Codeur/décodeur vidéo FGS à bas débit

WAVIX

Table des matières

I. Introduction

Le domaine de la compression vidéo a connu, ces dernières années, de fortes évolutions menant à l'émergence d'un nombre important de standards internationaux (H.26X, MPEG-X). Malgré le nombre important de solutions, la compression reste un domaine de recherche ouvert notamment dans le cadre de transmision audiovisuelle sur différents types de canaux filaires ou non. L'arrivée de ce type d'infrastructures a également été à l'origine de nombreux travaux visant à optimiser la qualité de service de bout-en-bout. Ces travaux concernent la compression bas débit mais également la résistance aux pertes et aux erreurs et, plus généralement, la flexibilité d'adaptation des flux compressés aux caractéristiques non stationnaires du réseau. En particulier, le concept de scalabilité à grain fin (FGS) a été introduit afin de permettre, notamment, l'adaptation de flux pré-encodés dans des scénarios de streaming.

Le codeur et le décodeur vidéo faisant l'objet du dépôt APP se place dans le cadre de la compression bas débit avec en outre des fonctionalités de représentation scalable à grain fin. Il met en oeuvre les modules fonctionnels suivants:

- un estimateur de mouvement hiérarchique contraint en débit;
- des outils de codage des champs de mouvement et de la structure décrivant les tailles de blocs variables utilisées dans l'analyse du mouvement;
- une décomposition par ondelettes temporelles compensées en mouvement sur un groupe d'images (GOF), avec plusieurs options d'analyse de mouvement (estimation de mouvement arrière, estimation avant OU arrière) et de filtrage temporel (Haar, lifting tronqué 5-3 ou 9-7);
- une option d'analyse du GOF permettant de réduire la distance temporelle entre les paires d'images considérées dans l'analyse de mouvement;
- une décomposition spatiale avec une technique de lifting 9-7;
- une prédiction Inter-GOF (Inter groupes d'images) en boucle fermée et compensée en mouvement;
- un codage de résidu de la prédiction inter-GOF;
- une régulation de débit par groupe d'images.

Le groupe de bandes de fréquences temporelles incluant le résidu de la prédiction décomposé spatialement conduit à des sous-bandes spatio-temporelles qui sont ensuite codées par l'algorithme EBCOT [**?**]. On utilise ici le logiciel du modèle de vérification de JPEG-2000 (VM JPEG-2000). Les résultats d'expérimentation montrent un gain important par rapport à MPEG-4 part 2 et proches de ceux obtenus avec H.264.

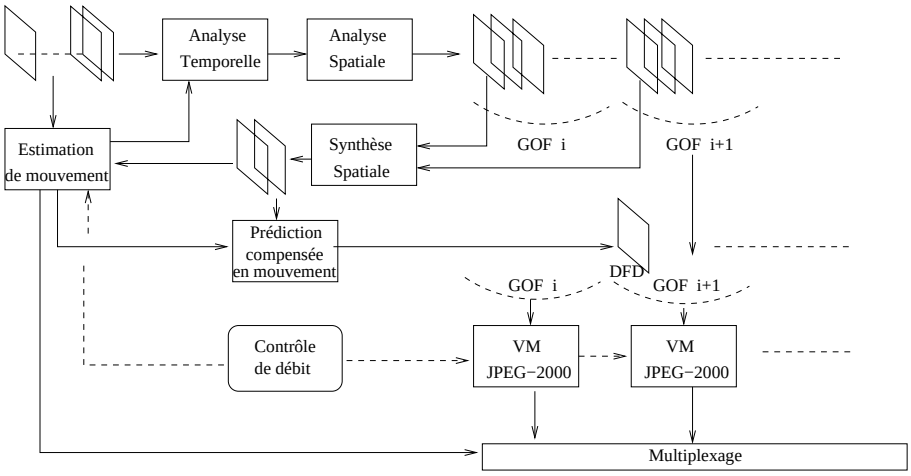

Fig. 1. Architecture sans codage de résidu.

II. Structure générale

L'architecture du codeur proposé ici est donné sur les figures 1 et 2, respectivement pour les options sans et avec codage du résidu de la prédiction Inter-GOF compensée en mouvement. Les images sont traitées par groupe appelé GOF. Après une phase d'estimation de mouvement, les GOF passent par une phase de transformation temporelle compensée en mouvement. Chaque sous-bande temporelle est ensuite décomposée par un filtre d'ondelettes bi-orthogonal 9-7 dans les directions horizontale et verticale, à l'aide d'un schéma de lifting. Les sous-bandes spatio-temporelles sont ensuite quantifiées uniformément. Enfin, celles-ci sont encodées en utilisant le logiciel du modèle de vérification de JPEG-2000 qui comprend un codeur arithmétique basé contexte EBCOT permettant d'obtenir un train binaire scalable à grain fin.

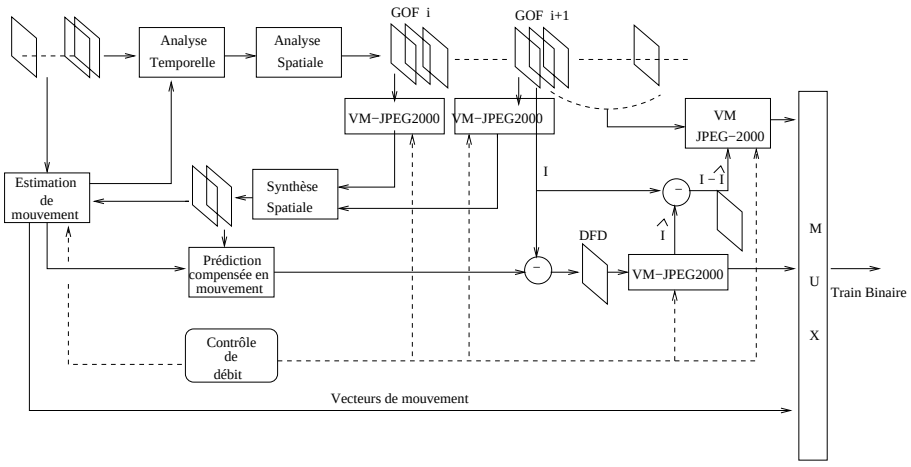

Fig. 2. Architecture avec codage de résidu.

III. Estimateur de mouvement

Afin de réguler finement le débit alloué aux champs de mouvement, l'estimation de mouvement basée bloc utilise une structure d'arbre contraint en débit (i.e. *Quadtree*). La taille des blocs est ensuite adaptée aux caractéristiques des mouvements locaux au sens débit-distorsion. Le débit fait reférence ici au budget (en bits) alloué à l'encodage des vecteurs mouvement et la distorsion fait reférence à l'EQM résultante. Afin d'accélerer l'estimation de mouvement, nous utilisons une estimation hiérarchique. Les vecteurs mouvement obtenus dans une première étape (à faible résolution) sont ensuite raffinés pour les résolutions supérieures. De plus, les estimations des blocs de tailles importantes, situés dans les premiers niveaux du quadtree, servent de base pour l'estimation de mouvement des blocs de tailles inférieures. Enfin, afin d'obtenir un champ de mouvement plus lissé, limitant les pixels connectés, nous utilisons ici une méthode d'estimation de mouvement avec recouvrement de blocs. Les tailles de blocs varient entre 64x64 et 8x8 et l'estimation est réalisée avec une précision pixelique. Après élagage du quadtree, les vecteurs mouvement sont codés de manière prédictive. Le prédicteur utilisé est la valeur médiane des vecteurs associés aux blocs voisins. L'erreur de prédiction est alors codée par des codes de Huffman.

A. Estimation hiérarchique

Le but est de construire un *quadtree* de vecteurs mouvement associés aux blocs de l'image pleine résolution. On produit tout d'abord une pyramide multirésolution spatiale pour chacune des deux images entre lesquelles le mouvement doit être estimé. On combine ensuite des opérations dites de *raffinement* des vecteurs mouvement entre niveaux de résolution spatiale et des opérations de *découpe* des blocs courants à un niveau de résolution donnée. Les premières opérations sont liées à l'estimation de mouvement hiérarchique alors que les suivantes sont liées à la construction des blocs de taille variable. Le processus d'estimation est illustré pour un bloc sur la figure 3. Nous en donnons ici l'algorithme :

1. Génération des pyramides multirésolution pour chacune des images I_t et I_{t-1}. On obtient alors pour chacune des images les différentes résolution I_t^l avec $l = 0, 1, ..., L$. La résolution L représente l'image de plus faible taille.

2. Estimation du mouvement à la résolution la plus grossière. L'estimation est réalisée ici par bloc de manière classique dans une fenêtre de recherche de dimensions (W_x^{L-1}, W_y^{L-1}) dépendantes du niveau de résolution courant. On obtient alors les vecteurs mouvement (dx^0, dy^0).

3. On découpe chaque bloc en quatre sous-blocs et on estime alors leur mouvement (dx^1, dy^1) avec $i = 0, 1, 2, 3$. Pour cela, on restreint la zone de recherche en forçant comme base de recherche le vecteur obtenu pour le bloc parent. De plus, une fenêtre de taille réduite est utilisée autour de cette position. On compare ensuite, l'erreur d'estimation moyenne des fils avec celle du bloc

père. Si cette dernière est supérieure, le bloc père est réellement *découpé*.

4. On change alors de niveau de résolution. A cette étape, tous les noeuds (internes ou feuilles) de la résolution précédente sont *raffinés*. On raffine alors les vecteurs $(dx^0, dy^0), .., (dx^n, dy^n)$ en mulipliant tout d'abord par deux chacun de ceux-ci. Ce raffinement est complété par une recherche autour de ce vecteur également réalisée dans une fenêtre de taille très réduite (i.e. 1 ou 2 pixels autour).

5. On découpe chaque bloc en quatre sous-blocs et on estime alors leur mouvement (dx^{n+1}, dy^{n+1}) avec $i = 0, 1, 2, 3$. On compare ensuite, l'erreur d'estimation moyenne des fils avec celle du bloc père. Si cette dernière est supérieure, le bloc père est réellement *découpé*.

6. Tant que la résolution courante n'est pas la pleine résolution aller à l'étape 4.

7. Le quadtree initial est ensuite formé à partir des différentes découpes réalisées sur chaque bloc de l'image à pleine résolution.

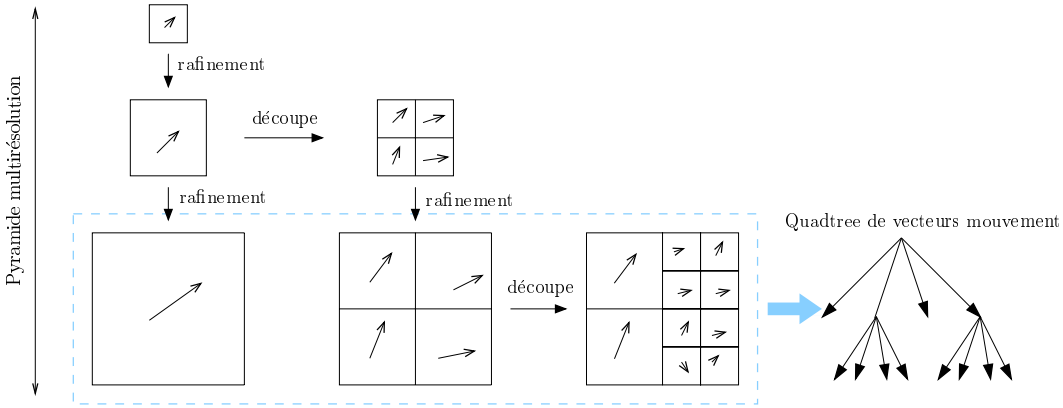

Fig. 3. Construction d'un quadtree de vecteurs mouvement pour un bloc seulement.

L'estimation de mouvement est réalisée ici avec une précision pixelique et en prenant pour critère de mise en correspondance entre blocs la somme des différences absolues SAD. La taille des blocs varie entre 8x8 et 64x64.

Remarque:
Afin de lisser le champ de mouvement nous utilisons une technique de recouvrement de blocs à l'estimation (OBME : *Overlapped Block Motion Estimation*). Ce recouvrement n'est cependant pas réalisé lors des phases de compensation.

B. Quadtree contraint en débit

Le but de la formation d'un quadtree de vecteurs de mouvement est de pouvoir adapter son débit d'encodage en fonction du besoin. Pour cela, il est nécessaire de définir une stratégie d'élagage des

branches de l'arbre permettant d'obtenir le compromis débit-distorsion optimal. Il est également nécessaire d'associer à chaque noeud de l'arbre une fonction de coût. C'est en fonction de ce coût que l'algorithme d'élagage prendra ces décisions.

Classiquement, dans une approche débit-distorsion, on utilise une technique à base de multiplicateurs de Lagrange, ce qui revient à minimiser un critère du type $R + \lambda D$ pour un λ donné, avec R le débit (coût de codage) et D la distorsion associée. Une recherche sur λ est ensuite effectuée jusqu'à atteindre le débit ou la distorsion désirés. L'algorithme procède de la manière suivante.

A chaque noeud de l'arbre est associé un ensemble d'information servant dans le déroulement de l'algorithme. Ainsi au noeud i est associé $\{D_i, R_i, \Delta D_i, \Delta R_i, \lambda_i, \lambda min_i\}$ avec D_i la distorsion du noeud i, R_i son coût de codage, ΔD_i et ΔR_i définis plus haut, $\lambda_i = \Delta D_i / \Delta R_i$ et λmin_i la valeur de λ_j minimale parmi tous les descendants du noeud i. L'algorithme d'élagage est le suivant :

- **Initialisation**
  - Initialisation de chaque noeud avec les valeurs de débit et distorsion associée.
  - Pour chaque feuille, on a $\Delta D_i = 0$, $\Delta R_i = 0$ et $\lambda_i = \infty$.
  - Pour chaque noeud interne, on calcule en partant des feuilles les ΔD_i, ΔR_i et λ_i. Le paramètre λmin_i prend la valeur minimum entre le λ_i courant les λmin_j de ces fils.
  - Calcul du coût R_{glob} et de la distorsion D_{glob} de l'arbre entier.
- **Elagage**

  1. Si le débit R_{targ} et/ou la distorsion D_{targ} cibles ne sont pas atteints, onrecherche le noeud de l'arbre qui offre un λ_i minimum. Cette recherche est facilitée par la valeur λmin_i contenue dans chaque noeud.

  2. Une fois trouvé ce noeud on supprime sa descendance.

  3. Mise à jour des paramètres ΔD_i, ΔR_i, λ_i et λmin_i du noeud courant et de tous les parents de celui-ci.

  4. Nouveau calcul de R_{glob} et D_{glob}. Si $R_{glob} > R_{targ}$ ou $D_{glob} < D_{targ}$ retourner à l'étape 1.

La distorsion D_i retenue est la SAD calculée dans la phase d'estimation de mouvement. Le coût de codage R_i est estimé en se basant sur la table de codes VLC utilisés pour le codage du mouvement dans H.263+. C'est le coût de codage (i.e. la taille du code VLC) du différentiel entre le vecteur du noeud courant et le vecteur du père qui sert pour estimer R_i. Ce coût est raffiné avec celui de la structure de l'arbre à coder. Nous détaillons dans la section suivante le codage de la structure.

IV. Codage des informations de mouvement

Une fois l'arbre élagué, il est nécessaire de coder les informations de mouvement et de former un train binaire en conséquence. La structure du train binaire est composée de deux parties : la structure de l'arbre et les vecteurs mouvement.

A. *Codage de la structure de l'arbre*

L'idée est de coder, pour chaque noeud de l'arbre, l'information de paternité ou non. Ainsi lorsqu'un noeud est parent (i.e. le bloc est subdivisé) on code 1 alors que si c'est une feuille on code 0. Le train binaire correspondant à la structure est alors codé en parcourant le quadtree en largeur d'abord et en insérant les 1 et 0 en conséquence. En pratique, le coût peut encore être réduit par l' exploitation des informations de taille de bloc minimale et maximale. Dans ce cas, on ne commence à coder la structure qu'au niveau des blocs de taille maximale. De plus, aucun bit n'est codé pour les noeuds correspondant à des blocs de taille minimale, car ce sont obligatoirement des feuilles. La figure 4 montre un exemple de codage d'une structure.

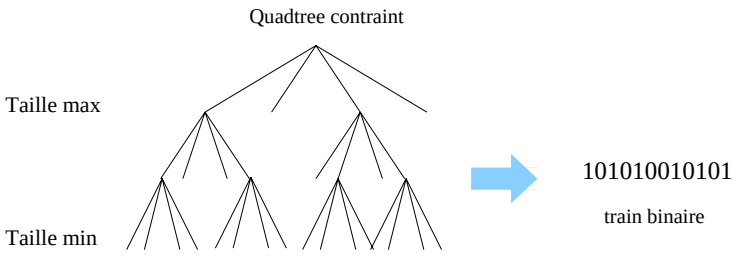

Fig. 4. Exemple de structure codée.

B. *Codage des vecteurs mouvement*

La structure de codage admise dans la phase de contruction et d'élagage du *quadtree* n'est pas exactement celle que nous utilisons. Nous utilisons une technique donnant de meilleures performances. Ceci implique que la phase d'élagage n'est plus tout à fait optimale au sens débit-distorsion. Toutefois les résultats montrent une dérive faible.

Le codage de la structure de l'arbre nous fournit une carte spatiale des blocs et non plus une structure arborescente (i.e. la structure est mise à plat nous permettant d'avoir les tailles respectives de chacun des blocs). Nous codons ensuite les ve cteurs mouvements associés à ces blocs de manière prédictive. Le prédicteur utilisé est la valeur médiane des vecteurs associés aux blocs voisins (cf. figure 5). L'erreur de prédiction est alors codée via les codes VLC du codeur H.263+.

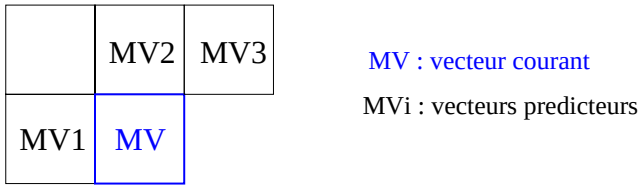

Fig. 5. Prédiction des vecteurs mouvement.

V. Régulation de débit

A l'initialisation, nous attribuons un budget de débit pour chaque GOF en fonction de la contrainte de débit cible globale. La contrainte par GOF R_{GOF} est alors donnée par :

$$R_{GOF} = G_{size} \times R_{target}/Fr$$

avec G_{size} le nombre d'images composant le GOF, R_{target} le débit cible en bits/s et Fr la fréquence temporelle de la source exprimée en nombre d'images/s. Le débit obtenu R_{GOF} est donc exprimé en bits.

Il s'agit ensuite de répartir ce budget entre informations de mouvement et informations de texture. Pour cela un pourcentage de débit p_{mv} est alloué au mouvement. Ce débit est donné par

$$R_{mv} = p_{mv} \times R_{GOF}.$$

Ce budget est ensuite partagé en un débit par champ de mouvement. Ce débit évolue au cours du temps en fonction du débit utilisé par les champs précédents. Une meilleure politique non utilisée ici serait de répartir le débit parmi les n champs de mouvement à coder en fonction de leur coût initial avant élagage. Ceci requiert toutefois que tous les champs de mouvement soient disponibles à $t = 0$, ce qui n'est généralement pas possible.

En pratique, le budget utilisé réellement pour le mouvement est différent de R_{mv} et est noté $\bar{R}_{mv}$. Le budget alloué à la texture est donc :

$$R_{texture} = R_{GOF} - \bar{R}_{mv}.$$

VI. Filtrage temporel compensé en mouvement

Les différentes approches d'analyse temporelle compensée en mouvement utilisées sont fonction des modèles de mouvement sous-jacents.

A. Estimation de mouvement arrière

La solution de filtrage temporel implémentée pour ce type de modèle de mouvement se base sur un filtre de Haar appliqué itérativement sur des paires d'images. La figure 6 illustre les champs de mouvement utilisés pour l'application d'un tel schéma sur un GOF de taille 8 avec 3 niveaux de décomposition. Aux niveau 1 et 2, l'estimation et la compensation sont réalisées sur les sous-bandes basses fréquences temporelles du niveau précédent.

B. Estimation de mouvement avant ou arrière

Nous utilisons ici les termes d'estimation *avant ou arrière* pour exprimer le fait qu'il n'y a entre chaque couple d'images qu'un seul champ de mouvement qui est soit avant, soit arrière. Ce type

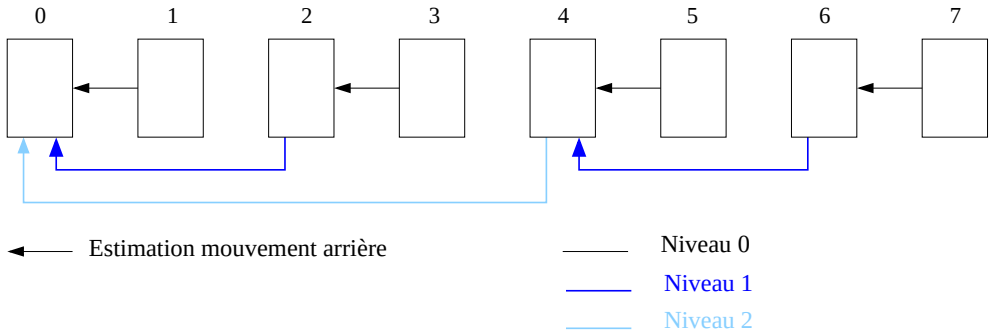

Fig. 6. Différents champs de mouvement utilisés au sein d'un GOF pour le filtre de Haar it éré sur 3 niveaux.

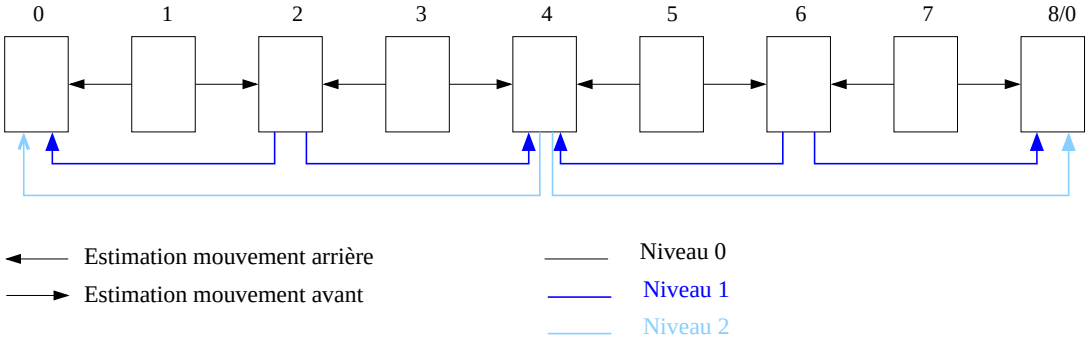

Fig. 7. Différents champs de mouvement utilisés au sein d'un GOF pour le filtre 5-3 tronqué sur 3 niveaux.

de solution est intéressant puisqu'il permet de réduire de manière significative le nombre de champs de mouvement utilisés. En fait, pour chaque image d'indice impair un champ allant vers l'image précédente et un autre allant vers la suivante sont calculés. Ceci permet de capturer des redondances provenant des deux directions temporelles. La figure illustre les champs de mouvement nécessaires à la mise en oeuvre de ce schéma sur 3 niveaux de décomposition. Le GOF utilisé ici est de taille 9 mais la dernière image sert de première image au GOF suivant, ce qui revient en pratique à coder 8 images à chaque fois. Dans ce schéma, les images extrémités sont codées en Intra. Enfin, 14 champs de mouvement sont utilisés pour une décomposition sur 3 niveaux.

C. Filtrage temporel basé sur filtres de Haar itérés

Chaque pixel de l'image de référence t et son correspondant dans l'image $t+1$, s'il existe, définissent une paire de pixels *connectés* par information de mouvement (s'il en existe plusieurs, le premier dans l'ordre lexicographique est choisi). Soit $p' = p + d$ et d le vecteur déplacement associé au pixel p, alors le filtrage de ces pixels est donné par

$$\begin{cases} L_t(p') &= \frac{1}{\sqrt{2}}(I_{t+1}(p) + I_t(p')) \\ H_{t+1}(p) &= \frac{1}{\sqrt{2}}(I_{t+1}(p) - I_t(p')). \end{cases}$$

Les autres pixels sont dits *non connectés*. Lorsqu'un tel pixel p est présent dans l'image t, une basse

fréquence temporelle est produite par

$$L_t(p) = \frac{2}{\sqrt{2}} I_t(p). \tag{1}$$

Les pixels *non connectés* de l'image $t+1$ produisent, quant à eux, une haute fréquence donnée par

$$H_{t+1}(p) = \frac{1}{\sqrt{2}} (I_{t+1}(p) - I_t(p')). \tag{2}$$

D. Autre option de structure d'analyse

Une autre option a été incorporée dans le logiciel situant la basse fréquence temporelle en milieu de groupe d'images, comme indiqué sur la figue Fig. 8. Cela permet de réduire la distance temporelle entre les images considérées dans l'analyse de mouvement et ainsi d'avoir une analyse de mouvement plus fiable.

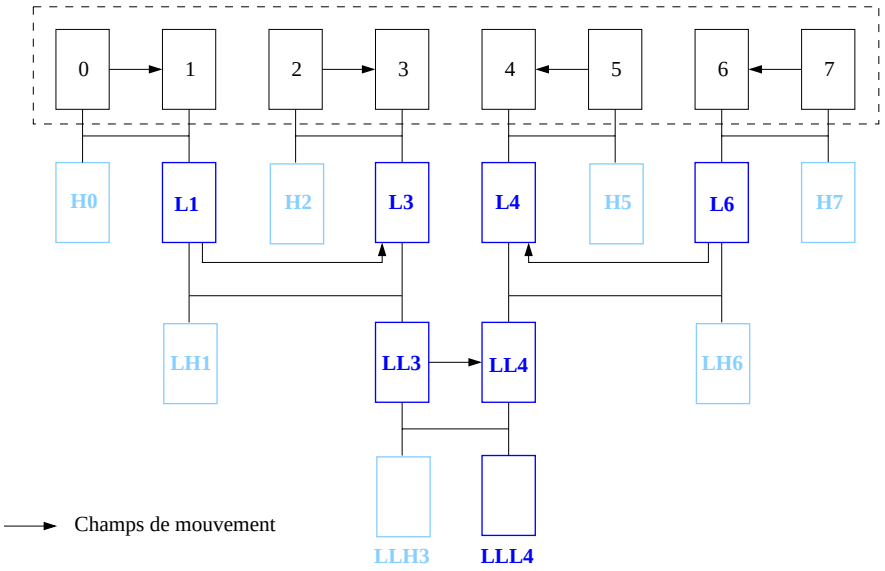

Fig. 8. Autre structure de filtrage temporel compensé en mouvement.

VII. Filtrage spatial

Nous utilisons ici une implémentation lifting d'un filtre de Daubechies 9-7 pour la transformation ondelettes spatiale. Afin d'améliorer les performances, des niveaux de décomposition spatiale différents selon les sous-bandes temporelles sont utilisés. Ainsi, sur la basse fréquence temporelle 3 niveaux de décomposition sont utilsés alors que sur les sous-bandes hautes fréquences seulement 2 niveaux sont appliqués. En effet, il n'est pas nécessaire de décomposer plus car la quantité d'information à décorréler dans les hautes fréquences temporelles est moins importante. Par conséquent, 2 niveaux de décomposition sont également utilisés pour l'ondelette spatiale appliquée à la DFD utilisée dans la prédiction inter-GOF.

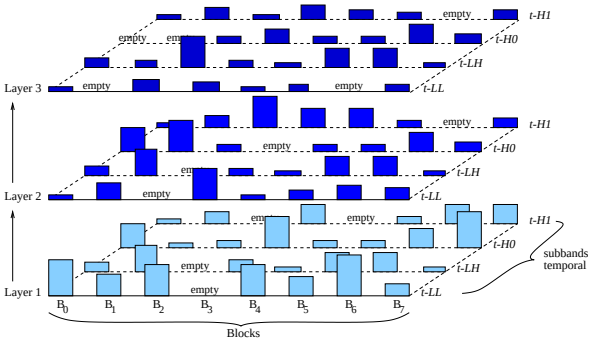

Fig. 9. Couches de qualité générées en sortie du logiciel du VM JPEG2000.

VIII. Prédiction inter-Gof

Une prédiction temporelle inter-GOF a également été rajoutée au système de codage. On distinguera alors deux types de GOF : Intra et Inter. Ce mécanisme de prédiction temporelle est réalisé en boucle fermée et necessite un champ de mouvement supplémentaire. La prédiction en boucle fermée peut-être réalisée en prenant comme information de reférence une image (sous-bande) décodée à un débit plus faible, comme dans les couches basses d'une représentation scalable classique.

Après le codage d'un GOF Intra, la basse fréquence temporelle est reconstruite à l'encodeur par une phase de décodage et de synthése temporelle. Cette sous-bande reconstruite est ensuite utilisée comme information de reférence dans la compensation de mouvement de la sous-bande basse fréquence temporelle du GOF suivant (Inter).

IX. Codage des sous-bandes

Les coefficients quantifiés des sous-bandes sont codés en utilisant l'algorithme EBCOT mis en oeuvre dans le VM JPEG-2000 mais prenant en entrée un ensemble de sous-bandes spatio-temporelles. Cela permet de générer des couches de qualité (cf figure 9). Le principe de base consiste à décomposer chacune des sous-bandes en blocs (typiquement 64x64 coefficients). Chaque bloc est ensuite compressé indépendamment à l'aide d'un codeur arithmétique contextuel. Les trains binaires obtenus sont tronqables en un nombre multiple de points. Etant donné un débit alloué à une couche donnée, le train binaire de chacun des blocs est tronqué de façon à minimiser la distorsion globale (algorithme PCRD : *Post Compression Rate-Distortion*) associée au décodage de la couche considérée. Cette optimisation, menant à la formation de couches de qualité (cf figure (9)), est particulièrement bien adaptée à une régulation fine de l'information de texture, et permet d'obtenir une scalabilité à grain fin. De plus, la flexibilité permise dans l'agencement des paquets EBCOT permet d'obtenir un train binaire hautement scalable.

X. Compilation de Wavix

Le logiciel "Wavix" a été developpé et validé sur trois environnements:

- Windows (2000 et XP),
- SunOS 5.7,
- Linux Redhat 8.0 (Kernel 2.4.18-18.8.0).

Pour chacun de ces environnements, les sources fournis permettent de reconstruire trois exécutables:

- le codeur seul, "wavix_cod",
- le décodeur seul, "wavix_dec",
- les codeur/décodeur regroupés au sein d'un même exécutable, "wavix_codec".

Pour chacun de ces exécutables, deux types de transformations temporelles sont possibles: Haar et BIME (bidirectional motion estimation). A chaque type de transformations coresspondent des makefiles spécifiques. Le codeur "Wavix" utilise le VM 5.2 de JPEG2000. Aussi, il est nécessaire de le compiler en premier lieu.

Les instructions de compilation sont précisées ci-après.

A. Wavix sous Windows

Le VM5.2 de JPEG2000 et les sources de "Wavix" se recompilent à l'aide du même workspace Visual C++ 6.0. Suivant le type de transformation temporelles mis en oeuvre, il se trouve:

- pour BIME, sous le répertoire /WavixAPP/SrcCodec/main/WAVIX_BIME/ (workspace "wavix_bime.dsw")
- pour HAAR, sous le répertoire /WavixAPP/SrcCodec/main/WAVIX_HAAR/ (workspace "wavix_haar.dsw")

B. Wavix sous SunsOS et Linux

Les reconstructions sous SunOs et Linux sont analogues.

Tout d'abord, il faut recompiler le VM 5.2 de JPEG2000 à l'aide de la commande "gmake" depuis le répertoire "/WavixAPP/JP2K/JP2KLIB_VM5.2".

Cela produit respectivement pour SunOS et Linux les librairies "libjp2kvm5_SunOS" et libjp2kvm5_linux".

Ensuite, "Wavix" se recompile à l'aide de la série de commandes "gmake clean", "gmake depend", "gmake", lancées suivant le type de transformation temporelle, depuis les répertoires:

- /WavixAPP/SrcCodec/main/WAVIX_BIME/ pour BIME,
- /WavixAPP/SrcCodec/main/WAVIX_HAAR/ pour HAAR.

"Wavix" requiert pour son exécution un fichier de configuration dans lequel sont précisés les champs suivants:

Nom du champ	**Description**
Format_echantillonnage	Format du fichier YUV, les valeurs possibles sont 444, 422, 420, 400.
Frame_rate_original	Nombre d'images par seconde de la source.
Frame_rate_target	Nombre d'images par seconde de la cible.
Frequence_Image_Intra	Fréquence de GOF intra (e.g. 2 signifie un tous les deux GOF)
Nom_fichier_entree	Nom du fichier en entrée (e.g. "foreman_150_cif.yuv", attention les quotes sont obligatoires).
Nom_fichier_sortie	Nom du fichier en sortie (e.g. "out.raw").
Nom_bitstream	Nom du fichier contenant le train binaire texture e.g "out.bits". Remarque le fichier contenant le train binaire mouvement a un nom constant "mvtbitstream".
Nombre_colonne	Nombre de colonne (e.g. 352). C'est un entier multiple de 16.
Nombre_ligne	Nombre de lignes (e.g. 288).
Nombre_decomposition_tempo	C'est une valeur entière (e.g. 3). Ne sert pas pour le haar.
Tempo_transfo	Type de la transformation temporelle Haar=0, BIME= 6.
Taille_GOF	Taille d'un GOF (e.g. 8). Pour HAAR c'est un multiple de 2.
Num_premiere_image	Numéro de la première image à encoder.
Num_derniere_image	Numéro de la dernière image à encoder.
Ratecod	Débit d'encodage. Attention c'est un float (e.g. 140000.)
Ratedec	Débit de decodage. Attention c'est un float (e.g. 140000.). Dans la version actuelle une valeur identique à celle du débit d'encodage est preconisée.
RateMvt	Pourcentage alloué au mouvement. C'est une borne max et, en pratique, dans haar, c'est moins. Ce doit être un float (e.g. 0.35).
Overlap	Significatif qu'avec du haar 0 ou 2.
Taille_de_Blocs	Taille de block maximale.

Ce fichier de configuration est passé en argument des différents exécutables "Wavix". Par exemple:

wavix_cod config.cfg

où config.cfg est le nom d'un fichier contenant les valeurs des paramètres listées ci-dessus.