSNR de 29dB. Ces résultats montrent que la qualité d'image (PSNR) est meilleure avec l'algorithme de codage par longueur de plage, mais dans certains cas les taux de compression sont trop bas par rapport à notre objectif.

La figure 10 illustre les résultats obtenus avec l'algorithme de compression en ondelettes. Dans cette approche, un seuillage est appliqué sur les coefficients d'ondelettes pour éliminer les valeurs faibles, et ils sont codés en utilisant la méthode du codage par bloc. Cette figure montre qu'avec un objectif de taux de compression de 10 :1, la qualité de l'image reste bonne et correspond à nos objectifs(PSNR $>$ 30dB). Tout ces algorithmes ont été implantés de manière séparé et nous permet d'obtenir les résultats attendus. La combinaison des 2 algorithmes permettant la compression est en cours de validation pour son implantation, ce qui ne nous permet pas d'avoir d'images.

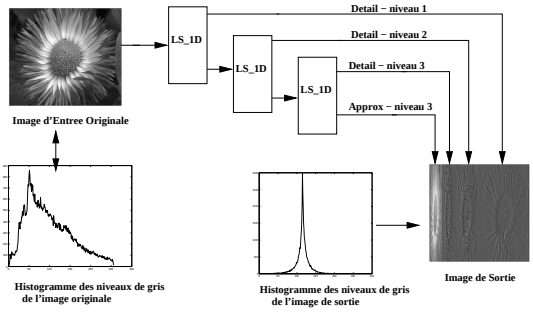

FIG. 8 – *Algorithme pyramidale de Lifting-scheme multi-résolution : exemple avec 3 niveau de décomposition*

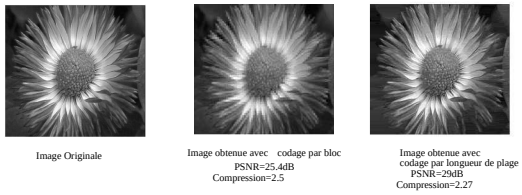

FIG. 9 – *Résultats obtenus avec le codage par bloc et le codage par longueur de plage*

5 CONCLUSION ET PERSPECTIVES

Dans cet article, nous avons montré qu'il était possible d'implanter une compression temps réel basée sur un codage en transformée en ondelettes dans un FPGA. Nous proposons donc une caméra rapide basée sur un capteur CMOS et un composant FPGA. Les performances de cette caméra sont les suivantes : capturer les images d'une résolution de 1280$\times$ 1024 pixel à la vitesse de 500 images par seconde, et transmettre en temps réel par liaison USB2 des images codées avec un taux de compression de 20 :1 et un PSNR supérieur à 30 dB. Avec de telle performances, il est possible de mémoriser de longues séquences d'images directement sur le PC sans utiliser de mémoire interne spécifique, ce qui est un avantage car nous pouvons profiter des améliorations constantes des PCs, spécialement sur les mémoires. La figure 11 représente une séquence d'images non compressée avec une résolution de 128x640 pixels et une fréquence de 250 images par seconde ce qui permet déjà de faire un bon nombre d'applications.

Dans le futur, nous aurons une nouvelle version avec une liaison Ethernet 1 Gigabit, ce qui devrait réduire notre taux de compression (6.5 :1) et ainsi améliorer notre qualité d'image (PSNR augmenté). Avec cette approche, il nous sera possible de créer de nouvelles caméras avec des capteurs plus rapides (comme 1Mpixels à 1000 images par seconde ou plus) ou de plus grandes résolutions.

En utilisant la technologie FPGA, l'intégration de nouveaux traitements d'images temps réel à l'intérieur de la caméra sont alors possible, par exemple le suivi d'objet (tracking), l'analyse d'image, ou reconnaissance de modèles.

FIG. 10 – *Résultats obtenus avec le codage de compression en ondelettes*

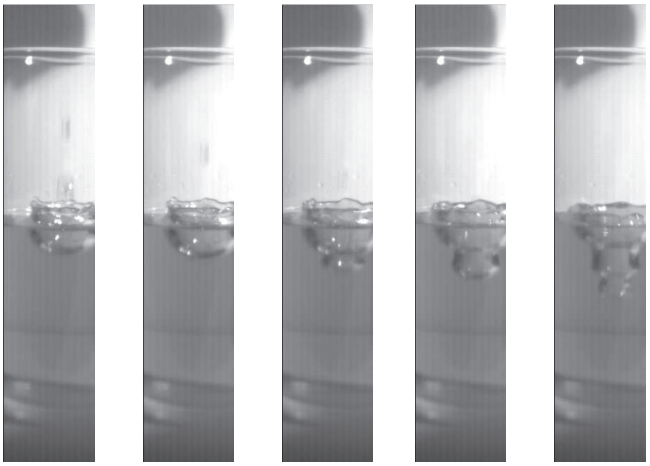

FIG. 11 – *Séquence d'images d'une goutte tombant dans un verre d'eau avec une résolution de* 128×640 *pixels à 250 images par seconde*

BIBLIOGRAPHIE

[BOUFFAULT, 1994] F. Bouffault, C.Milan, M.Paindavoine, J. Febvre, G. Cathebras, *High speed cameras using CCD image sensor and new high speed sensor for biological applications*, 21th International Congress on High-Speed Photography & Photonics, Taejon , September 1994.

[BOUFFAULT, 1997] F.Bouffault, J.Febvre, C.Milan, M.Paindavoine, J.C.Grapin *High speed video microsystem*, Measurement Science and Technology Journal (IOP Publishing), vol. 8, n˚5, pp. 398-402, may 1997.

[FAUVET, 1990] E.Fauvet, M.Paindavoine, F.Cannard *Fast image analysis system*, 19th International Congress on High-Speed Photography & Photonics Cambridge (UK) - Septembre 1990.

[GROSSMAN, 1984] D.Grossman and J.Morlet, *Decomposition of Hardy functions into square integrable wavelets of constant shape* SIAM J.Math, 15, 1984, pp.723-736.

[HUFFMAN, 1952] D.A.Huffman, *A method for the construction of minimum redundancy codes*, Proc.IRE 1952, vol.40, no10, pp1098-1101

[LEMPEV, 1977] J.Ziv, A.Lempel, *A universal algorithm for sequential data compression*, IEEE Transactions on information theory, 1977

[MALLAT, 1989] S.Mallat, *A theory for multiresolution signal decomposition : the wavelet representation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.11, 1989, pp 674-693

[MICRON] Micron Company, *MT9M413 CMOS Image Sensor*, http ://micron.com/products/imaging/

[PAINDAVOINE, 1999] M.Paindavoine, D.Dolard, JC.Grapin, *Real-time imaging system applied to human movement analysis*, SPIE International Symposium on Optical Science and Technology, Denver, USA, July 1999

[SWELDENS, 1995] W.Sweldens, *The lifting scheme : A new philosophy in biorthogonal wavelet constructions*, Wavelet applications in signal and image processing III, pp.68-79, Proc. SPIE 2569, 1995

[XILINX] Xilinx Company, *FPGAs devices*, http ://www.xilinx.com