

# Méthode rapide de segmentation et d'indexation du flux MPEG1-2 par bloc DCT

Lionel BRUNEL, Pierre MATHIEU

Laboratoire I3S UMR 6070 CNRS Université de Nice - Sophia Antipolis  
2000 rte des lucioles BP 121, 06903 Sophia-Antipolis Cedex FRANCE  
lbrunel@i3s.unice.fr, mathieu@i3s.unice.fr

**Résumé** – L'accessibilité des données multimédias est tributaire d'une indexation précise, ce qui demande un temps très long. Cet article propose une nouvelle méthode pour renseigner de façon automatique plusieurs champs MPEG7 (par exemple, le mouvement de la caméra et des objets). Nous exploitons au maximum les informations contenues dans le flux MPEG1-2 [1]. Afin d'accélérer le calcul, les images ne sont pas décompressées, mais nous nous limitons au décodage entropique et à la quantification inverse. Les informations de mouvement présentes dans le flux MPEG1-2 permettent d'estimer le mouvement apparent de la caméra. La segmentation des zones de couleur est obtenue grâce à un algorithme de division-fusion. Les valeurs des coefficients DCT sont aussi utilisés.

**Abstract** – The accessibility of the multimedia data depends on a precise indexing, which takes a lot of time. This paper proposes a new method to inform in an automatic way several fields MPEG7 (for example, the movement of the camera and objects). We exploit the most of the information contained in flow MPEG1-2 [1]. In order to accelerate calculation, images are not uncompressed, but we limit calculation to entropic decoding and the inverse quantization. The calculation of the camera movement is obtained from information of movement prediction in MPEG1-2 flow. The segmentation in zones of color is obtained by a "split and merge" algorithm. The values of DCT coefficients are also used.

## 1 Introduction

Avec l'augmentation de la quantité des données multimédias, MPEG7 donne une forme normalisée à l'indexation, ce qui permet d'atteindre rapidement un extrait d'une séquence. Nous proposons un algorithme d'indexation automatique avec une forte contrainte de temps, quitte à perdre en précision. Pour cela, nous utilisons le flux MPEG1-2 (format de diffusion de la télévision par satellite, par exemple) et nous exploitons au maximum l'analyse effectuée lors de la compression. La prédiction de mouvement nous permet d'estimer le mouvement de la caméra ; les coefficients de la DCT nous apportent, par exemple, des informations sur la validité de la prédiction. Nous nous limitons donc au décodage entropique et à la quantification inverse, ce qui permet de ne pas calculer la DCT inverse ni d'effectuer la reconstruction de l'image au niveau du pixel, ce qui accélère le processus.

Ce texte est divisé en trois parties. Tout d'abord, un bref rappel sur la norme MPEG1-2 (partie vidéo). La deuxième partie aborde les données que nous utilisons et qui sont tirées directement du flux, ainsi que les méthodes mises en œuvre. Pour terminer, la troisième partie montre les résultats obtenus.

## 2 MPEG1-2 (partie vidéo)

Un film peut-être considéré comme une succession d'images fixes qui peuvent être codées soit au format RVB (rouge-vert-bleu), soit, comme pour JPEG, en luminance (notée  $L$ ) et chrominances ( $Cr$  : pour la chrominance rouge et  $Cb$  : pour la bleu). Les capteurs CCD donnent en sortie une trame RVB qui peut être transformée en  $L-Cr-Cb$  par une formule non linéaire. Ce changement de représentation permet d'utiliser la

faible sensibilité de l'œil face aux chrominances, ce qui autorise un sous-échantillonnage sur les chrominances. Cette transformation permet déjà une première compression.

Dans un film, il y a une forte corrélation temporelle entre les images successives ; c'est ce que se propose d'exploiter MPEG1-2[1]. Il décompose le film en une succession de groupes d'images (GOP : *Group of pictures*), qui débute par une *Intra (I)* suivie d'une succession de *Prédites (P)* et de *Bidirectionnelles (B)* intercalées. Chaque image est découpée en macroblocs  $16 \times 16$ , qui sont composés de six blocs de  $8 \times 8$  (quatre blocs  $L$ , un bloc  $Cr$  et un bloc  $Cb$ ).

Voyons maintenant chaque type d'image plus précisément :

\* *Image de type I* : dans chaque bloc, les valeurs des pixels sont fortement corrélées, c'est pour cela que l'utilisation de la DCT, suivie d'une quantification et d'un codage entropique donne un bon taux de compression pour une image de qualité.

\* *Image de type P* : pour chacun de ses macroblocs, le codeur effectue une prédiction de mouvement à partir du  $I$  ou du  $P$  immédiatement précédent en y cherchant une zone qui lui ressemble. Cette recherche se déroule dans un voisinage spatial qui vaut généralement  $\pm 7$  pixels dans toutes les directions. Nous verrons plus bas les problèmes induits par la petitesse de la fenêtre de recherche sur la précision du calcul du mouvement de l'objet. Cette recherche, coûteuse en temps, peut être améliorée par des algorithmes plus rapides [2]. Ceci fait, nous avons donc un vecteur déplacement (appelé vecteur *Forward*).

Le codeur calcule le macrobloc d'erreur (différence pixel à pixel entre la zone identifiée comme la plus similaire dans l'image  $P$  ou le  $I$  précédente et le macrobloc original). Si cette mise en correspondance est réussie, l'image d'erreur doit avoir des pixels dont la valeur moyenne est proche de zéro avec un faible écart type. La compression du macrobloc d'erreur est

plus forte que pour un macrobloc de type *I*. Il faut donc vérifier si la reconstruction du macrobloc de *P*, i.e. le macrobloc *Prédit* avec le vecteur *Forward* additionné du macrobloc *Erreur* (qui a subi une forte compression), ne sera pas trop différente de l'image originale. Si tel est le cas, il sera codé sous forme d'un macrobloc *Intra*.

\* *Image de type B* : nous avons deux prédictions, l'une vers l'avant, à partir du *P* ou du *I* précédent (comme pour les macroblocs de type *P*) toujours appelée vecteur *Forward* et l'autre, vers l'arrière, à partir du *P* ou du *I* suivant, appelée vecteur *Backward*. Ces deux prédictions utilisent le même algorithme. Le codeur MPEG1-2 calcule pour chaque macrobloc, trois images d'erreur :

- \* celle avec le vecteur *Forward* seul
- \* celle avec le vecteur *Backward* seul
- \* celle avec les deux vecteurs (*Forward* et *Backward*).

Le codeur vérifie avec laquelle l'ajout de cette image d'erreur (qui a été compressée) à sa prédiction va donner l'image la plus proche de l'image originale.

Comme pour les *P*, il peut aussi coder le macrobloc sans prédiction. Aucun *B* n'est utilisé comme base de prédiction ; cela permet, si l'image d'erreur est "très proche" de zéro, de faire l'économie de l'envoi.

### 3 Algorithmes mis en œuvre

Les grandes lignes de l'algorithme mis en œuvre, sont tout d'abord, pour chaque image, un calcul du vecteur *Forward normalisé*. Ce dernier nous permet de calculer le mouvement apparent de la caméra. A partir de là, nous utilisons un algorithme de division-fusion. Avec les couleurs, nous séparons les zones d'intérêts. Grâce au vecteur résiduel, qui est la différence du vecteur mouvement donné dans le flux et du vecteur idéal calculé par le mouvement apparent trouvé, nous fusionnons les zones de couleurs. L'utilisation de l'image d'erreur permet d'affiner les résultats. Après l'extraction et le suivi des objets, nous calculons le mouvement des objets. Avec toutes ces informations, nous renseignons certains champs de MPEG7.

#### 3.1 Calcul du vecteur *Forward normalisé*

Le vecteur *Forward* donné dans le flux pour le *P* ou les vecteurs *Forward* et *Backward* donnés dans le flux pour le *B* représentent la longueur du déplacement avec l'image qui a servi pour la prédiction. Afin de calculer le mouvement apparent de la caméra entre deux images consécutives, il faut avoir le mouvement d'une image sur l'autre. De plus, pour les *Intras*, aucun mouvement n'est donné.

Tout d'abord, nous nous plaçons avec un GOP de la forme IBBPBBPBBPBB, puis nous supposons que le mouvement est continu sur une suite de quelques images. Nous interpolons donc le vecteur mouvement donné dans le flux pour le ramener à un mouvement avec l'image d'avant, c'est le vecteur *Forward normalisé*  $\vec{F}_n$ .

$$* \text{Cas } \underline{\text{Intra}} : \vec{F}_n = \frac{\vec{F}_s - \vec{B}_p}{2}$$

$\vec{F}_s$  : vecteur *Forward* attaché à l'image suivante

$\vec{B}_p$  : vecteur *Backward* attaché à l'image précédente

$$* \text{Cas } \underline{\text{Prédit}} : \vec{F}_n = \frac{\vec{F}}{nb+1}$$

$\vec{F}$  : vecteur mouvement donné dans le flux

$nb$  : nombre d'images qu'il y a entre ce *P* et le *P* ou le *I* précédent qui a servi à la prédiction

$$* \text{Cas } \underline{\text{Bidirectionnel}} : \vec{F}_n = \frac{\vec{F} - \vec{B}}{2}$$

$(\vec{F}, \vec{B})$  : vecteurs mouvement donnés dans le flux

$nb_F$  : nombre d'images qu'il y a entre ce *B* et le *P* ou le *I* précédent qui a servi à la prédiction

$nb_B$  : nombre d'images qu'il y a entre ce *B* et le *P* ou le *I* suivant qui a servi à la prédiction

#### 3.2 Détermination du mouvement apparent de la caméra

Avec le vecteur mouvement  $\vec{F}_n$ , nous déterminons le mouvement apparent de la caméra. Pour cela, nous supposons dans cet article que le fond est majoritaire par rapport à la surface totale des objets en mouvement.

Nous étudions le système simplifié à six inconnues suivant [6] (le système non simplifié contenait neuf inconnues) :

$$\begin{cases} u_x = -\frac{f}{Z}(T_X - xT_Z) + y.R_Z \\ u_y = -\frac{f}{Z}(T_Y - yT_Z) - x.R_Z \end{cases} \quad (1)$$

avec :

$(u_x, u_y)$  : mouvement donné dans  $\vec{F}_n$ ,

$(\hat{u}_x, \hat{u}_y)$  : mouvement estimé,

$f$  : la focale,

$Z$  : la distance de l'objet par rapport à la rétine,

$(x, y)$  : la position sur la rétine,

$T_X$  (resp.  $T_Y$  ou  $T_Z$ ) la translation selon l'axe  $X$  (resp.  $Y$  ou  $Z$ ) appelée généralement *Pan* (resp. *Tilt* ou *Zoom*),

$R_Z$  : la rotation selon l'axe  $Z$ .

Nous supposons, de plus, que nous travaillons avec une caméra à sténopé ainsi qu'avec un modèle orthographique (i.e. l'objet est dans un plan moyen) ; cela nous permet d'approximer  $\frac{f}{Z}$  par une constante. Nous nous retrouvons donc avec un système à quatre inconnues ( $T_X, T_Y, T_Z, R_Z$ ).

Pour estimer  $\hat{u}_x$  et  $\hat{u}_y$ , nous minimisons le critère des moindres carrés suivant :

$$J = \sum_{i,j} \left\{ [u_x - \hat{u}_x]_{(x_i, y_j)}^2 + [u_y - \hat{u}_y]_{(x_i, y_j)}^2 \right\} \quad (2)$$

Par le calcul des dérivées partielles par rapport aux quatre variables ( $T_X, T_Y, T_Z, R_Z$ ), nous obtenons donc un système à résoudre. Cette approche donne un résultat plus rapide que dans [3] et [4].

Pour augmenter la précision de l'estimation des vecteurs mouvement, nous supposons que sa fonction de répartition suit une loi normale centrée sur le mouvement apparent de la caméra. Nous réitérons le calcul en éliminant, à chaque fois, les vecteurs en dehors de l'intervalle de confiance fixé à 90%. Cela nous permet d'obtenir le mouvement apparent de la caméra.

### 3.3 Séparation des zones d'intérêts par leur couleur

Nous voulons découper chaque image en zones de luminance et de chrominances uniformes. Pour cela, il nous est apparu inutile de reconstruire entièrement l'image, car nous désirons obtenir de grandes zones et nous avons les valeurs moyennes par bloc pour la luminance et par macrobloc pour les chrominances. Pour les  $I$ , ces valeurs sont obtenues à une constante multiplicative près (égale à 8), par les coefficients DC des blocs DCT. Pour les  $P$  (resp.  $B$ ), il faut reconstruire la valeur moyenne de chaque bloc ou macrobloc grâce aux vecteurs mouvement, aux moyennes des blocs ou macroblochs du  $P$  ou du  $I$  précédent (resp. précédent ou suivant), et aux valeurs moyennes des blocs ou macroblochs de l'image d'erreur.

Nous utilisons, dans un seul critère, les trois composantes de couleur simultanément, ce qui nous permet d'introduire une notion de distance entre deux blocs. Pour rendre le calcul plus robuste, nous normalisons chaque différence par sa moyenne sur toute l'image.

Pour l'axe  $x$ , nous avons :

$$dist_{i,j}^x(f) = \frac{\sum_{l \in L} \left[ \lambda_l \cdot \left( \frac{f_{i,j}^l - f_{i,j+1}^l}{moy_l} \right)^2 \right]}{\underbrace{\sum_{l \in L} \lambda_l}_{L=\{lum, Cr, Cb\}}} \quad (3)$$

\*  $f_{i,j}^l$  et  $f_{i,j+1}^l$  représentent la valeur moyenne, de type  $l$ , des deux blocs voisins sur l'axe  $x$ ,

\*  $moy_l$  représente la valeur moyenne de l'image de type  $l$ ,

\*  $\lambda_l$  représente un poids fixé empiriquement (différents entre la luminance et les chrominances).

Remarque : pour la séparation de deux blocs dans le même macrobloc,  $\lambda_{Cr}$  et  $\lambda_{Cb}$  sont nuls (car les chrominances  $y$  sont constantes).

Pour chaque bloc  $(i, j)$ , si  $dist_{i,j}^x(f)$  est supérieure à un seuil fixe, déterminé empiriquement, nous supposons que les deux blocs ne font pas partie de la même zone.

Pour le calcul sur l'axe  $y$ , il suffit de remplacer  $f_{i,j+1}^l$  par  $f_{i+1,j}^l$ .

Nous avons donc séparé les zones uniformes de luminance et de chrominances. Nous appliquons un algorithme simple de fermeture de contour (cf. FIG. 1).

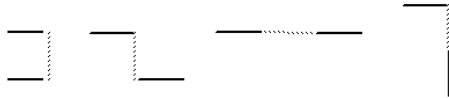


FIG. 1 – Types de contours qui va être fermés  
 traits pleins : contours détectés qui peuvent être fermés  
 pointillés : candidat à la fermeture du contour

Nous gardons les zones qui ont un contour fermé.

Afin d'agrandir, le plus possible, les zones de couleurs proches [5], nous appliquons aux zones voisines, candidates à une fusion, la distance définie dans l'équation (3).  $moy_l$  représente ici la valeur moyenne, de type  $l$ , des deux zones réunies. Si cette distance est inférieure à un seuil, nous fusionnons ces

deux zones.

Nous avons deux pas d'échantillonnage : le macrobloc pour les chrominances et le bloc pour la luminance ; une rupture peut être donc détectée au milieu d'un macrobloc (par la luminance seule) et redétectée sur le contour du macrobloc. Pour éviter cela, nous analysons sur toute l'image, les trois ruptures consécutives qui débutent au milieu d'un macrobloc. Si sur le bord du macrobloc, avec la luminance seule et un seuil plus faible, il n'y a pas de rupture, celle-ci sera considérée comme redondante et ne sera donc pas affichée. En appliquant cet algorithme, nous obtenons sur la séquence *Hall* du *COST 211* la FIG. 3a.

### 3.4 Fusion des zones de couleurs par le mouvement

Nous avons donc segmenté toutes les zones de couleurs différentes ; seulement, nous voulons extraire les objets en mouvement. Pour cela nous fusionnons les zones de couleurs différentes qui appartiennent au même objet. A cet effet, nous utilisons les vecteurs résiduels pour chaque zone segmentée. Ce vecteur est la différence du vecteur mouvement donné par le codeur MPEG et du vecteur idéal, calculé à partir du mouvement apparent de la caméra (le vecteur idéal donne le champ de vecteurs qui correspond à un mouvement global de l'image qui suit le mouvement apparent de la caméra).

Nous avons vu que le choix de l'amplitude maximale de recherche du bloc similaire dans l'image  $P$  ou  $I$  précédente (ou suivante) est totalement laissée à l'appréciation du codeur ; de plus, cette valeur maximale n'est pas transmise dans le flux. Nous sommes donc tributaires et ignorants du choix arbitraire effectué par le codeur.

La norme conseille  $\pm 7$  pixels. En effet, plus cette valeur est grande, plus la recherche devient longue. Cette valeur de recherche peut paraître faible et donc fausser totalement le calcul du mouvement apparent de la caméra. Mais il est rare qu'une caméra, dont le fond est supposé majoritaire, bouge autant. Il n'en est pas de même pour le mouvement des objets.

Ci-dessous, la représentation des vecteurs mouvements avec une même séquence encodée avec le codeur *MPEG Software Simulation Group 1994* (FIG. 2).

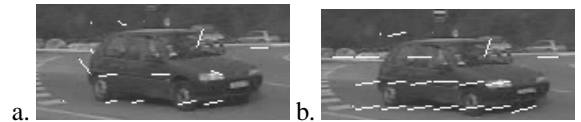


FIG. 2 – a : Recherche max.  $\pm 7$ , b : Recherche max.  $\pm 14$

La FIG. 2a représente une recherche du macrobloc identique dans un voisinage spatial de  $\pm 7$  pixels ; la FIG. 2b, dans un voisinage de  $\pm 14$  pixels. La voiture effectue une translation de 12 pixels entre deux images ; nous remarquons que dans le premier cas (2a), le résultat de la recherche du bloc le plus approché n'est pas très bonne ; par contre, dans le deuxième cas (2b), le mouvement des vecteurs résiduels correspond mieux au mouvement réel de l'objet.

L'utilisation de l'image d'erreur nous permet de vérifier la pertinence de la prédiction.

### 3.5 Utilisation de l'image d'erreur dans l'espace DCT

Une image d'erreur est composée de macroblocs codés sous forme d'un  $I$  et de macroblocs d'erreur.

Tout d'abord, l'existence d'un macrobloc codé  $I$  implique que la prédiction n'a pas été possible, soit parce que sur l'image qui sert pour la prédiction, ce macrobloc n'existait pas (occlusion), soit parce que le mouvement à prédire est supérieur à l'amplitude maximale de recherche, soit parce qu'il y a une trop grande modification de la zone (un zoom, par exemple).

Dans les autres cas, lors de l'envoi d'un macrobloc d'erreur, sur les six blocs qui le composent ( $4L, Cr, Cb$ ), tous ne sont pas envoyés. En effet, si les valeurs du bloc sont "très proches" de zéro, cet envoi n'est pas nécessaire. Par conséquent, le fait de ne rien envoyer signifie que la prédiction a été bonne. Par contre, la présence des blocs d'erreur, nous permet d'évaluer la justesse de la prédiction et donc la validité du vecteur mouvement donné dans le flux. Pour cela, nous étudions la valeur DC de la DCT, qui représente la moyenne du bloc et qui doit être généralement très proche de zéro (par valeur positive ou négative), mais aussi, le nombre de valeurs non nulles, leur amplitude et leur position dans le bloc DCT.

### 3.6 Calcul du mouvement des objets

Pour les objets, le nombre de vecteurs est trop faible, il n'est donc pas possible de calculer directement, entre deux images, la rotation  $R_Z$  ou le zoom, assimilé à  $T_Z$ . Nous nous limitons donc au calcul du déplacement moyen de l'objet par rapport au fond ( $T_X$  et  $T_Y$ ). Pour le calcul de  $T_Z$ , nous utilisons plusieurs images, en regardant la variation de la surface de l'objet dans l'image. Cela nécessite d'avoir un objet bien extrait et suivi.

### 3.7 Indexation

Nous obtenons donc, quasiment en temps réel, l'estimation du mouvement apparent de la caméra, mais aussi l'extraction des objets en mouvement dans la séquence et le calcul de leur mouvement dans la scène (mouvement réduit pour l'instant à  $T_X$  et  $T_Y$ ). Il est donc maintenant possible de faire le suivi d'un objet d'une image sur l'autre, avec :

- \* le numéro de l'image où il apparaît, et le numéro de celle de sa disparition
- \* le mouvement de sa boîte minimale rectangulaire englobante
- \* son coefficient d'occupation dans sa boîte...

Grâce à ces informations, il est possible de remplir les champs que demande MPEG7 pour chaque objet.

## 4 Résultats expérimentaux

Les images qui suivent sont extraites de la séquence *hall* donnée par le *COST 211*.

La première image (*3a*) représente les ruptures entre bloc des zones de luminance et chrominances uniformes. La deuxième image (*3b*) représente la fusion obtenue par les vecteurs mouvement de chaque zone.

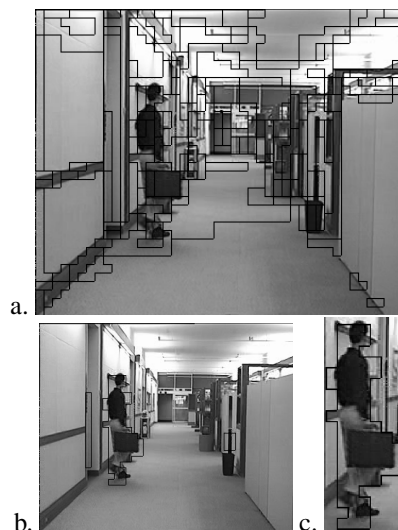


FIG. 3 – a : Segmentation par la luminance et les chrominances, b : Fusion par le mouvement, c : Zoom sur le personnage

## 5 Conclusion et perspectives

La méthode mise en œuvre évite la décompression totale du flux MPEG1-2 et utilise le maximum d'informations déjà présent dans le flux :

- \* vecteurs mouvement pour les  $B$  et les  $P$ ,
- \* coefficients de la DCT : dans les  $I$ , pour la reconstruction de l'image ; dans les  $B$  et les  $P$  pour la pertinence des vecteurs mouvement.

Le fait de rester sur la notion de bloc et macrobloc donne des résultats moins précis mais dans un laps de temps beaucoup plus court (proche du temps réel). Pour ce qui est du calcul du mouvement de la caméra, les résultats trouvés sont proches du mouvement connu des séquences et ce quel que soit le codeur utilisé (à  $\pm 1$  pixel pour les translations).

Dans l'avenir, il faudra arriver à différencier les sept paramètres (les trois translations, les trois rotations et le zoom), mais pour cela, nous ne pourrions plus considérer une image et sa suivante, mais une suite d'images.

## Références

- [1] ISO, "MPEG : generic coding of moving pictures and associated audio", Recommendation H.262, ISO/IEC, 13818-2, 1995
- [2] Han «Efficient encoding of dense motion field for motion-compensated video compression» IEEE 1999 International conference on image processing 25AP2.8
- [3] Srivasan «Qualitative estimation of camera motion parameters from video sequences» Pattern Recognition Vol. 30 pp 593-606 1997
- [4] Wang 1999 «Fast Camera Motion Analysis in MPEG domain» IEEE 1999 International conference on image processing 27PP1.3
- [5] Swee-Seong Wong «Color Segmentation and figure-ground segregation of natural images» ICIP 2000 TA04 II-120
- [6] «MPEG-7, visual part of experimentation model version 3.1» ISO / IEC JTC1 / SC29 / WG11 / M5578 décembre 1999