République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

ECOLE NATIONALE POLYTECHNIQUE

Département d'Électronique

Mémoire de fin d'études

En vue de l'obtention du diplôme d'Ingénieur d'Etat en Electronique

Thème :

Étude, simulation et implémentation sur FPGA d'un module d'estimation de mouvement pour un encodeur vidéo

Réalisé par :

Abd Ennour BOUZENAD Zine elabadine DAHMANE

Soutenu devant le jury composé de:

Pr. M.MEHENNI	**Président**
Pr. R.SADOUN	**Examinateur**
Pr. R.AKSAS	**Examinateur**
Mr. L.ABDELOUEL	**promoteur**
Mr. M.TAGHI	**co-promoteur**

Promotion : **juin 2013**

ملخص

يدخل مشروعنا النهائي في مجال تطوير تطبيق ضاغط فيديو رقمي حيث نصب اهتمامنا على الوحدة المكلفة بتقدير الحركة.

بعد دراسة نظرية حول تقنيات تقدير الحركة و محاكاة وظائف ضاغط فيديو نقوم بتحليل النتائج و عرض تجزئة برمجية/فزيائية

في النهاية قمنا بتحسين الخوارزمية المكلفة بتقدير الحركة لتركيبها في هندسة مختلطة في دارة FPGA /CPU

كلمات مفتاحية: تقدير الحركة،معيار MPEG ، ضاغط فيديو

Résumé

Notre projet de fin d'étude s'inscrit dans le cadre de développement d'une application de compression vidéo numérique, nous nous intéressons tout particulièrement au module chargé de l'estimation de mouvement.

Après une étude théorique des techniques d'estimation de mouvement et une simulation des fonctionnalités d'un encodeur vidéo de la norme MPEG, les résultats sont analysés et un partitionnement logiciel/matériel est proposé.

Enfin une amélioration a été apportée à l'algorithme d'estimation de mouvement en vue d'une implémentation simple autour d'une architecture mixte FPGA/CPU.

Mots clé : Estimation de mouvement, norme MPEG, compression vidéo.

Abstract

Our final project enrolls in the video compression domain; we have bestowed particular interest upon the module tasked with motion estimation.

Following a detailed theoretical study of motion estimation techniques, a simulation of the features of an MPEG encoder has been performed. We have analyzed the results and suggested a software/hardware spread of computation.

Finally, we have proposed an improvement of the motion estimation algorithm in the perspective of a mixed FPGA/CPU implementation.

Keywords: Motion estimation, MPEG norm, video compression.

Projet de Fin d'Études réalisé au Laboratoire des

Dispositifs de Communications et de Conversions Photovoltaïques

École Nationale Polytechnique

10 avenue Hacène BADI – El Harrach

BP182–16200 Alger

Algérie

Tél : (+213) 21 52 53 01/03

Fax : (+213) 21 52 29 73

Web:http: //www.enp.edu.dz/

Remerciements

Au terme de ce projet de fin d'études réalisé à l'Ecole Nationale Polytechnique d'Alger, nous remercions les membres du jury qui ont accepté de juger ce travail. Nous tenons à exprimer notre profonde gratitude à notre encadreur Mr L.ABDELOUEL qui, grâce à sa disponibilité, aide et rigoureux conseils, nous avons pu mener à bien ce projet.

Nos remerciements s'adressent également à Mr M.TAGHI pour son aide et disponibilité.

Nous remercions également nos familles qui nous ont soutenus tout au long de nos cursus, nos amis qui nous ont encouragés et à tous ceux qui nous ont aidés, de près ou de loin, à accomplir ce projet.

Table des matières

Table des figures

Liste des tableaux

Introduction générale

I. Introduction

Avec l'introduction de la télévision numérique haute définition (TVHD), et le développement des communications via Internet, la vidéonumérique est actuellement en plein essor. Les anciennes normes PAL, SECAM et NTSC laissent peu à peu leur place à des normes de codage numérique.

La compression des données joue ici un rôle primordial, que ce soit dans le stockage ou dans la diffusion des vidéos, puisqu'elle doit permettre destocker plus de données sur moins d'espace, de diffuser plus de chaines numériques sur une même bande passante disponible en minimisant les données à transmettre.

II. Problématique

Le transport d'un flux vidéo numérique nécessite de compresser celui-ci afin de l'adapter au débit du canal de transmission ou à la capacité du support. La compression des données est réalisée en supprimant la redondance et les composants qui ne sont pas nécessaires pour une reproduction fidèle des images. La plupart des méthodes de codage vidéo exploite les redondances temporelles et spatiales pour obtenir la compression.

Dans ce PFE nous nous intéressons au problème de l'estimation du mouvement pour un encodeur vidéo MPEG4. Nous nous intéressons tout particulièrement à l'implémentation d'une solution matériel/logiciel du module de calcul du SAD.

III. Objectifs du projet

Ce travail s'inscrit dans le cadre des activités du Laboratoire des Dispositifs de Communications et de Conversions Photovoltaïques (LDCCP) de l'ENP. Il vise à explorer les possibilités qu'offrent les circuits programmables (FPGA) pour l'accélération des calculs numériques.

Plus spécifiquement, les objectifs suivants sont poursuivis pour ce projet de fin d'étude :

- Étude des techniques de compression vidéo,
- Étude des algorithmes d'estimation de mouvement
- Profilage et partitionnement logiciel/matériel d'un système d'estimation de mouvement.
- Implémentation du module d'estimation de mouvement autour d'une architecture mixte CPU/FPGA.
- Implémentation et gestion de la communication FPGA/CPU

IV. Organisation du mémoire

Ce mémoire contient quatre chapitres.

- Dans le premier, nous passons en revue un état de l'art sur les techniques de compression ou on décrit les principes de base des normes de codage vidéo.
- Dans le chapitre II, nous étudions la norme MPEG4. L'étude est suivie par une simulation des spécifications fonctionnelles et un profilage des différents modules de la norme.
- Dans le chapitre III, on explore les différentes techniques d'estimation de mouvement, avec une attention particulière aux algorithmes de mise en correspondance de blocs (Block Matching), qui constituent de bons candidats à l'implémentation matérielle.
- Dans le chapitre 4, nous présentons une nouvelle méthode d'estimation du mouvement. Une architecture est proposée et son implémentation sur la plateforme de développement Celoxica RC203E sera décrite.
- Enfin, en conclusion, nous rappelons les objectifs atteints et les perspectives …

Chapitre 1 État de l'artdes techniques de compression vidéo

Introduction

La compression vidéo se compose d'une paire complémentaire des systèmes, un compresseur (encodeur) et un décompresseur (décodeur).

Dans ce chapitre, nous exposerons dans une première partie, les principes de base de la compression avec un survol des normes de compression vidéo définies à ce jour, de H.261 à MPEG4/H.264.Dans une deuxième partie, les spécificités du standard MPEG4 qui promet de fournir une meilleur compression vidéo, seront présentées.

1.1 Les principes de base de la compression vidéo

Une vidéo contient une quantité importante de données, et malgré l'augmentation de la puissance des processeurs et des capacités des périphériques de stockage, nous avons besoin de la représenter dans un format plus concis (figure 1.1). C'est la raison de la naissance du terme « compression vidéo ».

Fig. 1.1 Schéma bloc d'une chaine de compression numérique [1].

La plupart des applications vidéo numériques s'appuient sur l'affichage de la vidéo couleur et donc qu'il existe un mécanisme pour capturer et représenter les informations de couleur. Une vidéo numérique non compressée se caractérise par son système de représentation des pixels, sa taille et son rafraîchissement.

Une image monochrome exige une seule valeur pour indiquer la luminosité ou de luminance de chaque échantillon spatiale. D'autre part, les images en couleur, exigent au moins trois nombre pour représenter la couleur de chaque pixel. La méthode choisie pour représenter la luminosité (luminance ou luma) et la couleur est décrite comme un espace colorimétrique.

1.1.1 Espace colorimétrique

De nombreux espaces colorimétriques existent. On peut citer parmi eux le RVB et le YUV. En RVB (rouge, vert, bleu), un pixel est présenté par trois nombres qui indiquent la proportion relative de rouge, de vert et de bleu. Le RVB est très utilisé dans la capture et l'affichage grâce à la facilité de séparation de la lumière en ces trois composantes. La vision humaine est moins sensible aux couleurs qu'à la luminosité. En RVB, les trois couleurs possèdent la même importance, si bien qu'il faut garder la même résolution pour chaque composante. Il existe des espaces qui séparent la luminosité des couleurs. Il est possible d'augmenter la résolution de la luminance et de réduire celle de la chromatique, pour améliorer la qualité du signal décodé [2]. Le YUV et ses variations (YCrCb) représentent un pixel de couleur par une luminance Y et trois composantes chromatiques Cb, Cr et Cg. Heureusement, les composantes sont interdépendantes, si bien qu'avec trois d'entre elles, nous obtenons la quatrième. En général, seul Y, Cb et Cr sont transmis. Il y a plusieurs formats différents d'échantillonnage en YUV. Les plus communs sont le 4 :2 :0 (YV12), le 4 :2 :2 (YUY2) et le 4 :4 :4 (RVB).

1.1.2 Formats

Le standard de compression vidéo peut comprimer une grande variété de formats colorimétriques sous diverses résolutions (nombre de lignes x nombre de colonnes). En pratique, le format CIF est la base d'une hiérarchie de formats. Les pixels peuvent être représentés de plusieurs manières comme le montre la figure 1.2.

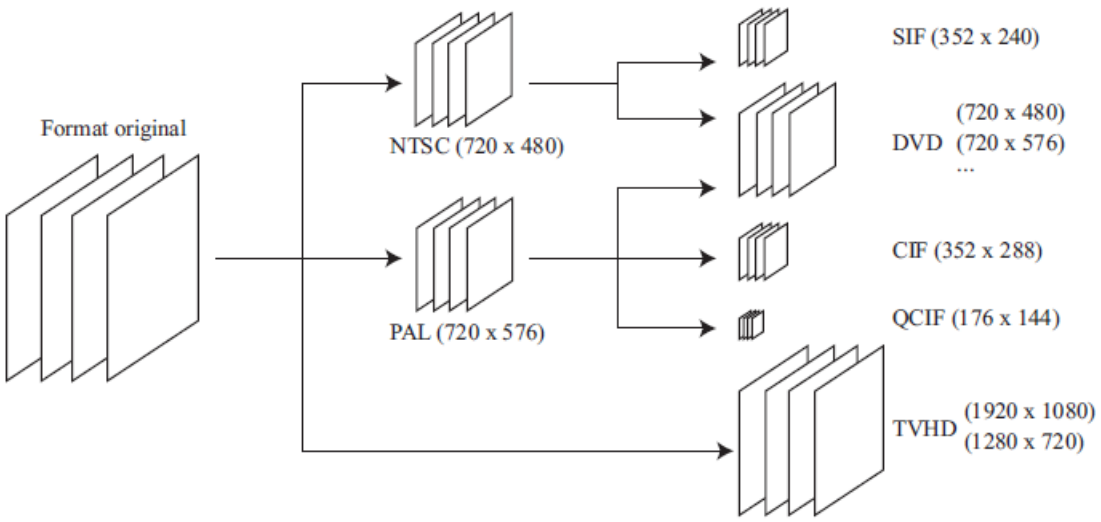


Fig. 1.2 : Différents formats utilises en vidéo numérique [3]

Dans une image à niveaux de gris, un pixel possède la plupart du temps 256 niveaux de gris (codés sur 8 bits). Le niveau 0 code le noir et le niveau 255 le blanc. Une image couleur est représentée avec trois composantes. Le tableau 1.1 récapitule les paramètres des formats standards. La taille nécessaire à la mémorisation de l'image et Le débit nécessaire à la transmission des données non compressées sont calculés pour un format en 4 :2 :0 avec 8 bits par composante.

FORMAT	**Résolution Luminance (Horizontal X Vertical)**	**Bits par image (4 :2 :0,8 bits)**	**Application**	**Fréquence**	**Débit**
Sub_QCIF	128 x 96	147456	Vidéo mobile	10 à 30 Hz	
Quarter_CIF	177 x 144	304128	Vidéo conférence	10 à 30 Hz	380 Ko 1.1 Mo/s
CIF	352 x 288	1216512	Vidéo surveillance	60 Hz	1.5 à 4.6 Mo/s
4CIF	704 x 576	4866048	TV, DVD	50 Hz	
720i50	1280 x 720	11059200	HD TV	24 à 60 Hz	33.2 à 83 Mo/s
1080i50	1920 x 1080	24883200	HD DVD	50 à 60 Hz	78 à 93 Mo/s

TAB. 1.1 : Formats standard [3]

Comparées aux débits actuels des communications numériques (de quelques centaines de Kbits/s pour le téléphone portable (3G) à quelques dizaines de Mbits/s pour une chaîne satellite), les images non compressées représentent d'énormes quantités de données.

Par exemple résolution 1920×1080 pixels de la télévision numérique haute définition (TVHD1080i) atteint des débits non compressésd'au moins 1,5 Gbits/pour le futur, on parle déjà de l'UHDTV (ultra-high definition TV). Les vidéos produites pour ce format auront une résolution maximale de 7680 × 4320[4], soit 32 millions de pixels, avec une fréquence de 60 images par seconde. Une minute d'UHDTV non compressée demanderait environ 195 Go d'espace de stockage (soit plus de 40 DVD).

Cette courte présentation illustre la nécessité du développement de codeurs vidéo efficaces, permettant la retransmission de contenus à haut débit sur des supports dont la capacité évolue moins vite (figure 1.3).

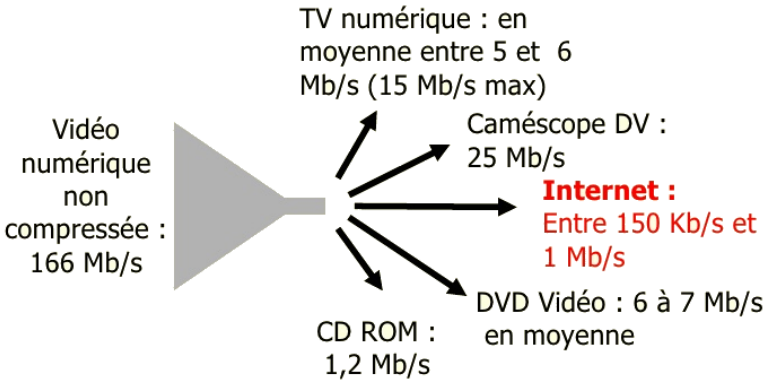


Fig. 1.3 : Domaines d'application de compression des vidéos numériques [5].

1.1.3 Architecture générique de compression vidéo

Tous les systèmes de compression vidéo actuellement utilisés fonctionnent selon le même principe et possèdent les mêmes étapes décisives. La compression vidéo consiste à réduire leurs tailles en supprimant la redondance ainsi que l'information imperceptible par l'œil humain. Elle exploite les redondances temporelles et spatiales pour extraire le minimum de données nécessaires pour la reproduction des images originales.

Une vue globale d'un système générique de compression vidéo comprend un encodeur et un décodeur comme l'illustre la figure 1.4.

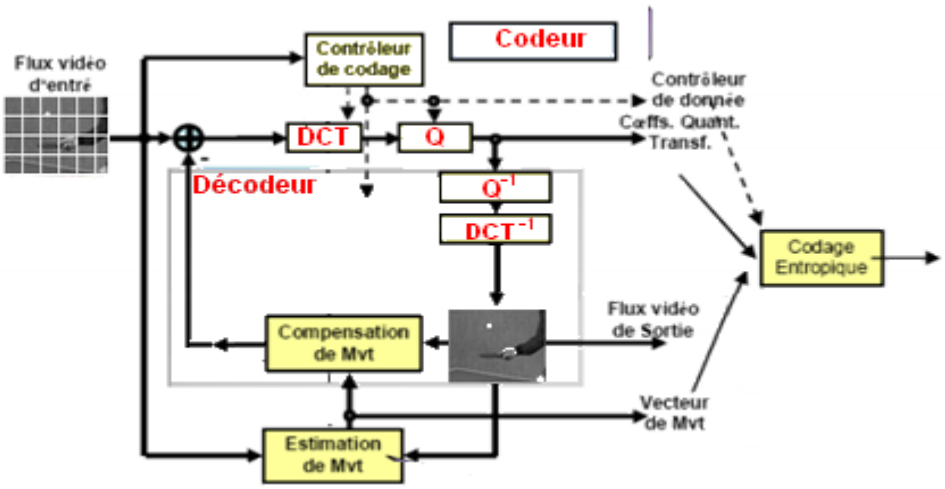


Fig.1.4 - Schéma global d'un encodeur vidéo [5]

L'encodeur est constitué de trois modules : la compression spatiale, la compression temporelle (ou prédiction) et le module de codage entropique. A l'entrée de l'encodeur, nous fournissons une séquence vidéo paramétrée.

La compression spatiale consiste à éliminer la redondance entre pixels d'une même image et ce en exploitant la corrélation que chacun d'eux présente avec son voisinage. Une transformation convertit une trame en coefficients, qui sont ensuite quantifiés pour obtenir une série de coefficients significatifs représentant la trame de manière plus compacte.

Vient ensuite le module compression temporelle (ou de prédiction) qui contrairement au module de compression spatiale ce module exploite la redondance entre les trames voisines et construit une prédiction des trames suivantes à partir de trames dites de référence.

Ces compressions convertissent une trame en coefficients. Ces coefficients subissent à leur tour une compression statistique par un codage entropique.

Le décodeur quant à lui reconstruit la vidéo à partir de la séquence de sortie de l'encodeur. Les coefficients et les vecteurs de mouvement sont décodés par un décodeur entropique et puis, le décodeur reconstruit une version de la trame résiduelle. Ensuite, à partir d'une trame précédemment reconstruite, des vecteurs de mouvement et de la trame résiduelle, le décodeur obtient la trame de départ.

1.1.4 Evaluation de la qualité de l'image

La compression vidéo réduit de façon plus ou moins la qualité des trames vidéo compressées. L'évaluation de leurs qualités est très complexe. Il existe des métriques de qualité objectives, basées sur un calcul de distorsions comme le PSNR (Peak Signal to Noise Ratio) très utilisé dans le traitement du signal en général [6]:

$$PSNR = 10\ Log_{10}\left(\frac{(Amplitude\ Max)^2}{\frac{1}{N}\ X\ \sum_0^{N-1}(Originale - Reconstruite)^2}\right) \quad \textbf{(1.1)}$$

Le PSNR mesure une erreur quadratique moyenne non pondérée sur toute l'image. Une alternative est proposée par la SSIM (Structural SIMilarity) [7] qui pondère les défauts en fonction de leur voisinage. En effet l'œil humain perçoit moins les défauts dans les zones texturées que dans les zones homogènes. D'une manière générale, ces métriques sont très limitées pour évaluer la façon dont l'œil humain perçoit les défauts dans une image et la façon dont le cerveau les interprète.

1.2 Modules d'un système de compression vidéo

Dans une séquence vidéo, deux images successives contiennent très probablement un grand ensemble d'informations semblables. En définissant le résidu comme la différence entre deux trames consécutives, nous trouvons que les parties statiques interviennent peu dans le codage du résidu.

Le processus de compression permettant de passer d'une vidéo non-compressé a une vidéo compressée est constitué de quatre modules (figure 1.5).

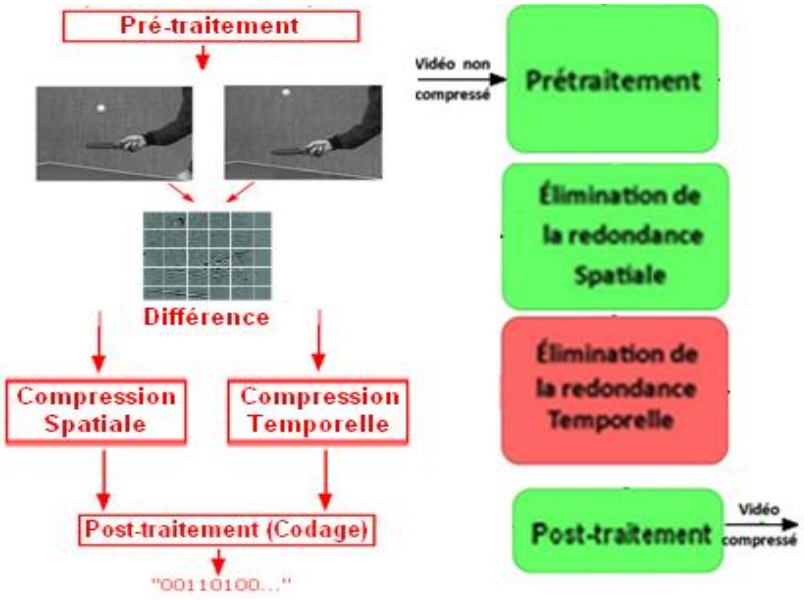


Fig. 1.5 - Processus et modules d'un système de compression vidéo.

1.2.1 Module de prétraitement

Avant de commencer le processus de compression toutes les trames d'une séquence vidéo passent par un prétraitement obligatoire pour les adapter aux étapes aux modules de compression.

La conversion de couleurs : calcul les composante YCrCb à partir des composantes RVB, en utilisant les formules suivantes [8]:

$$\begin{pmatrix} Y \\ Cr \\ Cb \end{pmatrix} = \begin{pmatrix} 0.257 & 0.504 & 0.098 \\ 0.439 & 0.368 & 0.071 \\ -0.148 & 0.291 & 0.439 \end{pmatrix} \times \begin{pmatrix} R \\ V \\ B \end{pmatrix} + \begin{pmatrix} 16 \\ 128 \\ 128 \end{pmatrix} \quad \textbf{(1.2)}$$

Pour reconstruire l'image originale il suffit d'utiliser le système d'équations inverses [8] :

$$\begin{pmatrix} R \\ V \\ B \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1.140 \\ 1 & -0.395 & -0.580 \\ 1 & 2.032 & 0 \end{pmatrix} \times \begin{pmatrix} Y \\ Cr \\ Cb \end{pmatrix} + \begin{pmatrix} 464.2541 \\ -335.3364 \\ 670.3702 \end{pmatrix} \quad \textbf{(1.3)}$$

Etant donné que le Système Visuel Humain (SVH) est plus sensible à la luminance qu'aux chrominances, une opération de sous échantillonnage de l'information de chrominance Cr et Cb est effectuée par tous les encodeurs vidéo. Le principe est d'échantillonner le signal de luminance à un certain taux, et les signaux de chrominance à d'autres taux inférieurs ou égaux. La figure 1.6, illustre l'échantillonnage réalisé avec différents formats (4:2:0, 4:2:2, et 4:4:4)

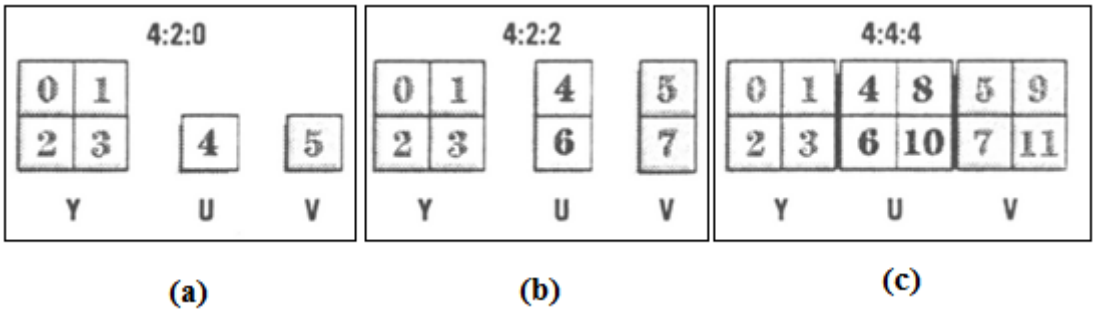


Fig. 1.6 : Format de la vidéo (a) 4:2:0, (b) 4:2:2, (c) 4:4:4[9]

Cette réduction du volume d'information permet de limiter la quantité de données à compresser puis à transmettre. En effet, la complexité calculatoire du processus de compression pour le format 4:2:0, est réduite de 50% [9].

1.2.2 Module de compression spatiale

La compression spatiale utilise la similarité entre des pixels adjacents sur une surface unie et tient compte des fréquences spatiales dominantes existant dans les zones d'une image. En codage spatial, la première étape consiste à effectuer une analyse de fréquence spatiale à l'aide de transformée cosinus discrète.

1.2.2.1 Transforme DCT :

En MPEG, la Transformée Cosinus Discrète (DCT) est utilisée sous sa sous forme bidimensionnelle.

La définition formelle de la DCT à deux dimensions est la suivante [10]:

$$f(u,v) = \frac{2}{N} \times C(u) \times C(v) \times \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} I(x,y) \cos\left(\frac{(2x+1)u\pi}{2N}\right) \cos\left(\frac{(2y+1)v\pi}{2N}\right) \quad \textbf{(1.4)}$$

$$u, v, x, y = 0, 1, 2, 3, \ldots\ldots\ldots N-1$$

$$avec\ C(u), C(v) = \begin{cases} \frac{1}{\sqrt{2}} & pour\ u, v = 0 \\ 1 & sinon \end{cases}$$

I : étant une image carrée (N × N) pixels.

f : la matrice des fréquences de (N × N) pixels.

Où x et y sont les coordonnées spatiales et u et v sont les coordonnées dans la transformée

La valeur d'une fréquence reflète l'importance et la rapidité d'un changement, tandis que la valeur d'une amplitude correspond à l'écart associé à chaque changement de couleur.

La combinaison des (N × N) coefficients de la DCT permettra de reconstituer le bloc initial de (N × N) pixels. La transformée inverse est donné par la formule [10]:

$$I(x,y) = \frac{2}{N} \times \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u) \times C(v) \times f(u,v) \cos\left(\frac{(2x+1)u\pi}{2N}\right) \cos\left(\frac{(2y+1)v\pi}{2N}\right) \quad \textbf{(1.5)}$$

$$u, v, x, y = 0, 1, 2, 3, \ldots\ldots\ldots N-1$$

Ainsi un bloc de (N × N) pixels est changé en bloc de (N × N) coefficients. De ce fait, une DCT ne constitue pas une compression, mais elle convertit la source de pixels en une forme facilitant la compression. En effet, la majorité des coefficients a une valeur nulle ou proche de zéro. Il ne sera donc point nécessaire de les transmettre. En plus, les changements rapides dans les hautes fréquences peuvent être imperceptibles, c'est pourquoi on peut les

éliminer [11]. Il en résulte une compression non négligeable sans perte véritablement conséquente.

1.2.2.2 Quantification :

La quantification n'est que le processus de réduction du nombre de bits nécessaires au stockage d'une valeur entière par la diminution de la précision de la qualité.

La quantification vient juste après la transformée en cosinus discrète DCT, C'est l'étape de l'algorithme chargé de l'élimination de la redondance spatiale de l'image.

La méthode classique consiste diviser les éléments de la matrice résultante après la DCT par un même nombre Ceci permet de "simplifier" l'information contenue, et donc de rendre la compression plus facile. En effet, on ramène l'ensemble des valeurs à un ensemble plus petit, ce qui le rendra plus aisément compressible.

Une quantification plus judicieuse est de diviser les éléments de la matrice par différents coefficients dans le but d'atténuer voir annuler les hautes fréquences dont les amplitudes sont déjà faibles (non perceptible par l'œil humaine).

Le calcul effectué par le module de quantification est le suivant [12]:

$$F(i,j) = \left\lfloor \frac{f(i,j)}{Q(i,j)} \right\rfloor, 1 \leq i,j \leq N \qquad \textbf{(1.6)}$$

Avec $\lfloor x \rfloor = entierdirectementinférieur$ à x

Les matrices de quantification utilisées sont standardisées de façon à avoir une perte d'information tolérable, pour reconstruire la matrice des fréquences le décodeur doit obligatoirement utiliser la même matrice de quantification que son encodeur pour l'opération inverse, nous obtenons donc la matrice des fréquences décompressées à l'aide de la formule suivantes [12]:

$$f^*(i,j) = F^*(i,j) \times Q(i,j) \ ,1 \leq i,j \leq N \qquad \textbf{(1.7)}$$

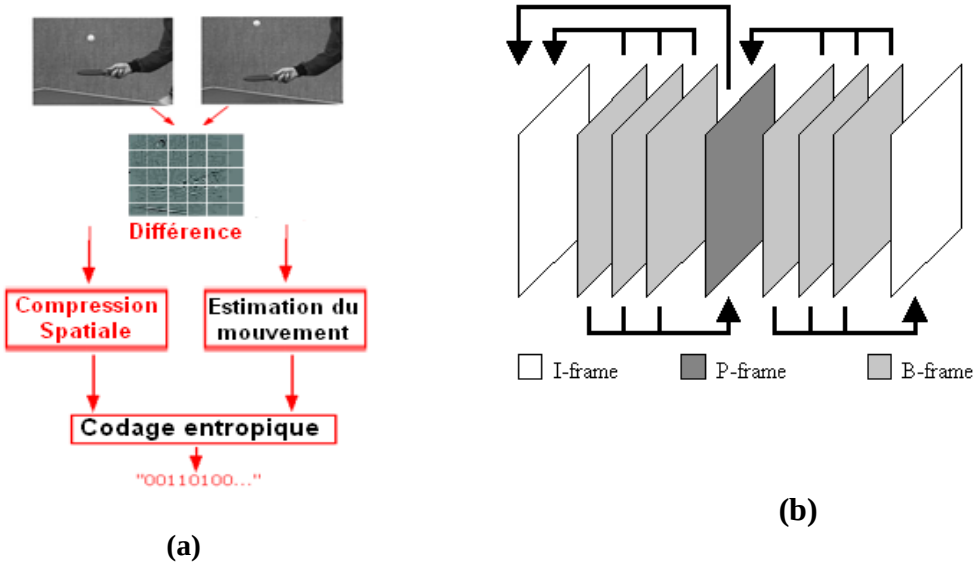
1.2.3 Module de compression temporelle (prédiction)

Dans une séquence vidéo, la différence entre une image et la suivante est relativement faible, sauf lors d'un changement de plan. La compression temporelle exploite la corrélation temporelle entre des images successives et s'effectue par GOP (Group of Pictures). Le codage

MPEG prévoit de numériser les images d'une séquence selon trois catégories : 1es image clés (I), les images prédictives (P) et les images bidirectionnelles (B) :

1. Les images extrêmes (I frame) de la séquence appelées images "intra" sont codées de façon autonome (compression spatiale uniquement) et elles sont utilisées comme référence pour reconstruire les images de type P.
2. Les images intermédiaires, appelées images "prédites" sont codées en tenant compte de leur différence avec l'image intra ou prédite qui les précède.
3. Les autres images, appelées images "bidirectionnelles" sont codées en tenant compte des images intra ou prédites qui les encadrent.

Les figures 1.7aet 1.7b illustrent respectivement le principe du codage et les catégories d'images d'une séquence vidéo selon la norme MPEG.



(a) **(b)**

Fig. 1.7 : le principe du codage MPEG et les catégories d'images d'une séquence vidéo.

1.2.4 Module de post-traitement (codage entropique)

L'étape finale avant de transmettre les données sur un canal est celle du codage entropique des données en flux de bits organisé sous forme de bitstream.

Le codage entropique (ou codage statistique à longueur variable) est une méthode de codage de source sans pertes, dont le but est de transformer la représentation d'une source de données pour sa compression et/ou sa transmission sur un canal de communication. Les principaux types de codage entropique sont le codage de Huffman et le codage arithmétique.

Le codage des coefficients quantifiés se fait comme suit :

- Le codage des coefficients DC exploite la corrélation spatiale des blocs adjacents. Il consiste à calculer les différences $DC_i - DC_{i-1}$(figure 1.8) et de les codifier. Cette méthode est appelée la DPCM (Differential Pulse Code Modulation).

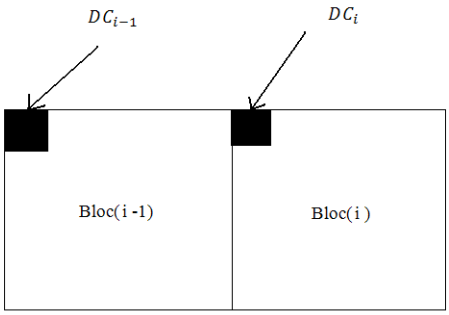


Fig. 1.8 : Coefficients DC de deux macro-blocs.

- Avant le codage des coefficients AC (hautes fréquences) quantifiés, on commence par l'ordonnancement de ces coefficients en fonction de leurs énergies décroissantes par rapport à l'origine. Pour cela un algorithme de scan dit ZigZag est appliqué (figure 1.9).

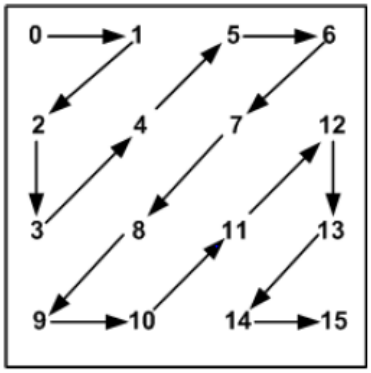


Fig. 1.9 : Scan Zigzag [13]

Pour obtenir une séquence de coefficients quantifiés qui sera transcodée en séquence binaire prête à être transmise La procédure de codage est présentée en [14]

1.3 Les standards de compression vidéo :

La propagation qu'a connu la vidéo numérique ces deux dernières décennies a été accompagné par le développement de plusieurs normes de compression vidéo, cependant le

standard international de compression proposé par le groupe de travail MPEG (Moving Picture Experts Group) publiéen novembre 1992 sous le nom de MPEG-1 a été le premier à utiliser la prédiction d'images, depuis plusieurs standards MPEG ont été développés Tableau. 1.2

Standard	Application
MPEG-1	Vidéo-CD, CDI
MPEG-2	Télévision numérique, Satellite, TNT TV Haute Définition, DVD
MPEG-4 Visual	Internet, vidéo mobile, VOD (Vidéo On Demand), Studio
MPEG-4 AVC H.264	Internet, vidéo mobile, Haute Définition VOD (Video On Demand), Studio
MPEG-7	Même encodage audio et vidéo que MPEG4 en plus découpe sémantique de l'image en objets« manipulables »
H.263	Transmission des vidéos sur des réseaux à très bas débit. Visioconférences, vidéos sur les téléphones portables de 3ème génération
H.264 AVC (Advanced Video Coding)	Codec utilisable sur beaucoup de réseaux et de systèmes différents (télévision, téléphonie, etc…).

TAB. 1.2 : Les standards de compression vidéo [3]

Un autre organisme s'intéresse au développement d'encodeur vidéo nommé ITU-Tet principalement aux applications nécessitant de faibles débits, applications de visiophonie et de visioconférence (figure 1.10).

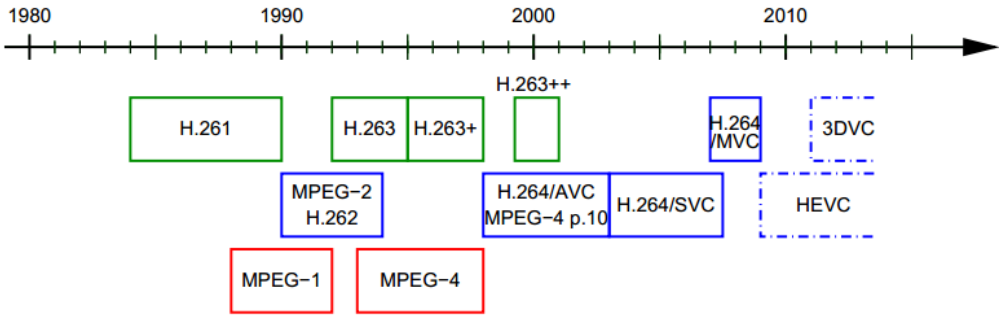


Fig. 1.10 – évolution des encodeurs vidéo [15].

1.4 Description du standard MPEG4/AVC – H.264

Le standard vidéo le plus attractif aujourd'hui est le MPEG-4 AVC ou H.264. Malgré une complexité très supérieure aux précédents standards (MPEG-2, MPEG-4 Visual), il est particulièrement apprécié pour ses performances en terme de compression.

Cette section décrit brièvement les caractéristiques de H.264 et présente plus précisément ses spécificités en matière d'estimation de mouvement.

1.4.1 Architecture MPEG4/AVC – H.264

La figure 1.11 donne une description générale de haut niveau d'un encodeur H.264. Les opérations classiques des codeurs vidéo sont reprises. Une vue d'ensemble du codeur peut être retrouvée dans [16].

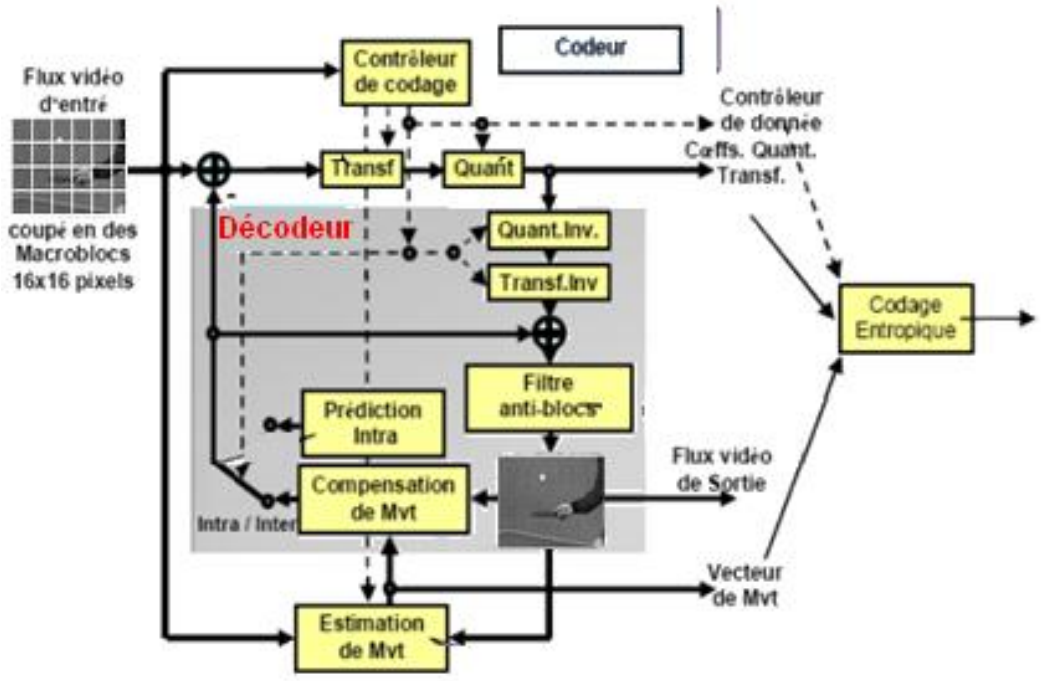


Fig. 1.11 : Diagramme en bloc du codeur H.264 [5].

Cette chaîne de compression vidéo est en grande partie similaire aux autres standards MPEG. Toutefois, les caractéristiques principales de la chaîne de compression H.264/AVC sont les suivantes :

- Simplification de la DCT grâce à l'utilisation d'une transformée entière. Cela reporte en effet la partie décimale des coefficients transformés à l'étape de quantification. Cette approche réduit la complexité calculatoire de la DCT vis-à-vis de celle utilisée dans les normes précédentes.
- Quantification : le pas de quantification est variable afin de pouvoir accéder à l'image entière, ce qui n'était pas possible avec les normes antérieures qui laissait des zones inaccessibles pour certains quantificateurs.

❖ Taille des bocs variable : L'estimation de mouvement peut être faite sur des blocs de taille variable. Les macro-blocs 16×16 peuvent être divisés en 16×8, 8×16 ou 8×8 et les sous-partitions 8×8 peut être à nouveau divisées en 8×4, 4×8 ou 4×4. Le choix de petits blocs améliore la précision mais nécessite la transmission d'un plus grand nombre de vecteurs Un algorithme de décision efficace doit donc être mis en place. Il peut faire partie intégrante de l'estimateur ou bien être séparément exécutée. Dans ce dernier cas l'estimateur de mouvement fourni les vecteurs pour toutes les tailles de blocs [17].

1.4.2 Les profils de la norme MPEG

Le MPEG peut s'utiliser dans diverses applications nécessitant des performances et des complexités diverses. A l'aide des outils de codage définis dans le MPEG, il existe des milliers de combinaisons possibles. Dans un but de simplification, le MPEG est divisé en profils, chaque profil étant lui-même subdivisé en niveaux. Un profil définit l'ensemble des fonctions que l'encodeur peut utiliser et limite la complexité d'implémentation. Un niveau est en fait un paramètre définissant par exemple la taille de l'image ou le débit du flux de bits.

Le MPEG possède plusieurs profils, qui ciblent chacun une catégorie spécifique d'applications. . La figure 1.12 montre le rapport entre les quatre profils H.264 normalisés.

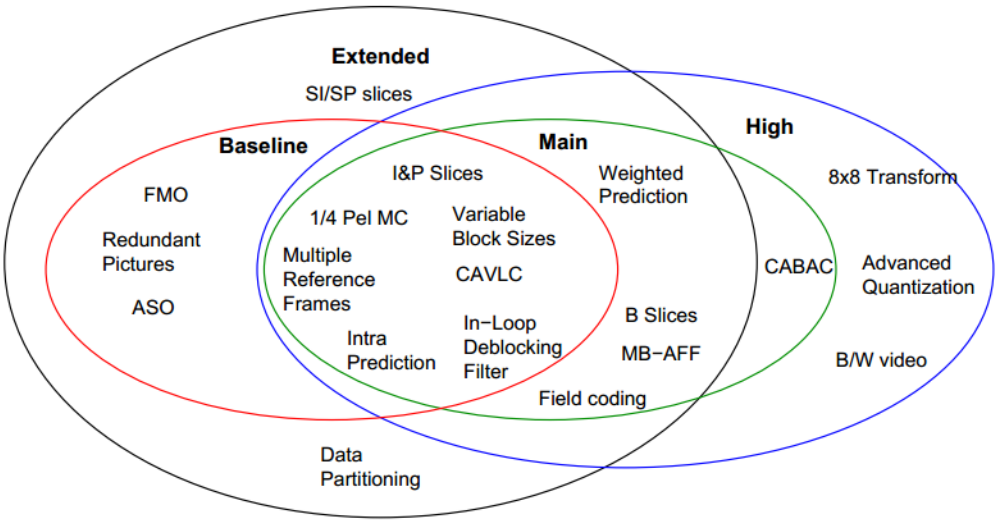


Fig. 1.12 : Les profils du H.264 [9]

1. Le profil de base « Baseline » est le plus simple des profils Il définit, par exemple, le balayage en zigzag de l'image et l'utilisation du sous-échantillonnage de chrominance 4:2:0. Avec ce profil, l'image est divisée en blocs de 4 x 4 pixels et

chaque bloc est traité séparément. Le profil Baseline utilise un codeur entropique à longueur variable : le CAVLC [18] (Context-Adaptive Variable-Length Coding) qui fait partie des techniques de compression sans perte. Les applications potentielles de ce profil comprennent la visiophonie, la visioconférence et les communications sans fil.

2. Le profil principal. « Main » et le profil Extended ont les mêmes fonctionnalités que Baseline à ceci près qu'ils disposent d'algorithmes de prédiction améliorés. La prédiction temporelle et la prédiction du mouvement sont très utilisées (codage-inter avec des B-Frames).
    a. Le profil « Extended »permet d'ajouter des informations (signal complémentaire) pour améliorer un certain aspect de la qualité. Par exemple, un codeur MPEG conventionnel peut, en re-quantifiant fortement les coefficients, coder une image avec un rapport signal/bruit modéré. Si cette image est localement décodée et soustraite pixel par pixel de l'image originale, le résultat constituera l'image de bruit de quantification. Cette image peut être compressée et transmise en tant que signal complémentaire.
    b. Le profil principal. « Main » inclut le support pour la vidéo entrelacée, et utilise le codage arithmétique CABAC (Context-Adaptive Binary Arithmetic Coding) comme codeur entropique [19]. Ce dernier fournit de meilleurs résultats de compression (un gain d'environ 10 % par rapport à CAVLC dans des conditions similaires). Les applications potentielles du profil incluent la télévision de radiodiffusion et le stockage vidéo.
3. Le profil High est le profil H.264 le plus puissant, c'est celui qui permet le codage le plus efficace de la vidéo. En plus du codage avec la méthode CABAC, on trouve le recours à un codage par transformation adaptative qui décide, à la volée, s'il convient ou non d'utiliser des blocs de 4 x 4 ou 8 x 8 pixels. Par exemple, des blocs de 4 x 4 sont utilisés pour les parties de l'image comportant beaucoup de détails, tandis que les parties peu détaillées sont transformées par blocs de 8 x 8.

Enfin, on note que chaque profil dispose d'une flexibilité suffisante pour soutenir une large rangée d'applications.

Conclusion

Dans ce chapitre, nous avons présenté les notions de base concernant les images numériques et la compression vidéo. Le processus de compression est constitué de plusieurs modules employés pour réduire le volume des données vidéo, aussi bien dans une image (compression spatiale) qu'entre deux séries d'images (compression temporelle).

De nombreux efforts de recherche ont été concentrés sur la réduction de la quantité des calculs nécessaires à l'opération d'estimation de mouvement, tout en préservant le maximum de qualité.

Chapitre 2 Simulation et profilage d'une chaîne de codage-décodage MPEG.

Dans ce chapitre, nous allons décrire les différents éléments et étapes qui interviennent dans une démarche d'implémentation d'une chaîne de codage-décodage MPEG sur une plateforme matérielle. Nous étudions par la suite de façon détaillée les phases de simulation, profilage et partitionnement logiciel/Matériel du module d'estimation de mouvement.

2.1 Les séquences de test

Pour les besoins de notre étude, nous avons utilisé des séquences de test largement utilisées pour la mise au point des chaines de codage-décodage [20], notamment les séquences naturelles « Tennis player » et « Caltrain » de la figure 2.1. La première de ces séquences présente des déplacements locaux simples sur peu de régions de l'image (balle, bras et main du joueur), ainsi qu'un changement d'échelle global (zoom arrière). La deuxième séquence correspond au suivi du train est beaucoup plus complexe. La scène présente de très nombreux détails et aucun objet n'est fixe. Les mouvements sont de type rotation, translation, changement d'échelle global (zoom arrière) et local (rapprochement-éloignement d'objets). Il est évident que l'analyse d'une telle scène est complexe.

Fig. 2.1 : séquences« Tennis player » et « Caltrain » [20]

2.2 Les étapes de la simulation d'un encodeur vidéo H.264/MPEG-4 AVC

Le schéma de la figure 2.2 décrit lachaine de traitement appliquée aux trames d'une séquence vidéo lors de leur compression par un encodeur MPEG, les modules intervenant sont ceux décrit dans la deuxième section du chapitre 1 adapté au profil du h264.

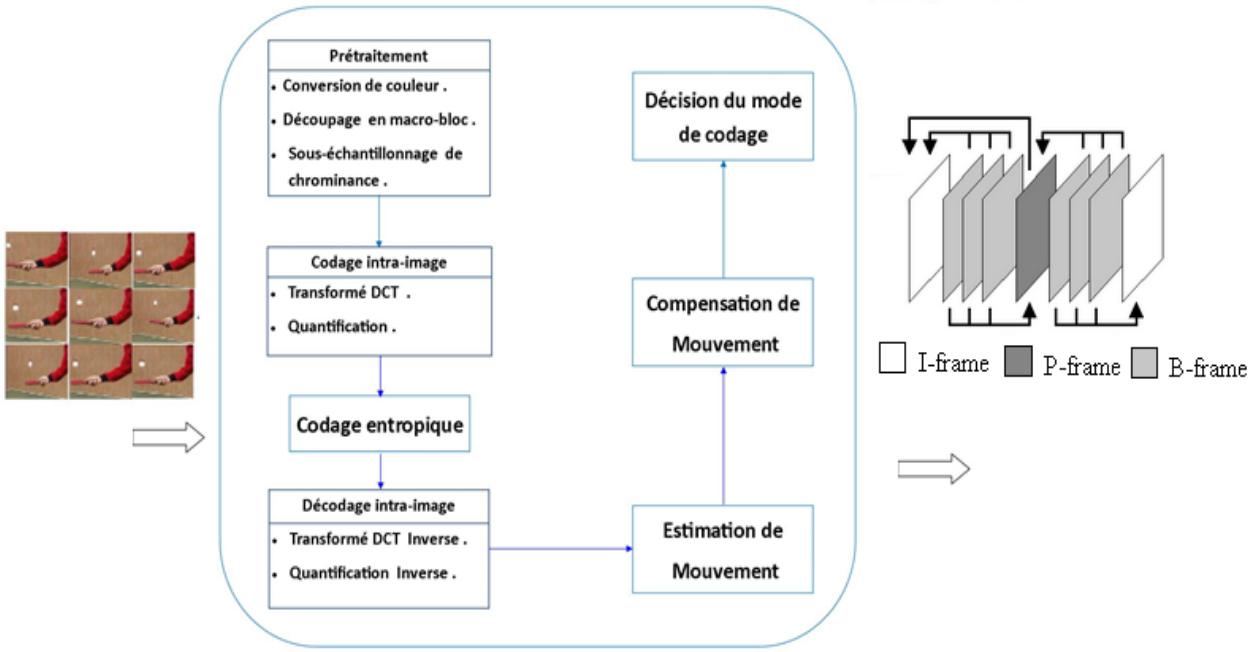


Fig. 2.2 : Etapes de traitement de compression vidéo d'un encodeur MPEG4

Afin d'illuster les differents traitements apporté par chaque module nous proposons de les acompagnés par un exemple de compression de trames video.

On choisit une sequence d'image « Tennis player »tirer de la base de données des sequences video de reference pour encodeurs video[20].

2.2.1 Module de pretraitement

L'etape de pretraitement presenté par la figure 2.3essentiel pour le processus de compression

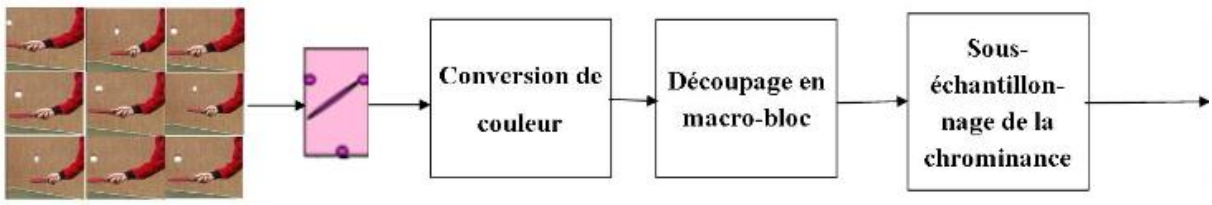


Fig. 2.3 : Schema synoptique de l'etape de pretraitement

Elle commence par calculer les composantes YCrCb apartir de ses composantes RVB à l'aide de l'equation 1.2comme le montre la figure 2.4.

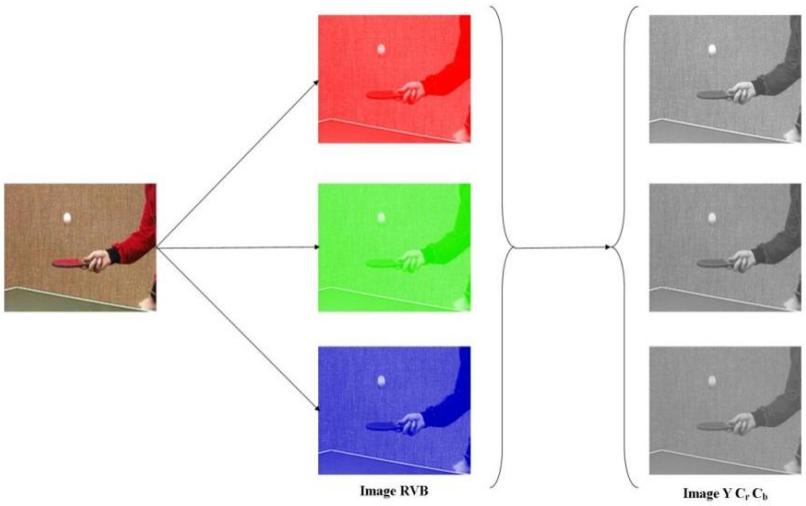


Fig. 2.4 : Conversion YCrCb d'une image RVB

Les trames resultantes vont subir un decoupage en macro-blocs, la taille de ces macro-bloc influe sur la qualité de compression,dans un encodeur H.264 MPEG-4 AVC la taille du macro bloc peut allerjusqu'à 4x4 pixels (figure 2.5).

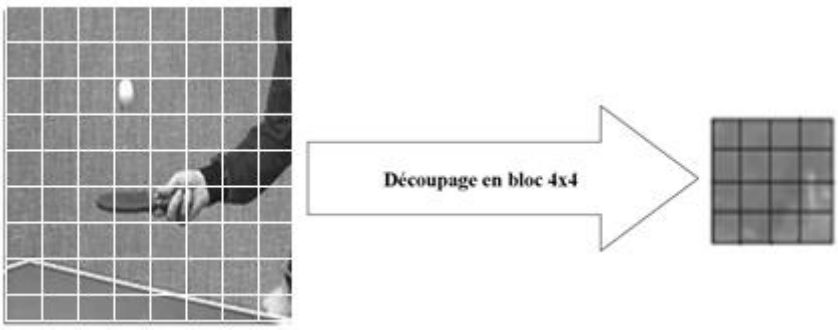


Fig. 2.5 : Fragmentation des composantes de l'image en blocs 4x4 pixels

Les macro-blocs de chrominance Cr et Cb de 4x4 pixels étant moins informatifs à l'œil humaine sont donc sous-échantillonnés pour donner des macro-blocs 2x2.

2.2.2 Module de compression spatiales

Le module de prétraitement délivre des macros blocs de 4x4 pixels pour les trames de luminance et de 2x2 pour des trames de chrominance. Le module de compression spatiale a pour rôle d'éliminer la corrélation intra-image (ou plus précisément intra-bloc) en deux étape comme l'illustre la figure 2.6.

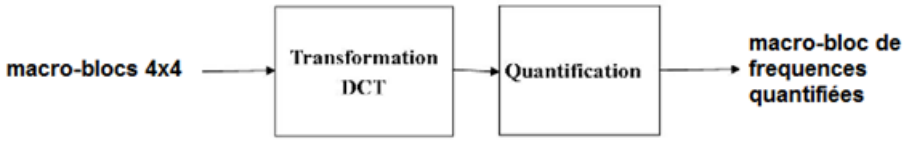


Fig. 2.6 : Schéma synoptique du module de compression spatiale.

Application de la DCT :

L'application de la DCT au bloc 4x4 de la figure 2.5sous sa forme numérique à niveau de gris donne la matrice des fréquences f :

$$Mbloc_{original} = \begin{bmatrix} 229 & 228 & 225 & 223 \\ 229 & 224 & 221 & 222 \\ 234 & 232 & 230 & 229 \\ 223 & 225 & 224 & 221 \end{bmatrix} \qquad f = \begin{bmatrix} 904.7500 & 7.7505 & 0.2500 & -0.2338 \\ -0.0040 & 222.4874 & 1.9831 & -0.1768 \\ -5.7500 & -1.4419 & -3.2500 & -0.2146 \\ 11.0962 & -0.1768 & -1.0920 & 0.0126 \end{bmatrix}$$

L'application de la DCT sur chaque macro-bloc résulte une matrice de fréquences de 16 éléments (4x4) concernant le signal luminance. La valeur de la position (0,0) est le coefficient continu (DC) qui représente l'énergie maximale. Les 15 coefficients représentent les détails (AC). Plus on s'éloigne de l'origine plus l'énergie des coefficients décroit très rapidement. Les coefficients de haute fréquence sont donc les plus faibles, cette opération est théoriquement sans perte d'informations, les coefficients initiaux peuvent être retrouvés en appliquant la « DCT inverse » au résultat de la DCT. Dans la pratique, une certaine perte d'informations reste cependant possible en raison des erreurs d'arrondis introduites en cours de calcul.

Application de la Quantification :

Le calcul effectué par le module de quantification est le suivant :

$$F(i,j) = \left\lfloor \frac{f(i,j)}{Q(i,j)} \right\rfloor, 1 \leq i,j \leq 4$$

Où $\lfloor x \rfloor = entierdirectementinférieur$ à x

Avec la matrice de quantification Q suivante [21]

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 \\ 12 & 12 & 14 & 19 \\ 14 & 13 & 16 & 24 \\ 14 & 17 & 22 & 29 \end{bmatrix}$$

On obtient la matrice des fréquences quantifiée F*

$$F^* = \begin{bmatrix} 51 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Module Quantification inverse/DCT Inverse

A cette étape le module quantification inverse délivre un macro-bloc dont seuls un nombre réduit de coefficients sont non-nuls, et peuvent être exploités pour reconstruire l'image originale nécessaire pour l'étape de l'estimation de mouvement et ce en appliquant la DCT inverse

$$f^*(i,j) = F^*(i,j) \times Q(i,j), 1 \leq i,j \leq 4 \qquad F^* = \begin{bmatrix} 912 & 11 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 14 & 0 & 0 & 0 \end{bmatrix}$$

La DCT inverse appliqué à F^*de l'exemple précédant, donne le macro-bloc de la figure 2.4 :

$$Mbloc^* = \begin{bmatrix} 233 & 231 & 228 & 226 \\ 227 & 224 & 221 & 219 \\ 236 & 234 & 231 & 228 \\ 229 & 227 & 224 & 222 \end{bmatrix}$$

Le macro-bloc est décompressé alors avec une redondance spatiale réduite, l'erreur de la reconstitution est donnée par:

$$err = \left| Mbloc_{original} - Mbloc^* \right| = \begin{bmatrix} 4 & 3 & 3 & 3 \\ 2 & 0 & 0 & 3 \\ 2 & 2 & 1 & 1 \\ 6 & 2 & 0 & 1 \end{bmatrix}$$

Soit un *PSNR* = 39.8782 *dB* qui correspond à une très bonne fidélité de reconstruction de l'image originale [22].

2.2.3 Module de codage entropique

Avant le codage des coefficients AC (hautes fréquences) quantifiés, on commence par l'ordonnancement de ces coefficients en fonction de leurs énergies décroissantes par rapport à l'origine. Pour cela un algorithme, dit Zigzag est appliqué (figure 2.7) à l'exemple précédent.

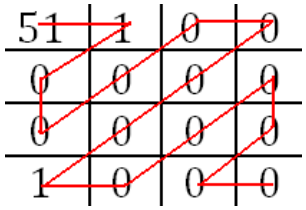


Fig. 2.7 : Scanning en Zigzag appliqué à la matrice des fréquences quantifiées

Ce qui donne la suite de coefficients AC suivante : {1,0,0,0,0,0,0,0,1,0,0,0,0,0,0}

Le codage entropique permet de gérer plus efficacement ces coefficients.

Le codage entropique exploite la longueur des séquences de nul pour générer un code de longueur variable grâce à des dictionnaires tabulés [23].

2.2.4 Module de compression temporelle

On cherche à estimer le déplacement (dx, dy) de chaque bloc de l'image de référence par rapport á l'image courante. La compensation de mouvement est l'opération inverse de l'estimation de mouvement, ayant pour but de reconstruire une image actuelle à partir du champ de vecteur et d'une image de référence, malgré que cette fonctionnalité soit propre au décodage, elle est utilisée dans un encodeur pour estimer l'erreur résiduelle

Il existe une multitude de techniques d'estimations de mouvement, pour les besoins de notre simulation, on a fait le choix d'utiliser la méthode de mise en correspondance de blocs (block-Matching) associée à la métrique SAD en parcours dit exhaustive. Une étude plus détaillé de ces techniques d'estimation de mouvement est présentée dans le chapitre suivant.

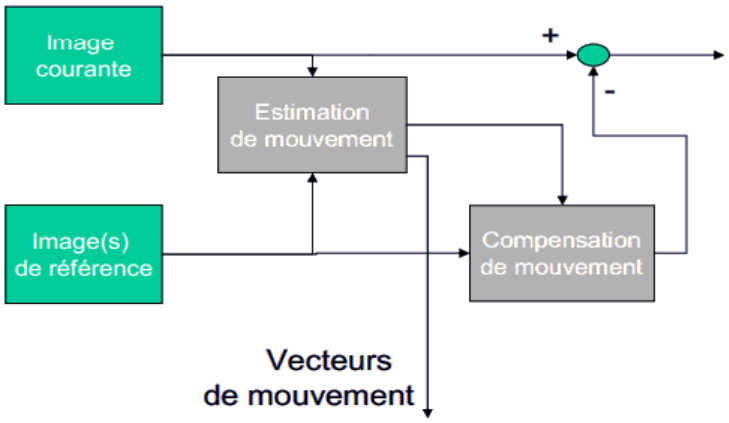


Fig. 2.8 : Estimation et compensation de mouvement [24]

2.3 Simulation et vérification des spécifications fonctionnelles de la norme MPEG-4

Pour la validation, nous avons, dans un premier temps, mené une série de simulations à l'aide du logiciel Matlab. L'ensemble des programmes écrits à cet effet permet de simuler les spécifications fonctionnelles d'un codeur vidéo MPEG

2.3.1 Spécifications fonctionnelles

Mais avant d'entamerla simulation, une première phase serait d'analyser le code de l'application afin de connaitre la liste des fonctions appelées.

Le Tableau 2.1 donne une description des fonctions principales du code Matlab utilisé.

Nom de la fonction	**module associée**	**Traitement**
Principal	Conversion de couleurs	Calculer les composantes YCrCb à partir de l'image RVB.
Découpage	Découpage en bloc 4x4	Fragmenter l'image en macro-blocs de 4x4 pixels.
SE_Chrom	Sous-échantillonnage de la chrominance	sous-échantillonner le signal de. chrominance.
dct2	Transformation DCT	Appliquer la transformée en cosinus discrète aux macro-blocs.
Qaunt	Quantification	Quantifier la matrice des fréquences et éliminer les hautes fréquences.
VLC	Codage entropique	générer un code de longueur variable pour la matrice des fréquences.
Iquant	Quantification Inverse	Récupérer la matrice des fréquences réduite.
idct2	Transformation DCT Inverse	Récupérer les macro-blocs.
EstimationDeMouvement	Estimation de mouvement	Calculer le champ des vecteurs de mouvement entre l'image actuelle et l'image de référence.
CompensationDeMouvement	Compensation de mouvement	Appliquer le champ de vecteur de mouvement a l'image de référence pour connaitre la qualité de la compensation de mouvement.
ModeCodage	Décision mode de codage	Choisir le type de codage que devra subir l'image compressée.

TAB. 2.1: Correspondance des fonctions principales d'un codeur vidéo MPEG-4

2.3.2 Simulation

Dans une première expérience, nous avons utilisé comme flux d'entrée vidéo, deux images successives (352×240 au format RAS) de la séquence vidéo Tennis. Les résultats de l'application des traitements de la chaine de compression MPEG sont présentés sur la figure 2.9. On constate que l'image reconstruite après l'étape de compensation de mouvement correspond parfaitement à l'image de référence avec un PSNR de 33.6165 dB.

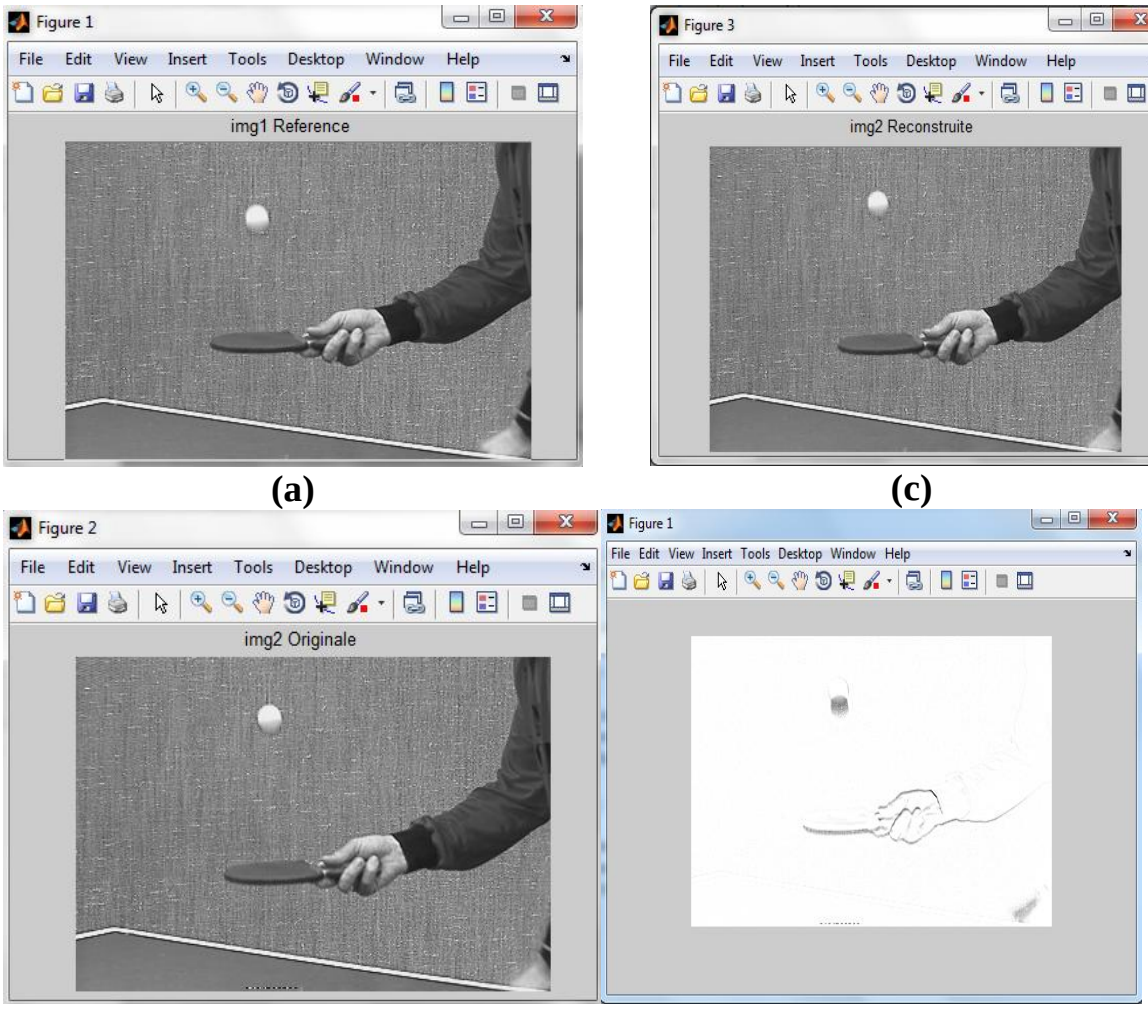


Fig. 2.9 : Résultats de simulation des trames « tennis » (352×240),
(a) image de référence, (b) image originale, (c) image reconstruite, (d) différence de mouvement.

Des résultats similaires ont été observés avec la séquence vidéo « caltrain ». Dans la figure 2.10, nous avons également représenté l'erreur de prédiction entre l'image réelle et l'image prédite. Comme on peut le constater, les plus grandes erreurs sont situées dans des régions avec un mouvement différent. Ainsi, le mouvement complexe engendre une chute de la valeur du PSNR (voir le tableau 2.2), ce dernier reste dans les limite d'une reconstitution acceptable.

Sequence video	Type de mouvement	PSNR
Tennis	Translation horizontale	33.6165 dB
Train	Rotation + deplacement	25.8404 dB

TAB. 2.2 : Resultat de decompression pour differents type de mouvement.

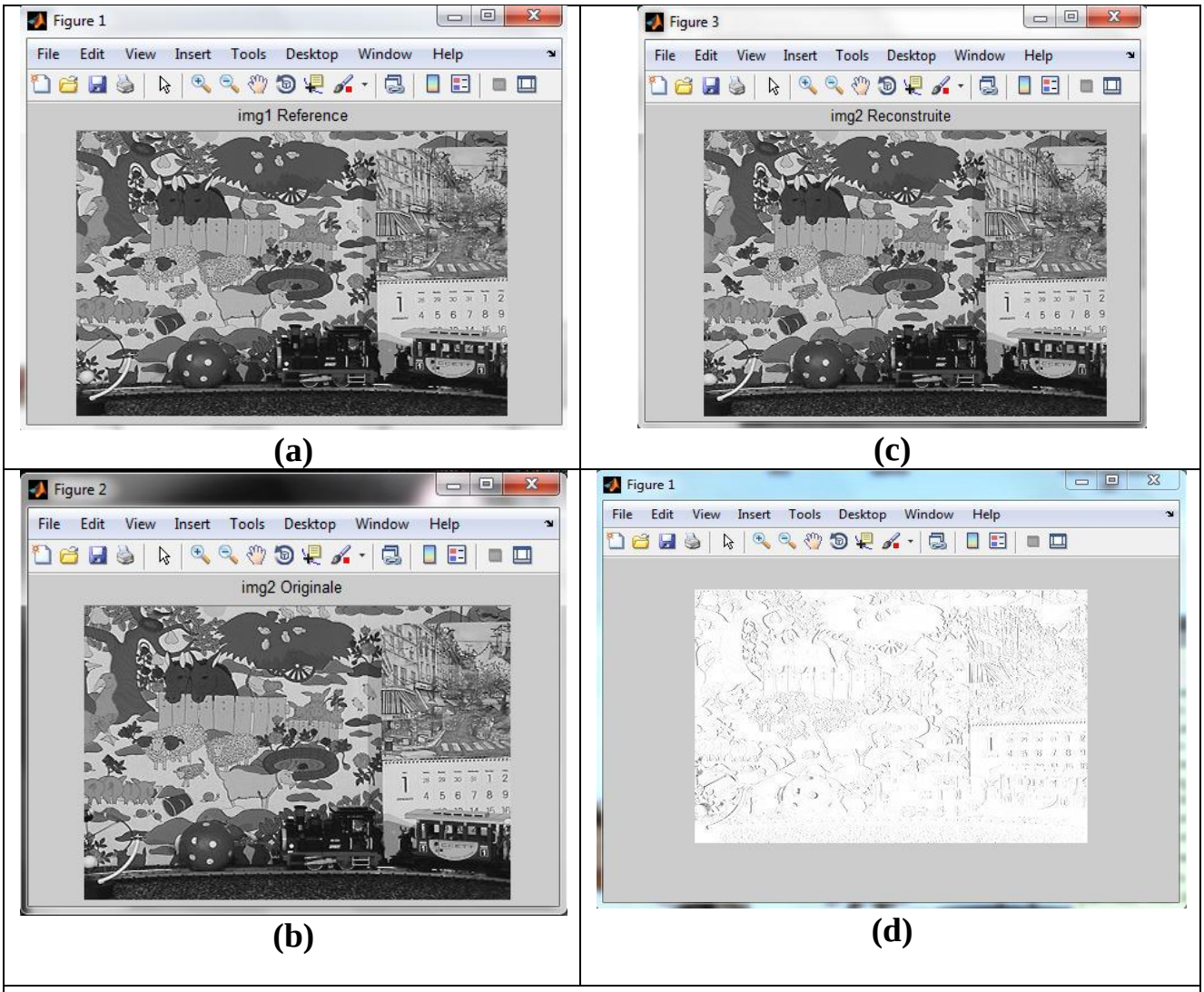


Fig. 2.10 : Résultats de simulation des frames «train» (352×240)
(a) image de référence, (b) image originale, (c) image reconstruite, (d) différence de mouvement entre deux trames.

2.4 Profilage et partitionnement logiciel/matériel :

Après avoir simulé sur MATLAB les spécifications fonctionnelles du codeur vidéo MPEG et afin de pouvoir proposer une implémentation matérielle, une étude détaillée du code du système est nécessaire, même indispensable à l'estimation globale des performances et à l'exploration d'architecture.

2.4.1 Profilage

Dans une seconde étape on cherche à identifier les goulots d'étranglement affectant les performances des simulations. Pour cela il faut suivre l'arbre des appels des fonctions ainsi que leurs coûts. Le coût peut être un temps d'exécution, un nombre de cycles ou d'instructions, une consommation ou toute autre information pouvant être fort utile dans les phases d'exploration.

Un système de compression vidéo du type H.264/MPEG-4 AVC est destiné à travailler en temps réel donc le temps d'exécution est la mesure de performance prioritaire pour passer à l'étape d'optimisation.

MATLAB offre un outil appelé "profiler" qui est en mesure de comptabiliser le temps consommé par chaque fonction lors de son exécution, puis de présenter les résultats de cette analyse sous forme graphique comme indiqué dans la figure 2.11.

Profile Summary

Generated 23-May-2013 18:39:02 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
AfficherResultat	1	1.144 s	0.331 s	
CalculMetriqueSAD	40303872	188.975 s	188.975 s	
ChargerImages	1	0.087 s	0.026 s	
CompensationMvt	1	0.790 s	0.790 s	
ConverssionCouleur	1	0.124 s	0.014 s	
EstimationMvt	1	678.595 s	334.446 s	
SelectionnerBloc	40353024	154.753 s	154.753 s	
SelectionnerBloc2	55296	0.223 s	0.223 s	
TrouverBornes	98304	0.421 s	0.421 s	
allchild	3	0.001 s	0.001 s	
axescheck	9	0.006 s	0.006 s	
cellstr	3	0 s	0.000 s	
cla	6	0.169 s	0.009 s	
close	1	0.035 s	0.005 s	
close>checkfigs	2	0 s	0.000 s	
close>request_close	1	0.030 s	0.004 s	
close>request_close_helper	2	0.011 s	0.002 s	
closereq	1	0.015 s	0.012 s	

Fig. 2.11 : Représentation graphique du profiler de Matlab au cours de la simulation du codeur MPEG pour une séquence vidéo 768 × 576 pixels.

On applique le script de simulation pour différentes résolutions standards de séquence vidéo, issues du le monde de la vidéo numérique. Le tableau 2.3 résume les résultats du profilage.

Nom du module	CIF 352 × 288 pixels	SD(DVD) 720×576 pixels	SD 768×576 pixels	HD 1280×720 pixels	Full HD 1920×1080 pixels
Conversion de couleurs + Sous-échantillonnage de la chrominance	0.028 s	0.120 s	0.124 s	0.057 s	0.070 s
Découpage en bloc 4x4	0.055 s	0.196 s	0.223 s	0.408 s	1.078 s
Transformation DCT	2.211 s	9.051 s	10.051 s	19.682 s	46.507 s
Quantification	0.043 s	0.145 s	0.183 s	0.353 s	0.867 s
Codage entropique	0.043 s	0.157 s	0.178 s	0.329 s	0.852 s
Quantification Inverse	0.029 s	0.111 s	0.158 s	0.280 s	0.702 s
Transformation DCT Inverse	2.655 s	11.101 s	12.112 s	24.246 s	57.244 s
Estimation de mouvement	135.176 s	567.343 s	678.595 s	1408.191 s	4024.638 s
Décision mode de codage	0.220 s	0.520 s	0.580 s	0.710 s	1.230 s
Compensation de movement	0.110 s	0.598 s	0.790 s	1.984 s	5.962 s
Temps Total	141 s	590 s	703 s	1457 s	4140 s

TAB. 2.3 : Résultats du profilage pour différentes définition vidéo.

L'analyse des résultats du profilage (Tableau 2.3) démontre clairement que le module de l'estimation de mouvement et le plus couteux en temps de calcul avec un taux dépassant 95% du temps total. De plus comme le montre le Tableau 2.4 l'estimation du mouvement est telle que : plus la définition des séquences vidéo augmente plus sa consommation en temps de calcul devient prédominante (figure 2.12).

Définition	**CIF** 352 × 288 pixels	**SD(DVD)** 720×576 pixels	**SD** 768×576 pixels	**HD** 1280×720 pixels	**Full HD** 1920×1080 pixels
Consommation du Module de l'estimation de mouvement	95.86%	96.16%	96.53%	96.65%	97.21%

TAB. 2.4 : Taux de calcul du module de l'estimation de mouvement pour différentes définitions.

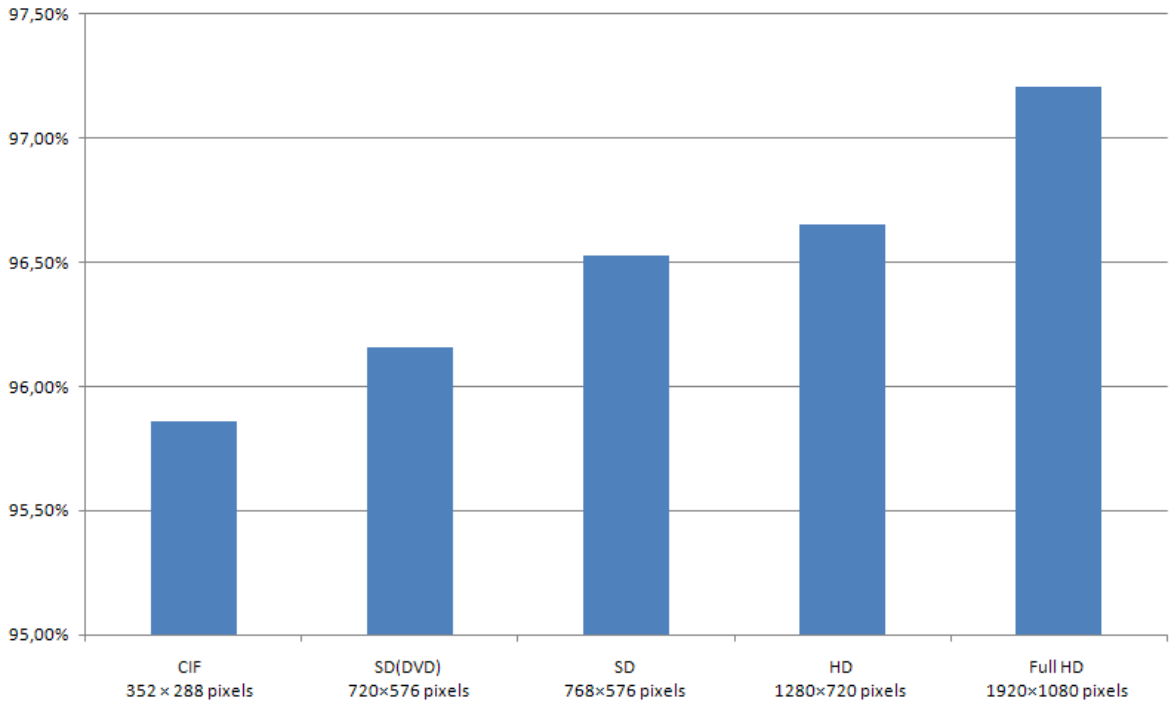


Fig. 2.12 - Taux de calcul du module d'estimation de mouvement pour différente définition

2.4.2 Partitionnement

Les simulations menées montrent le bon fonctionnement des modules constituant le codeur MPEG en reconstituant les images à partir des images de référence et des champs de vecteur avec une bonne qualité de compression, cependant en utilisant un langage haut niveau (algorithme décrit en MATLAB), le système est implémenté sur une architecture logicielle (figure 2.13a) avec un seul processeur (module de calcul) qui effectue tous les traitements nécessaires et une RAM permettant de stocker les données. Comme le montre le Tableau 2.3, le temps de traitement est très grand et ne convient pas à un traitement classique du flux vidéo où il faut défiler 25 images par seconde afin de maintenir une bonne fluidité de l'animation.

Il devient très clair que l'optimisation et la réduction du temps de traitement revient implicitement à optimiser le module de l'estimation de mouvement. L'utilisation d'un accélérateur matériel est la solution la plus judicieuse. Le modèle d'architecture est présenté dans la figure 2.13b.

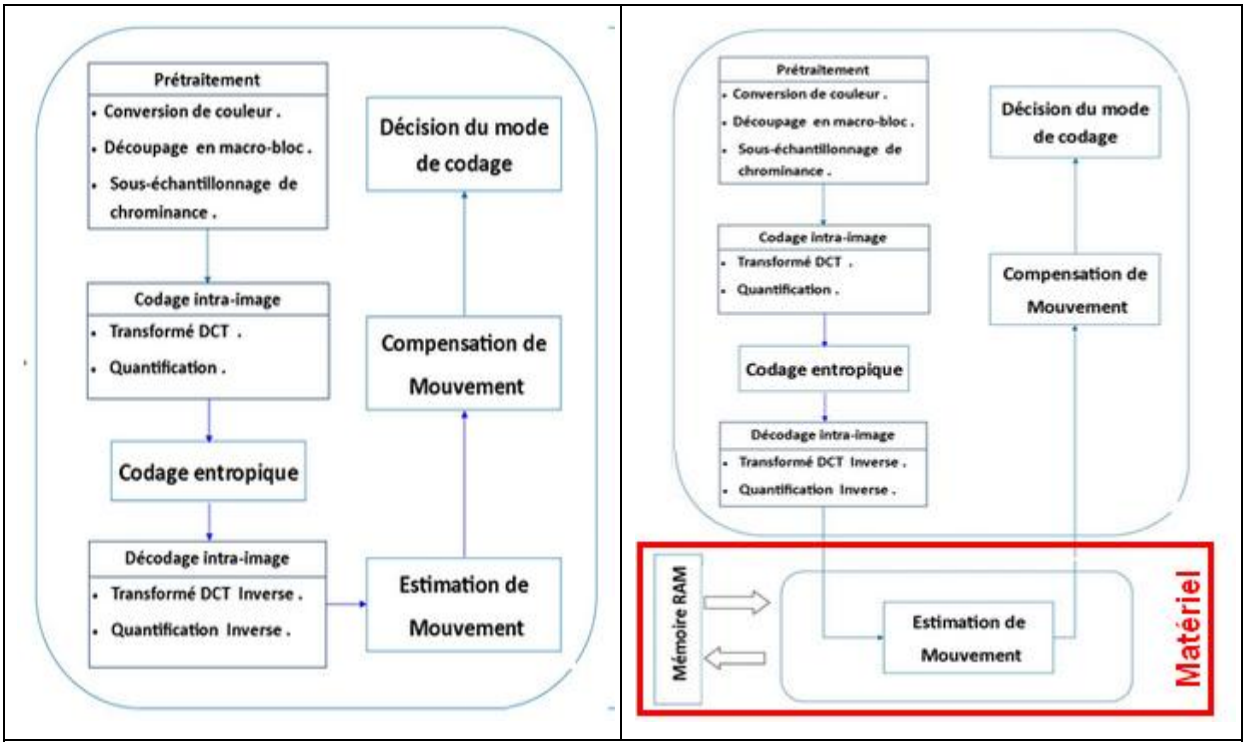


Fig. 2.13 - (a) Module de calcul logiciel, (b) Modèle avec accélérateur matériel

L'utilisation d'un circuit FPGA comme accélérateur matériel (figure 2.13 (b)) se présente comme la solution la plus performante et ce en exploitant la bonne puissance de calcul et le parallélisme de traitement offert par le FPGA afin de réduire le temps de compression et de respecter les temps de traitement en temps réel.

Conclusion

Dans cette partie, les différents modules du codeur MPEG (en l'occurrence : Analyse en DCT, Quantification, Estimation et compensation de mouvement) ont été simulés sous Matlab. L'analyse des résultats de profilage montre que l'opération d'estimation de mouvement occupe la partie la plus importante dans un encodeur vidéo. En effet, d'une part la qualité de l'estimation influe sensiblement sur les performances de compression et d'autre part c'est l'opération qui nécessite plus de temps de calcul. L'implémentation d'un accélérateur matériel pour le calcul des vecteurs de mouvement parait comme une solution alternative plus efficace pour réduire de manière significative la charge de calcul exigée.Une question reste encore posée : quel est le meilleur algorithme de calcul à implémenter sur la carte FPGA ?

Pour répondre à cette question, le chapitre suivant présente l'ensemble des techniques et méthodes utilisées pour l'estimation du mouvement.

Chapitre 3 Techniques d'estimation de mouvement

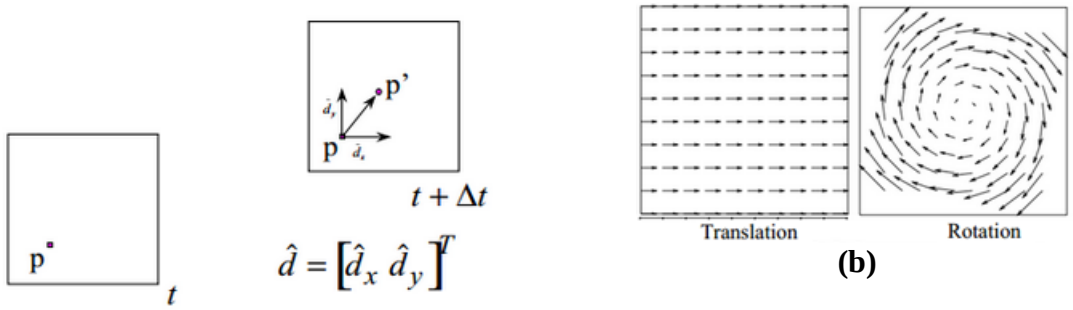
Introduction

Dans Ce chapitre nous présentons les techniques d'estimation de mouvement utilisées dans une chaine de compression MPEG. Nous commençons par décrire brièvement l'ensemble de méthodes, dites classiques. Nous nous intéressons ensuite aux méthodes adaptées à une implémentation matérielle. Enfin, nous montrons les orientations que nous avons choisies et qui conduisent à la méthode implémentée dans le cadre de ce projet de fin des études.

3.1 Caractérisation du mouvement

L'estimation du mouvement dans des séquences temporelles d'images bidimensionnelles est un des problèmes fondamentaux en traitement d'images. Les images représentent très souvent la projection de scènes réelles 3D. Pour cette raison, nous pouvons identifier trois types de mouvement : le mouvement réel, le mouvement apparent et le mouvement estimé.

Le mouvement apparent, observé à partir des changements de la distribution spatiale d'intensité lumineuse, est aussi appelé le flux optique. Ce mouvement est très souvent différent du mouvement réel et représente en général la projection du mouvement réel 3D (via des systèmes d'acquisition) dans le plan image 2D [25] (Fig.3.1. (a)).



(a) **(b)**

Fig.3.1 (a) Caractérisation du mouvement
(b) Exemple de °flux optiques engendrés par des mouvements de translation et de rotations [25].

Le premier niveau de caractérisation locale du mouvement correspond au calcul du flot optique qui peut être complétement défini par un des paramètres suivants (Figure 3.1) :

- Vecteur de déplacement élémentaire $\vec{d} = (dx, dy)^T$
- Vecteur vitesse $\vec{w} = (dx/dt, dy/dt)^T$

En général, le problème d'estimation du mouvement est complexe et revient à la recherche du flot optique (champ déplacement /vitesse) qui minimise en chaque point p un certain critère d'erreur.

Bien que ces méthodes de résolution soient très nombreuses, il est possible de les classer en trois familles principales :

- Méthodes différentielles;
- Méthodes fréquentielles ;
- Méthodes par appariement.

3.2 Méthodes différentielles (basées gradient)

Les méthodes différentielles d'estimation du mouvement sont basées sur les gradients spatiaux et temporels de l'intensité lumineuse des pixels. Le principe de ces méthodes repose sur une hypothèse forte, qui est celle de la conservation de l'intensité lumineuse d'un pixel le long de la trajectoire du mouvement. Cette hypothèse de conservation peut s'écrire sous la forme :

$$d\,I\frac{(x\,,\ y,t)}{dt}\ =0 \qquad (3.1)$$

Où $x\ et\ y$ sont les variables spatiales, t est la variable temporelle et $I(x,y,t)$ est l'intensité du pixel de coordonnées (x,y) dans l'image acquise à l'instant t.

Dans le cas discret, l'hypothèse de conservation de l'intensité lumineuse peut s'exprimer, par l'équation DFD (Displaced Frame Difference) [26], des différences entre les images déplacées donnée par :

$$DFD(x\ ,d)=I_{n-1}(x\ +\ d)-\ I_n(x) \qquad (3.2)$$

Où $I_n(x)$ est l'intensité lumineuse de l'image *n* et *d* le déplacement du point *x* entre les images $n-1$ et n.

Un développement de Taylor du premier ordre de l'équation (3.1) permet d'obtenir la relation suivante :

$$\frac{\partial i(x,y,t)}{\partial x}d_x+\frac{\partial i(x,y,t)}{\partial y}d_y+\frac{\partial i(x,y,t)}{\partial t}d_t=0 \qquad (3.3)$$

où $d_x\ et\ d_y$ Sont les déplacements du pixel courant qui doivent être estimés et dt est la période temporelle d'acquisition des images [26].

Si on divise l'équation (3.3) par dt, alors nous obtenons l'équation de contrainte de mouvement(ECM), également appelée équation du flux optique(EFO) [25].

$$\frac{\partial i(x,y,t)}{\partial x} w_x(x,y,t) + \frac{\partial i(x,y,t)}{\partial y} w_y(x,y,t) + \frac{\partial i(x,y,t)}{\partial t} = 0 \qquad (3.4)$$

Où: $w_x(x,y,t) = \frac{d_x}{d_{t,}}$, $w_x(x,y,t) = \frac{d_y}{d_t}$ sont les deux composantes de la vitesse.

Tel que l'équation (ECM) peut être écrite, en chaque point (x,y,t) :

$$I^x.w_x + I^y.w_y + I^t = 0 \quad ou : \langle \nabla I, w \rangle + \frac{dI}{dt} = 0 \qquad (3.5)$$

Où: $\nabla I = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right) = (I^x, I^y)$ est le gradient spatial de l'image avec les composantes $I^x = \frac{\partial I}{\partial x}$et $I^y = \frac{\partial I}{\partial y}$, respectivement $I^t = \frac{\partial I}{\partial t}$ est le gradient temporel de l'intensité, et $\langle .,. \rangle$ représente le produit scalaire entre deux variables.

Il s'agit donc d'une seule équation avec deux inconnues. Afin de pouvoir résoudre cette équation, différentes méthodes introduisant des contraintes supplémentaires ont été proposées [27, 28]. Une comparaison des performances des méthodes différentielles d'estimation du mouvement les plus utilisées est faite dans [29].

Les méthodes récursives basées gradient conduisent à un champ de vecteur dense qui convient parfaitement à des traitements pixel par pixel, tel que le filtrage ou la conversion de standard. Mais étant donné que la DFD n'est calculée que sur un pixel, ces méthodes sont très sensibles au bruit. Afin d'augmenter la robustesse, plusieurs pixels voisins peuvent être pris en compte.

Enfin, comme ces méthodes sont basées sur des développements limités, elles ne sont pas adaptées à l'estimation de grands déplacements. Une approche hiérarchique peut améliorer les résultats grâce à une représentation multi-résolution des images.

3.3 Méthodes fréquentielles

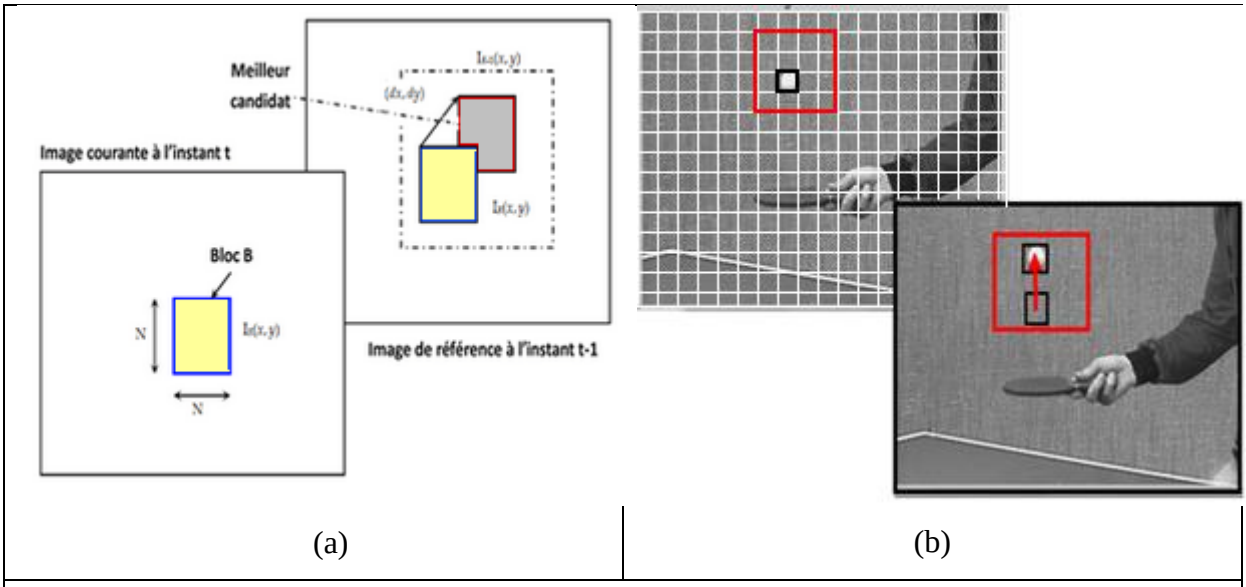
Les techniques fréquentielles d'estimation du mouvement entre deux images sont fondées sur les points suivants :

- Une translation dans le domaine de l'image se caractérise par un déphasage dans le domaine fréquentiel [30]. Il suffit donc, en théorie de considérer ce déphasage pour deux couples (u, v) pour calculer le déplacement (dx, dy).
- L'essentiel de l'énergie de la transformée spatio-temporelle d'une séquence d'images en translation uniforme sera concentré sur un plan orthogonal au mouvement [30]. Il suffit donc qu'à déterminer ce plan (plan de vitesse) pour obtenir localement la valeur du champ de mouvement. Plusieurs transformations ont été utilisées pour construire de tels filtres comme les transformées de Fourier court terme [31], de Wigner-Ville [30], et de Gabor [32]

On distingue deux approches : les méthodes basées sur l'énergie [33] et les méthodes exploitant la phase du signal [34]. Ces méthodes sont peu sensibles au bruit et aux variations d'intensité lumineuse et présentent de nombreux atouts tels que la simplicité de l'interprétation physique ou la cohérence avec d'autres traitements (compression, restauration d'images) utilisant les mêmes modèles. Mais la taille de la transformée contraint le déplacement maximal détectable et implique beaucoup de calculs.

3.4 Méthodes de mise en correspondance de blocs (BM : Block Matching)

La mise en correspondance de blocs est légèrement plus simple. On cherche à estimer le déplacement (dx, dy) de chaque bloc de l'image de référence par rapport à l'image courante. (figure3.2.a).



(a) (b)

Fig.3.2 : (a) Principe de mise en correspondance, (b) Exemple de fenêtre de recherche

Chaque image est découpée dans un premier temps en blocs, puis pour chaque bloc de l'image courante à l'instant (t), le bloc le plus ressemblant (au sens de la minimisation d'un critère d'erreur) est recherché dans l'image de référence à l'instant (t-1), (Figure3.2). Contrairement aux techniques précédentes ou le mouvement est évalué à partir de certains paramètres, ici on part d'une solution et sa pertinence est évaluée.

Deux types de prédiction sont possibles (et nécessaires pour la norme MPEG) : la prédiction avant (forward") et la prédiction arrière (backward"). La prédiction forward consiste à diviser l'image (n) en blocs et à trouver leur position dans l'image suivante (n+1). Comme le montre la figure 3.3, le block-Matching avec prédiction forward répond pour chaque bloc a la question "Ou va le bloc courant ?".

En revanche, la prédiction backward, divise l'image (n) en blocs et cherche leur position dans l'image précédente (n-1). Dans ce cas la question est "d'où vient le bloc courant ?".[35]

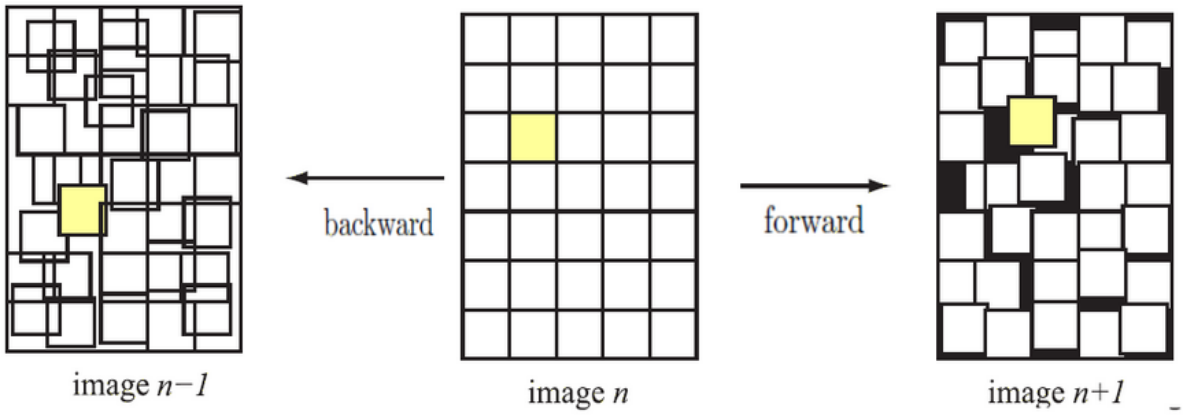


Fig.3.3. la prédiction forward et backward [35]

3.4.1 Zone de recherche

La ressemblance entre deux blocs est calculée sur l'ensemble du bloc. Ceci sous-entend une hypothèse supplémentaire selon laquelle tous les pixels d'un bloc effectuent le même mouvement [36], ce qui correspond. A un modèle de translation, mais on peut considérer aussi des modèles plus complexes (rotation, translation et rotation).

Pour accélérer le temps de calcul, on contraint l'estimation BM par une valeur maximale de vecteur (zone de recherche fig.3.2.b). Cette contrainte se justifie par le fait que les images successives se ressemblent beaucoup ; de plus même si un vecteur est hors plage, la prédiction sera mauvaise, l'objet mal codé, mais le déplacement est si rapide que l'œil ne verra pas forcément les erreurs.

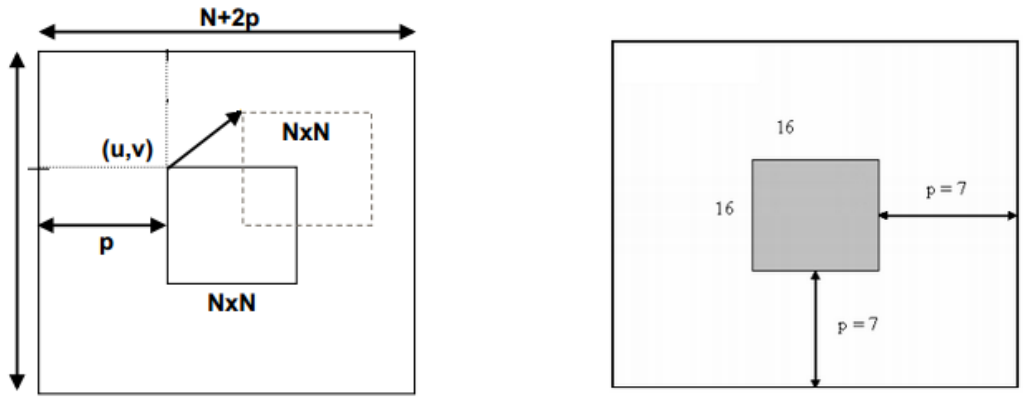


Fig.3.4 : Les limites de la zone de recherche avec exemple pour N=16 et p=7[36].

Dans l'exemple de la figure 3.4, pour un bloc B de dimension $N * N$, la recherche est limitée à une fenêtre de dimensions $N + 2.P$ [33]: $(2.P + N)^2\ pixels$ où P est la valeur maximale du déplacement.

Les méthodes de mise en correspondance de blocs peuvent être classées en fonction :

- ❖ du critère de mise en correspondance ;
- ❖ de la mise en œuvre nécessitant d'optimiser 2 éléments: la dimension du bloc et celle de la fenêtre de recherche;
- ❖ de la stratégie de balayage (ou de recherche) de la fenêtre de recherche.

3.4.2 Critères de mise en correspondance

Le vecteur de déplacement « d » optimal minimise l'erreur. L'opération d'estimation de mouvement s'appuie sur l'évaluation de toutes les valeurs de « d » candidates. Le bloc qui correspond au mieux au bloc initial et celui qui correspond au minimum de ces valeurs.

Il existe plusieurs critères permettant d'évaluer l'amplitude de l'erreur de prédiction. Les plus courants sont présentés ci-après avec les conventions suivantes :

- ❖ Les expressions sont données pour un bloc B de taille $N * M$, repéré sur le pixel de coordonnées $P = (x, y)$ dans l'image courante.
- ❖ Le décalage entre un bloc candidat et le bloc B de référence est noté $d = (i\,, j)$ et est borné par la taille de la zone de recherche.
- ❖ La position du meilleur bloc candidat est notée $(\hat{\imath}\,, \hat{\jmath})$.

$$\left.\begin{aligned} &SAD(i,j) = \sum_{k=1}^{N}\sum_{l=1}^{M}|B_2(k,l) - B_1(k+i\,, l+j)| \\ &(\hat{\imath}\,, \hat{\jmath}) = \arg\min\,(SAD(i,j)) \end{aligned}\right\} \quad (3.6)$$

Somme des carrés des différences (SSD)

$$\left.\begin{aligned} &SSD(i,j) = \sum_{k=1}^{N}\sum_{l=1}^{M}(B_2(k,l) - B_1(k+i\,, l+j))^2 \\ &(\hat{\imath}\,, \hat{\jmath}) = \arg\min\,(SSD(i,j)) \end{aligned}\right\} \quad (3.7)$$

Coefficient d'inter-corrélation (CC : corrélation croisée) [37]

$$\left.\begin{array}{l} CC(i,j) = \sum_{k=1}^{N} \sum_{l=1}^{M} B_2(k,l) * B_1(k+i\,, l+j) \\ (\hat{\imath}\,, \hat{\jmath}) = \arg\max\,(CC(i,j)) \end{array}\right\} \quad (3.8)$$

3.4.3 Choix de la taille du bloc et de la fenêtre de recherche

La stratégie de recherche et les dimensions de la fenêtre de recherche et du bloc influencent le coût de calcul. La détermination de leurs dimensions optimales respectives nécessite toujours un compromis. Une grande fenêtre de recherche implique des calculs longs et un risque élevé de confusion du bloc recherché avec un bloc semblable. Mais plus la taille de la fenêtre de recherche décroît, plus le déplacement maximal estimé diminue. De la même manière, quand les blocs sont grands, le coût de calcul est grand, la résolution spatiale est faible et un bloc peut contenir des pixels appartenant à des objets différents. Inversement, un bloc trop petit peut ne pas contenir suffisamment d'information discriminante. Ce dernier problème apparaît, en particulier, dans les zones homogènes où la mise en correspondance n'est pas fiable, pouvant devenir aléatoire.

En plus des problèmes déjà mentionnés (temps de calcul, résolution spatiale et fiabilité de la mise en correspondance) on peut rappeler les limites de l'hypothèse selon laquelle tous les pixels d'un bloc ont le même vecteur déplacement. La méthode suppose des mouvements de translation ou localement assimilables (figure3.2.b), mais cette hypothèse n'est pas respectée, par exemple dans le cas d'une rotation ou d'une homothétie rapide par rapport à la fréquence temporelle d'échantillonnage. Elle n'est pas respectée non plus à la frontière aux bords des objets animés de mouvement différent. Dans une chaine de compression MPEG, ce type de problèmes est réglé au niveau du module sélection du mode de codage.

3.4.4 Algorithmes de parcours du voisinage

Il existe de nombreuses méthodes de block-Matching cherchant à optimiser l'efficacité et la rapidité de l'algorithme [37]. Les principales méthodes sont :

- Recherche exhaustive (Full Search Algorithm) : parcours de l'ensemble de la fenêtre de recherche, algorithme optimal
- Recherche multi résolution a trois étapes (TSS :ThreeStep Search Algorithm) : recherche du minimum global du critère de comparaison utilisé .
- Recherche logarithmique à deux dimensions
- Recherche orthogonale

*3.4.4.1 Algorithme de recherche exhaustive (***FS. Fullsearch***) :***

Fig.3.5 : Recherche exhaustive

Cet algorithme est le plus simple mais aussi le plus coûteux en calcul puisque toutes les comparaisons possibles entre blocs sont réalisées, la figure 3.6 donne une illustration sur la fonctionnement de cet algorithme.
Le coût en calcul de cet algorithme est fonction de la taille de l'espace de recherche (fenêtre de recherche). Le bloc retourné par l'algorithme sera celui qui minimise le critère de comparaison.
La recherche exhaustive étant trop couteuse, plusieurs algorithmes dits rapides ont vu
le jour. Tous ces algorithmes rapides sont explicitement ou implicitement basés sur la proposition suivante.

Le critère de comparaison diminue de façon monotone vers le minimum global de la fenêtre Ainsi, plutôt que de parcourir tous les blocs, le but de ces algorithmes est de parcourir la fenêtre en se rapprochant pas à pas du minimum global.

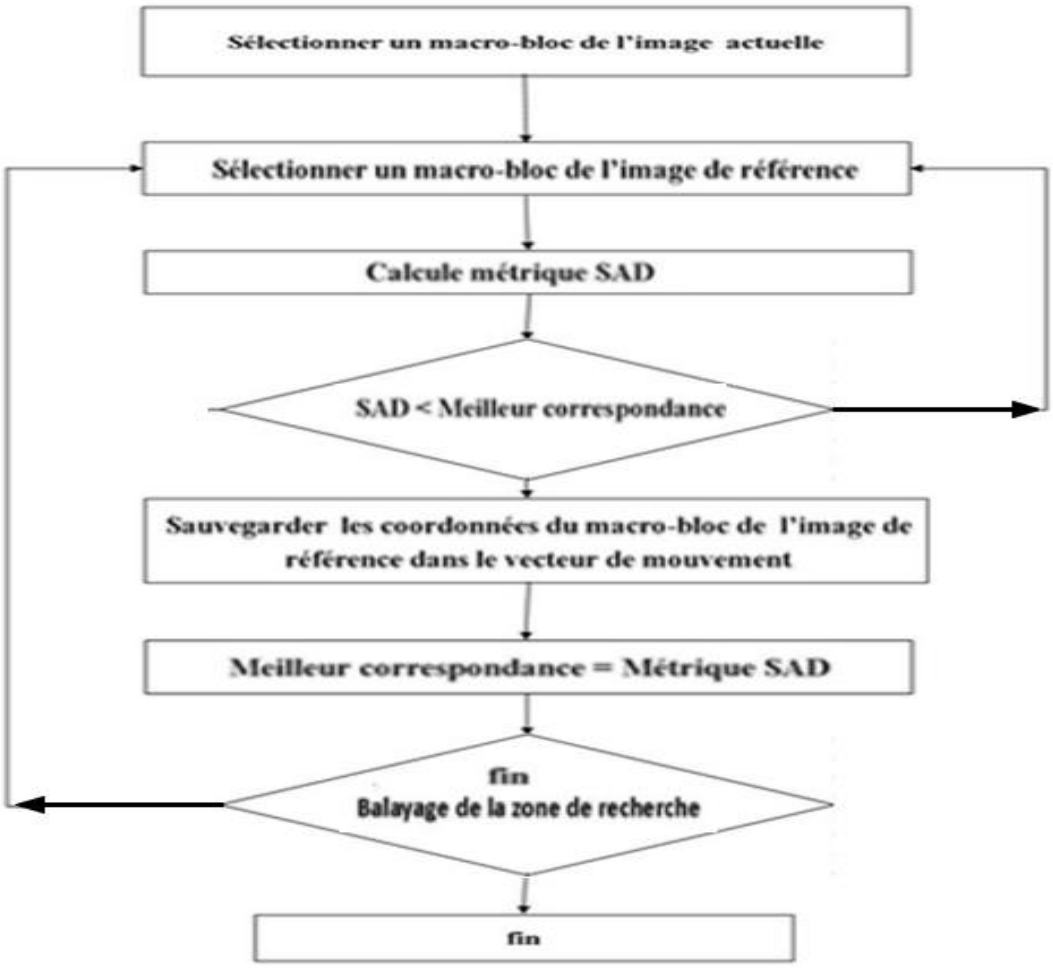


Fig.3.6. Organigramme de l'algorithme de recherche exhaustive

3.4.4.2 Recherche en trois pas (3SS ou TSS : ThreeStep Search)

L'algorithme 3SS, proposé en 1981 par Koga, est basé sur une approche par raffinements successifs avec un pas à décroissance logarithmique. A chaque itération le sous-ensemble choisi est un "carré" de pas égal au pas précédent divisé par 2. Le pas initial est [d/2]. La figure 3.7 illustre la recherche en 3 pas pour des vecteurs mouvements d'amplitude +/- 7 pixels [38].

La figure 3.7 montre les trois étapes nécessaires à l'algorithme pour retourner un résultat. Dans un premier temps le bloc centré et les huit blocs définis à une distance de quatre pixels du pixel central sont testés. Le bloc qui minimise le critère est conservé et est fixé comme étant la nouvelle origine. La deuxième étape consiste à réaliser la même opération mais en ne considérant que les huit points situés à une distance de deux pixels. On notera que le pixel situé à la nouvelle origine n'est pas testé car la valeur du critère avec ce bloc a déjà été calculée. Ainsi, il se peut que parmi les huit points considérés, aucun ne minimise le critère par rapport à l'origine. Enfin, nous réitérons le même procédé lors de la troisième étape mais avec un déplacement de un pixel.

En ce qui concerne le nombre de points visités. Pour une fenêtre de recherche de [-7;7] x [-7;7] pixels, là où l'algorithme exhaustif teste $15^2 = 225$ points, l'algorithme Three Step n'en parcourt que 25. Par contre cet algorithme permet seulement de parcourir une fenêtre de recherche de [-7;7] x [-7;7] pixels alors que l'algorithme exhaustif n'a aucune contrainte algorithmique par rapport à la fenêtre de recherche.

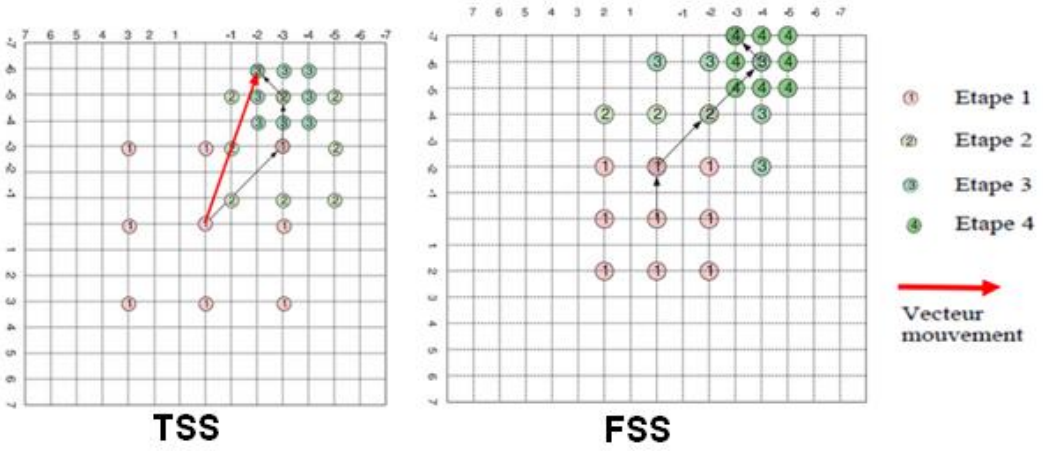


Fig. 3.7 : Recherche en TSS/ FSS [38].

3.4.4.3 Recherche en quatre pas (4SS ou FSS : Four Step Search) :

L'algorithme FSS utilise une zone de recherche avec 9 pixels sur une fenêtre de 5*5. Ci-après les différentes étapes de la recherche :

- **Étape 1**: Un point du minimum SAD est trouvé d'un modèle de 9 points de contrôle sur une fenêtre du 5x5 située au centre de la zone de recherche. Si le point du minimum SAD est trouvé au centre de la fenêtre de recherche, passez à l'étape 4 ; autrement passez à l'étape 2.
- **Etape 2** : La stratégie de recherche est la même que l'étape 1, mais finalement, on passe à l'étape 3.
- **Etape 3** : La stratégie de recherche est la même que l'étape 1, mais finalement, on passe à l'étape4.
- **Etape 4** : La fenêtre de recherche est réduite à 3x3 comme indiqué dans (la Figure 3.7) et la direction du vecteur global de mouvement est considérée comme le point du minimum SAD parmi ces 9 points de recherche.

3.4.4.4 Recherche logarithmique à deux dimensions

Proposé par Jain en 1981, cet algorithme utilise comme sous-ensemble de recherche le "plus". Le pas initial Ce pas est divisé par 2 lorsque le minimum de distorsion courant est au centre du "plus" ou lorsque le bord de la fenêtre de recherche a été atteint. Pour la dernière itération (pas égal à 1), le sous-ensemble est un "carré". La figure 3.8 illustre une réalisation [38].

La recherche 2D-logarithmique met en œuvre une stratégie de recherche en croix (+) à chaque itération. Le pas initial est égal à t [p/4]. Il est réduit de moitié quand le point correspondant à l'optimum trouvé est le centre ou situé au bord de la fenêtre de recherche. Dans le cas contraire, le pas ne change pas. Quand le pas atteint la valeur 1, les 8 points immédiatement voisins du pixel central sont testés. Sur la figure 3.8 on présente deux situations différentes: la situation en haut nécessite:

$n = 5 + 3 + 3 + 8 = 19$, points de calcul et la situation en bas et à droite nécessite: n=5+3+2+3+2+8=23 points de calcul.

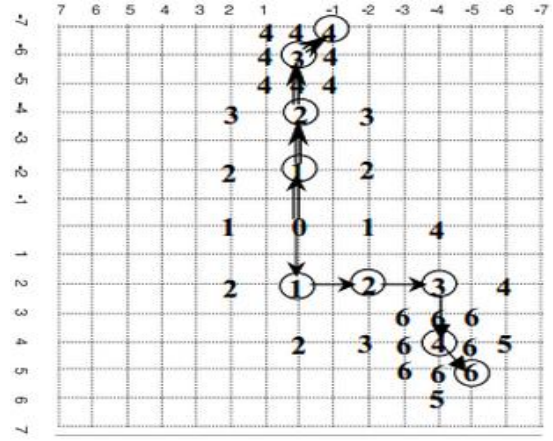


Fig. 3.8: Le principe de recherche 2D-logarithmique [38].

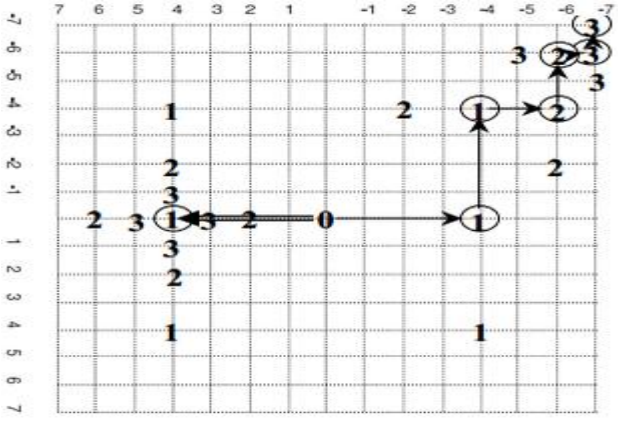


Fig. 3.9 : Principe de recherche orthogonale [38]

3.4.4.5 Recherche orthogonale :

L'algorithme de recherche orthogonale [38] consiste en paires de comparaisons horizontales et verticales avec une décroissance logarithmique du pas. La dimension initiale du pas est $[p/2]$. Le principe de recherche orthogonale est montré sur la figure 3.9.

Une itération a deux étapes. Dans la première étape, trois calculs du critère de ressemblance sont faits en trois pixels horizontaux, 0 et 1. L'optimum dans la direction horizontale devient le centre de la deuxième étape (recherche dans la direction verticale), le pas restant inchangé. Le pas est alors diminué de moitié en répétant la même stratégie de recherche à l'itération suivante. L'algorithme s'arrête quand le pas devient égal à 1.

Pour $p = 7$, dans le cas présenté sur la figure 3.9 en haut à droite, l'algorithme de recherche orthogonale nécessite d'estimer le critère en : $n = 3 + 2 + 2 + 2 + 2 + 2 = 13$ points de calcul. Dans le cas général: $n = 1 + 4 \cdot log2(p + 1)$ points de calcul sont nécessaires.

3.4.4.6 Amélioration des algorithmes de recherche

D'autres méthodes de parcours ont été proposées, on peut citer

- Méthode de recherche sur une grille en diamant (Diamond Search Algorithm) : modeles de recherche en diamant
- Méthode de recherche sur une grille hexagonale (Hexagon-Based Search Algorithm) : modeles de recherche de forme hexagonale

Une étude plus détaillée des techniques de parcours de voisinage peut être trouvé dans notre mémoire de Master.

D'une façon générale, afin de réduire le nombre d'opérations nécessaires à un algorithme de recherche pour apparier deux blocs, plusieurs techniques peuvent être utilisées :

- Limiter la recherche à un nombre restreint de candidats. La recherche n'est alors plus exhaustive, l'idée générale est de réduire le nombre de candidats en supposant que la SAD varie de façon monotone dans la fenêtre de recherche. Il est alors possible de suivre une direction de descente privilégiée (algorithmes prédictifs et récursifs),
- Arrêter les calculs le plus tôt possible (éviter les opérations inutiles).
  Le calcul peut être arrêté dès que les valeurs déjà accumulées dépassent le minimum courant. Ainsi les boucles *i* et *j* peuvent être arrêtées prématurément si SAD (j, i) > minimum. Comme la SAD peut augmenter très vite lorsque l'on s'éloigne de l'optimum, de nombreux calculs sont évités (élimination de candidats)
- Simplifier les calculs effectués : Il est possible de simplifier le critère d'erreur afin de réduire les besoins en ressource lors de l'implémentation. Par exemple le nombre de bits peut être réduit pour abaisser le coût d'une implantation câblée ou bien le nombre de pixels pris en compte dans le critère d'erreur afin de réduire le nombre de calculs.

Les méthodes de mise en correspondance de blocs sont les plus utilisées, grâce à leur simplicité d'implémentation software et hardware [36], [37], [38]. Ce type de méthode s'est imposé dans les standards de compression d'images, comme H.261 ou MPEG-1, MPEG-2 et MPEG-4.

Conclusion

Dans ce chapitre, nous avons présenté plusieurs techniques d'estimation de mouvement. L'estimation peut se faire selon deux approches ;

1. Calculer le mouvement apparent (déplacement/vitesse instantanée) de chaque pixel ce qui conduit á un flux optique dense.
2. Apparier certaines structures spatiales (mise en correspondance) pour chaque couple d'images.

Les méthodes différentielles sont très précises, mais ne permettent d'estimer que de faibles déplacements. Les techniques fréquentielles sont robustes, mais nécessitent une étape supplémentaire de mise en correspondance. Les techniques permettant la mise en correspondance de blocs restent la solution appropriée en termes de complexité, temps calcul, cohérence des résultats pour une application de compression vidéo.

Dans le chapitre suivant, nous détaillerons la conception et l'implémentation de l'architecture d'un estimateur de mouvement qui intègre une nouvelle méthode de recherche.

Chapitre 4 : Une nouvelle méthode de recherche :

Architecture, implémentation et résultats.

Introduction

Dans ce chapitre, nous proposons une nouvelle méthode d'estimation de mouvement qui s'appuie sur l'algorithme de recherche exhaustive (Full Search). Les performances sont comparées avec les méthodes de mise en correspondance de blocs classiques, et une architecture matérielle est proposée. Par la suite nous décrivons la démarche d'implémentation de l'estimateur de mouvement sur une plate-forme FPGA et nous terminons par la présentation des résultats et des tests effectués.

4.1 Description générale de la méthode

Bien que différentes techniques d'estimation de mouvement aient été développées ces dernières années, la recherche est toujours active dans le domaine pour améliorer l'efficacité des algorithmes basés sur la mise en correspondance de blocs (BMA: Blocs Matching Algorithms)

Nous proposons une nouvelle méthode de recherche basée sur une approche en deux phases de l'algorithme de recherche exhaustive (FSBM: Full Search Blocs Matching). Contrairement à la méthode classique de mise en correspondance de blocs (FSBM), qui compare le bloc de référence avec tous les blocs possibles -au pixel près- de la zone de recherche, notre méthode nommée TSFS (Two-step Full Search), adopte une autre approche par raffinements successifs en deux phases:

- Phase1 : Recherche globale rapide en évitant le chevauchement vertical et horizontal entre les différents blocs candidats.
- Phase 2 : Recherche exhaustive sur une petite zone de recherche autour du meilleur candidat trouvé dans la première phase.

La fig

ure 4.1.a illustre le processus d'estimation du vecteur de mouvement selon cette approche et la figure 4.1.b montre la répartition des blocs autour d'un bloc de référence pour les deux méthodes FS et TSFS avec une zone de recherche de 12x12 points et un bloc de 4x4 pixels.

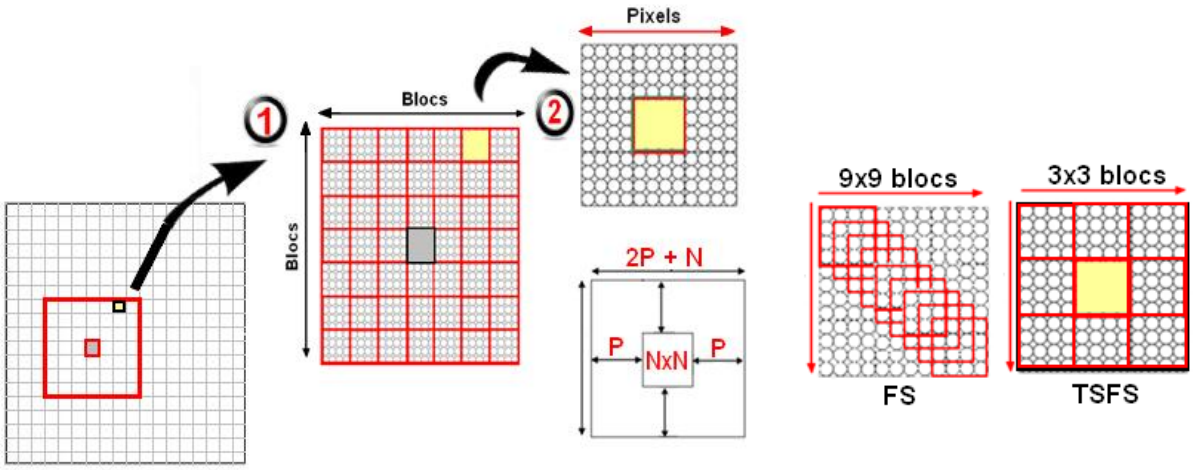


Fig.4.1 (a) Processus d'estimation du vecteur de mouvement du TSFS,
-(b) Répartition des blocs autour d'un bloc de référence pour le FS et le TSFS

Certes, la recherche effectuée lors de la première phase par le TSFS n'est pas exhaustive, donc le résultat sera moins précis que le FS. En d'autres termes, cela signifie qu'au lieu de choisir le bloc optimal, l'algorithme sélectionne un bloc dans son voisinage immédiat (exemple figure 4.2). On peut espérer que localement les différences entre blocs sont minimes ce qui a pour effet de minimiser l'impact visuel de l'erreur de la recherche rapide. Mais pour rectifier le tir et corriger le résultat, il vaut mieux effectuer –dans une seconde phase- une recherche exhaustive (au pixel près) sur une petite zone autour du meilleur candidat trouvé dans la première phase.

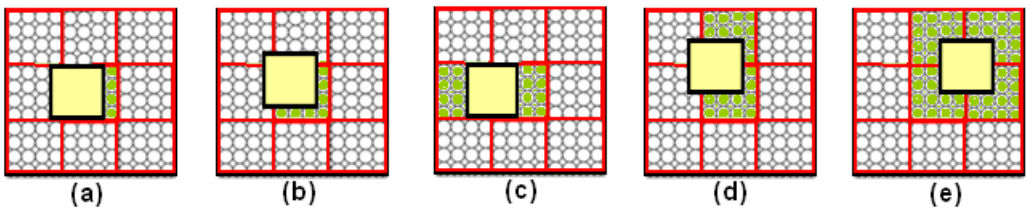


Fig. 4.2 - illustration de quelques cas d'erreurs de la recherche rapide dues au balayage non exhaustif de la zone de recherche

4.2 Complexité de l'estimateur TSFS

Comme pour le FS, plusieurs paramètres influent sur la performance de l'algorithme d'estimation de mouvement, notamment le rayon de recherche « p » et le nombre de macro-blocs comparés au cours du processus de recherche du meilleur correspondant.

	Recherche exhaustive FS	**Recherche Rapide puis exhaustive TSFS**
Nombre de candidats (1 phase)	$(2P+1)^2$	$\left(2\,\frac{P}{N}+1\right)^2$
Nombre de valeurs SAD calculés	$(2P+1)^2$	$\left(2\,\frac{P}{N}+1\right)^2+(2P''+1)^2$

TAB. 4.1 - Complexité des deux méthodes : FS et TSFS

P = rayon de recherche rapide (1er phase) , P'' rayon de la recherche exhaustive (2 phase) avec P'' <P

4.3 Paramètres de l'estimateur TSFS

Des travaux ont été effectués montrant que dans le cas général, une zone de recherche de rayon p = 15 pixels offre de bons résultats avec une bonne qualité visuelle (PSNR élevé) [39],

Nous avons mené des simulations sur la séquence d'images «Tennis player » et « Caltrain » (voir tableau 4.1) ; nous avons trouvé que le PSNR calculé pour les valeurs p=12, p = 15 et p = 20 sont pratiquement presque identiques, mais il y a une différence notable sur les temps de calcul

	Sequence d'image "Tennis player"		**Sequence d'images "Caltrain"**	
Rayon de recherché "p"	**PSNR (dB)**	**Temps de calcul (secondes)**	**PSNR (dB)**	**Temps de calcul (secondes)**
4 pixels	31.4900 dB	4.063	25.6012 dB	4.917
7 pixels	32.5540 dB	10.309	25.6714 dB	10.576
12 pixels	33.4610 dB	27.942	25.7941 dB	27.784
15 pixels	33.6165 dB	43.147	25.8404 dB	43.024
20 pixels	33.6890 dB	72.024	25.9270 dB	71.918

TAB. 4.1 - Variation de PSNR et du temps de calcul en fonction de rayon de recherche

Sur la base de ces simulations fixe la valeur qui détermine le rayon de recherche (et par conséquent la taille maximale des vecteurs de déplacements) à p= 12. On estime que c'est bon compromis entre une bonne qualité d'image (PSNR) et la rapidité de l'exécution.

Quant à la taille du bloc, le standard MPEG H264 utilise des blocs 16x16, mais peut travailler sur des partitions avec des blocs à taille variable. Un bloc 16x16 peut être décomposé en 6 tailles de blocs différentes (16×8, 8×16, 8×8, 8×4, 4×8, 4×4). Les blocs de petites tailles permettent une segmentation très précise de zones de mouvement. Sur ce, on a décidé de travailler avec des blocs 4x4 pixels. Cela nous permet d'avoir une représentation plus fine du mouvement, mais aussi de limiter les effets des erreurs de la recherche rapide

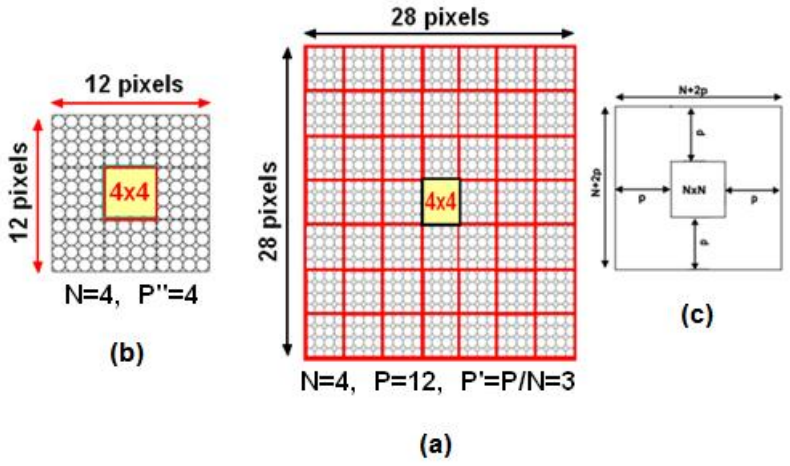


Fig. 4.3 – Paramètres de l'estimateur TSFS.

	Recherche exhaustive FS	**Recherche Rapide puis exhaustive (TSFS)**
Nombre de SAD calculés	$(2P+1)^2 = (25)^2 = 625$	$\left(2\,\frac{P}{N}+1\right)^2 + (2\,P'+1)^2$= 49+25=74

4.4 Les performances obtenues

Plusieurs séquences vidéo ont été encodées en utilisant alternativement les estimateurs de mouvement de référence (FS) et l'estimateur TSFS. Les résultats sont présentés dans le tableau ci- après

	Recherche exhaustive (FS)		Recherche Rapide puis exhaustive (TSFS)		Performances TSFS/FS	
Séquence vidéo	PSNR (dB)	Temps de calcul (seconde)	PSNR (dB)	Temps de calcul (seconde)	PSNR (%)	Temps %
Tennis player	33.4610	27.942	32.5055	5.292	97,1%	16%
Caltrain	25.7941	27.784	22.8623	4.203	88,6%	18%
Akiyo	28.4850	33.805	28.4066	5.303	99,7%	19%

TAB. 4.3 – Comparaison des performances de deux estimateurs FS et TSFS

On remarque qu'en termes de qualité (mesurée par le PSNR), les deux algorithmes ont des performances proches, avec un léger avantage pour FS pour la séquence « Caltrain » qui présente des mouvements complexes. Cela confirme nos suppositions, la recherche rapide, bien que la réduite, conduit à de bons résultats.

Quant aux temps d'exécution, en comparaison avec le FS, le TSFS est de loin le plus rapide, (voir figure 4.4). Cela était prévisible, la charge de calcul inhérente à la recherche exhaustive allonge le temps d'exécution du FS.

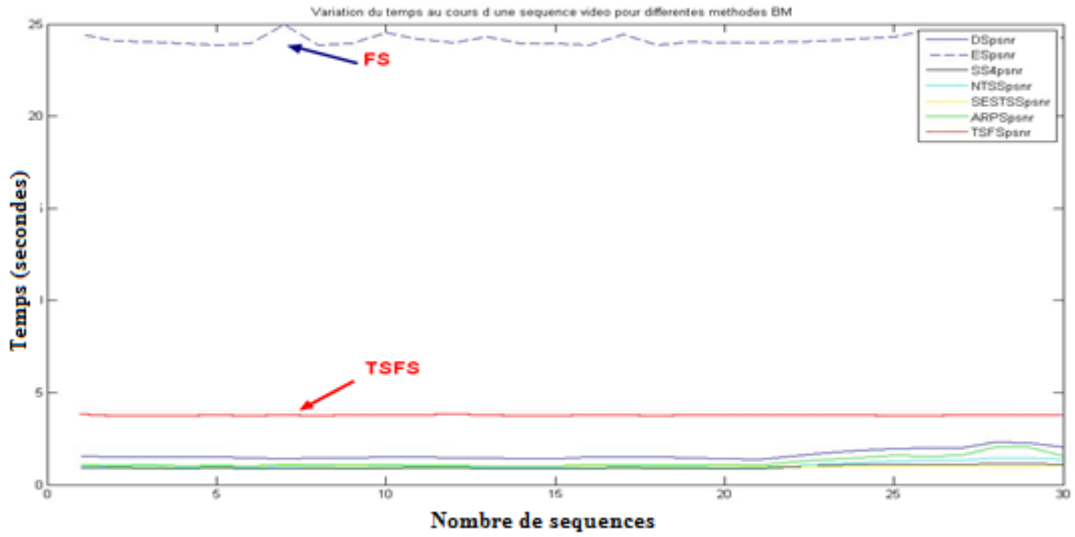


Fig. 4.4 –Variation du PSNR d'une séquence vidéo pour différente méthodes BM

Nous avons mené une série de simulations pour voir comment se situe notre méthode par rapport aux autres méthodes (TSS, FSS, etc). Les résultats sont présentés sur les figures 4.4 et

figure 4.5. En termes de qualité, notre méthode se classe en deuxième position, juste au-dessous du FS. Mais les autres méthodes sont en moyenne deux fois plus rapide que le TSFS.

Ainsi, même si le TSFS est moins rapide que les autres méthodes, il réduit le temps de calcul de 80% par rapport au FS. Ainsi son faible coût de calcul et la bonne qualité de l'estimation justifie son utilisation.

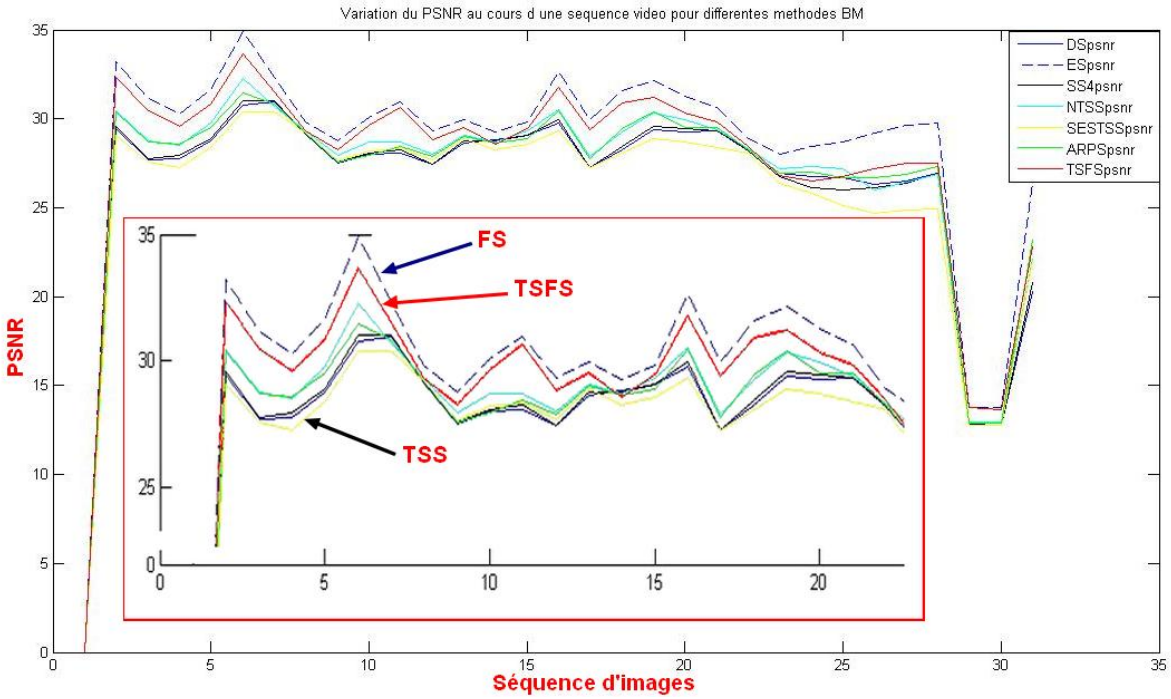


Fig. 4.5 – évaluation du PSNR pour différentes méthodes BM

4.5 Architecture du module de l'estimation de mouvement.

L'unité de calcul SAD de base pour trouver la métrique SAD minimal est donnée par la figure 4.6

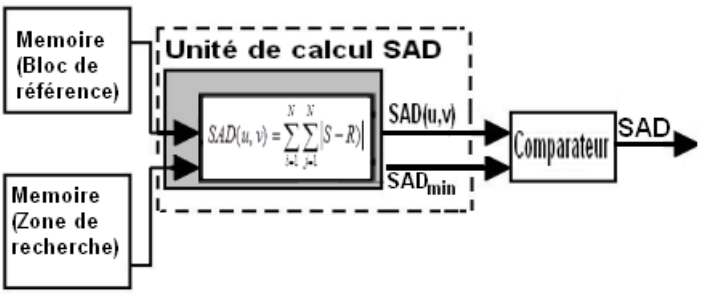


Fig. 4.6 - schéma d'une unité de calcul de la somme des différences absolue SAD.

Cependant afin de gérer les transferts mémoire et synchroniser les calculs on aura besoin d'autres unités. Une architecture plus détaillée et illustrée sur la figure 4.7

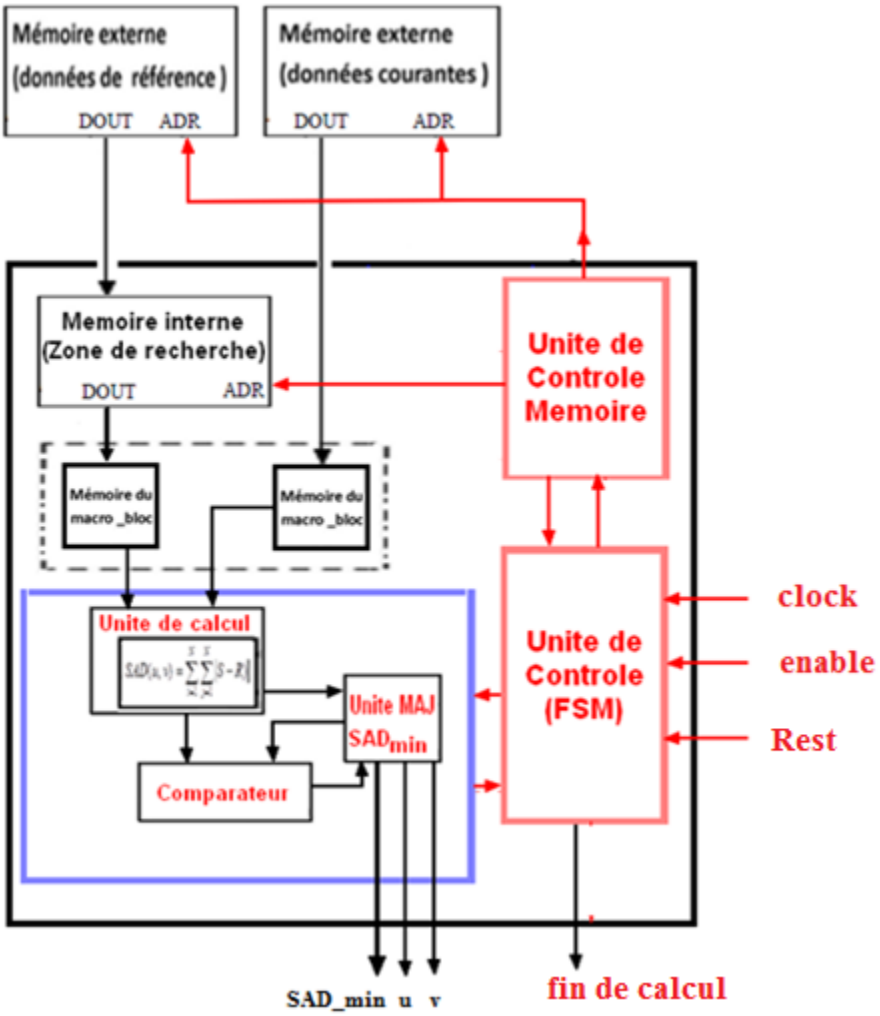


Fig. 4.7 - Architecture du module d'estimation de mouvement.

Le schéma principe de se compose de plusieurs modules interconnectés :

Une mémoire interne : pour contenir l'ensemble des données de références de la zone de recherche.

Mémoires de macro-bloc : mémoires locales destinées à contenir les macros bloc de référence et sources.

Unité de calcul : chargé de calculer la somme des différences absolue (SAD)

Un comparateur : pour comparer la valeur donnée par l'unité de calcul avec la meilleure correspondance trouvée au cours des calculs antérieurs.

Unité MAJ SAD_{min} : chargée de mettre à jour la valeur de SAD_{min} ainsi que les indices (u,v) lui correspondantes.

Une unité de contrôle FSM : gère l'enchaînement des différentes étapes du processus de calcul en basculant des différents états de fonctionnement (*Initialisation, Phase 1 de TSFS, Phase 2 de TSFS, Fin du traitement*).

Unité de contrôle mémoire : fonctionne intermédiairement entre l'unité de contrôle et les blocs mémoire afin de :

- Sélectionne le macro-bloc dans la mémoire de zone de recherche, en incrémentant les compteurs d'adresse mémoire.
- synchroniser le transfert des données dans les mémoires macro-bloc candidats et référence.

Fonctionnement de l'unité de control :

L'unité de control s'occupe de des différentes unités et de la gestion des différentes phases processus de calcul. Elle fonctionne selon une machine à états finis ayant 4 états illustrés dans la figure 4.8 :

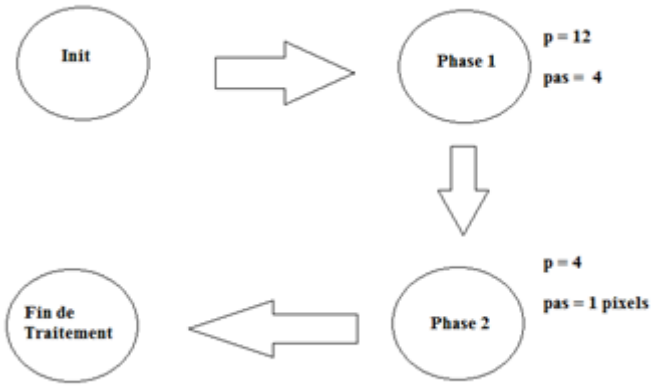


Fig. 4.8 – séquencement d'état de l'unité de contrôle

ETAT 1 : Initialisation

- Charger les éléments de la zone de recherche dans la mémoire locale.
- Charger le macro-bloc source
- Initialiser la valeur SAD_{min} par la plus grande valeur possible de la métrique SAD_{min} = 4x4x255 = 4080.

ETAT 2 : Phase 1

Dans cette états l'unité de contrôle fixe les paramètres de balayage de la première phase de la méthode TSFS (la recherche global et rapide), en fixant un rayon de recherche p=12 et un pas de balayage de 4 pixels.

ETAT 3 : Phase 2

Cet état correspond à la seconde phase de la méthode FSTS (recherche exhaustive), et fixe le rayon de recherche p' = 4 pixels avec un pas plus précis de 1pixel.

ETAT 4 : Fin de traitement

Lorsque le module d'estimation de mouvement est sur cet état, les résultats sont retourné indice du meilleur macro-bloc correspondant (u_{min} ,v_{min}) ainsi que la valeur SAD_{min}.

4.6 Implémentation du module :

Avant de présenter les travaux réalisés, nous commençons par une présentation de la plate-forme matérielle RC203E mise à notre disposition, de son environnement logiciel et de flot d'implémentation qu'elle supporte.

4.6.1 Présentation de la plateforme d'implémentation

Le kit de développement RC203E de Celoxica fournit une excellente plateforme pour l'implémentation matérielle. Elle est basée un FPGA Virtex-II de xilinx conçu pour des applications hautes performances. La figure 4.9 donne le diagramme simplifié de la carte de développement.

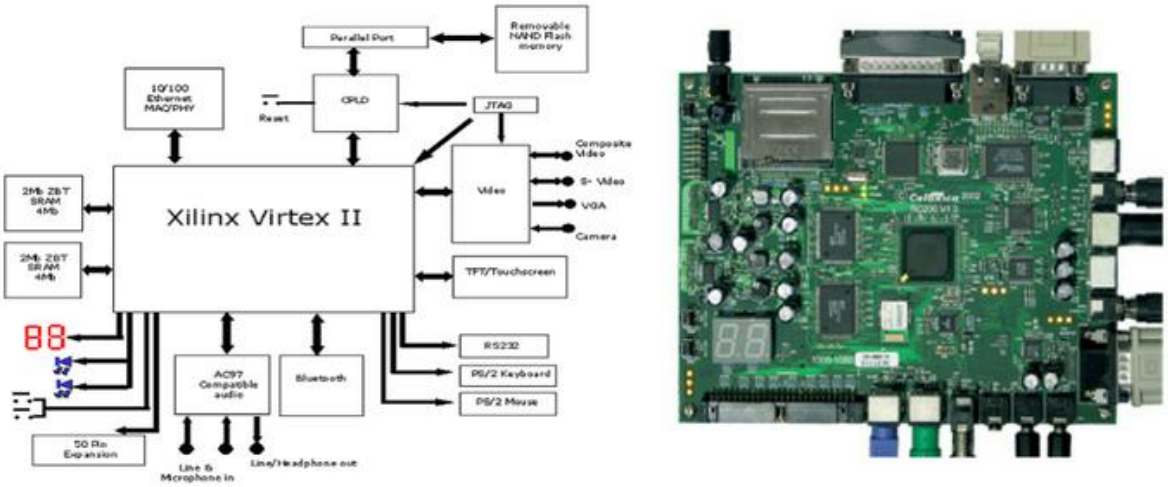


Fig 4.9 -Le kit de développement RC203E de Celoxica

L'environnement matériel

L'élément central de la carte est le circuit FPFA **XC2V3000-4FG456C ou** 4FG672.de la famille Virtex-II de Xilinx, doté de 3000 portes logiques, réparties sur 14 K tranches du circuit (slices). Le FPGA intègre une mémoire distribuée dynamique (SDRAM) de 448 Kbits et 96 blocs RAM (BRAM) de 18 Kbits chacun. Il comporte 96 multiplieurs embarqués exécutant des opérations sur des mots de 18 bits.

La carte comporte un circuit programmable CPLD pour la configuration/reconfiguration du FPGA à travers le port parallèle et une interface JTAG pour la programmation de l'ISP PROM. Elle offre aussi de multiples interfaces d'entrées et de sorties. Elle possède une interface RS232, un connecteur clavier PS/2 et un connecteur souris, et un écran TFT de 640x480 pixels. Deux LEDs et deux afficheurs 7 segments à cathode commune sont présents sur la carte, et peuvent être utilisés durant la phase de test et de mises au point des programmes (debug).

L'environnement logiciel

Le kit de développement RC203E est livré avec un environnement de d´développement basé sur le langage Handel-C (voir Annexe) : Le DK/PDK design suite de Celoxica. La dernière version (DK 5) est commercialisée sous le sigle Agility.

La suite DK est un outil de design ESL (Electronic System Level) , c'est `à dire qu'il fait abstraction du bas niveau au profit du développement d'architectures et d'algorithmes. En effet le langage Handel-C avec une syntaxe proche du langage C, est une solution alternative à la programmation bas niveau (VHDL) du circuit FPGA. Une présentation plus détaillée peut être trouve aux annexes A1 et A2.

4.6.2 Présentation générale du flot de conception

Le flot de conception classique représenté par la figure 4.10, admet une entrée sous la forme d'une spécification fonctionnelle/contraintes de l'application, et à chaque niveau, le concepteur s'intéresse à la résolution d'un problème lié à la spécification de l'application, la définition de l'architecture ou à l'implémentation du design sur circuit spécifique.

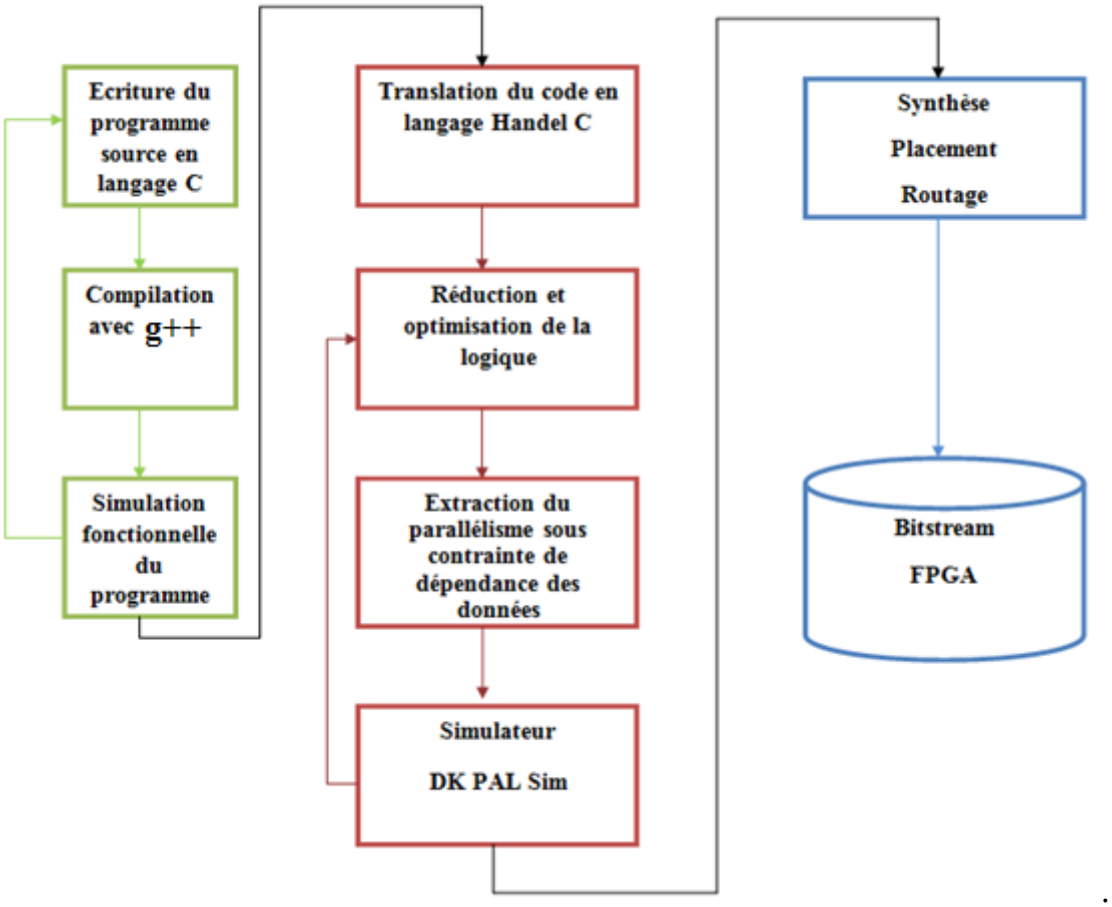


Fig. 4.10 - Schéma général du flot de conception.

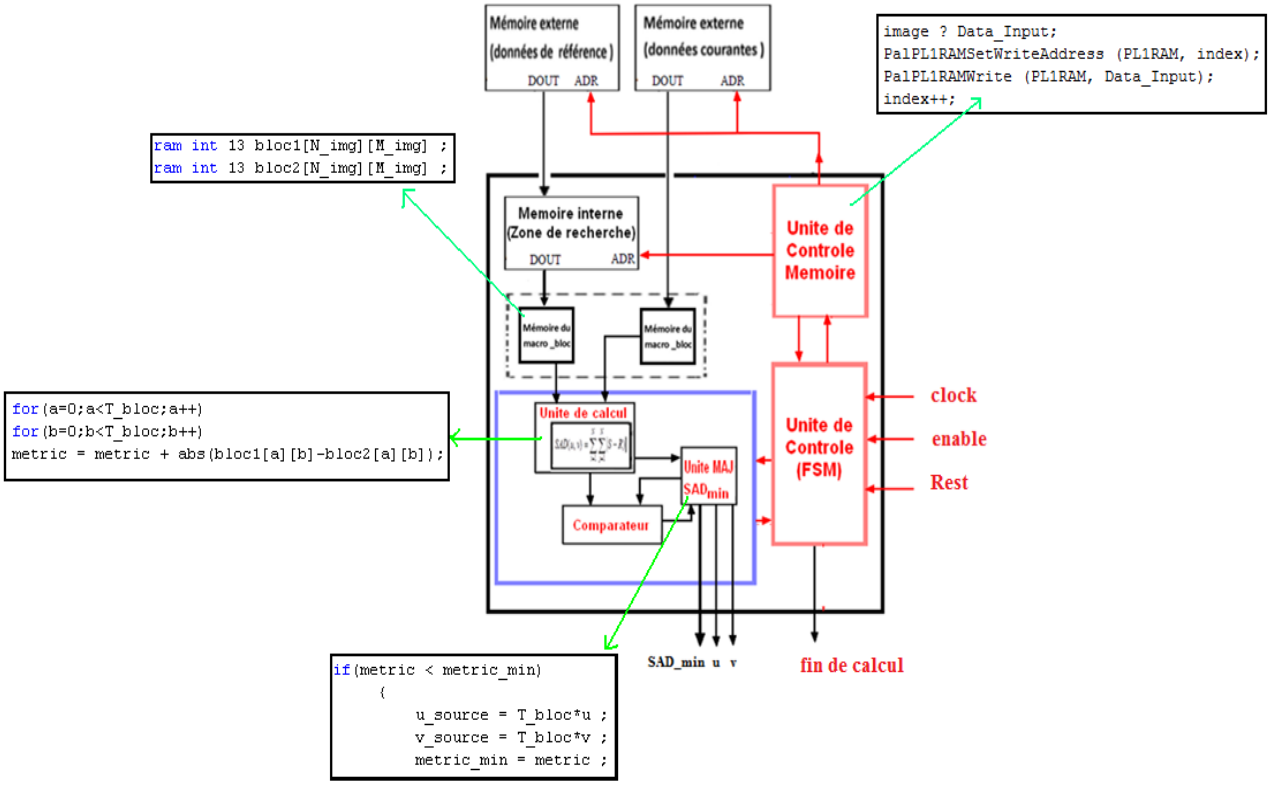
4.6.3 Travaux réalisés sur la plateforme de développement

Une fois familiarisés avec l'outil DK 5 et le langage Handle-C, on a programmé l'algorithme de correspondance de blocs « TSFS » avec le langage Handle-C.

Apres la compilation avec l'outil DK5 puis synthèse, placement et routage on a obtenu La ressource hardware utilisé pour l'estimation d'un macro-bloc après placement et routage sous XilinxVirtex II XC2V3000-4FG456C sont représenté sur le tableau 4.5

FPGA device	Virtex II XC2V3000-4FG456C	
Type ressources	Utilisé	Taux d'utilisation
Slices flip flop	860 de 28672	3 %
4 input LUTs	1300 de 28672	4 %
Slices occupée	1407 de 14336	9 %
Registres	64	
Bonded IOBs	60 de 484	12 %
Bloc RAMB16s	3 de 96	3 %
GCLK	1 de 16	6 %

TAB. 4.5 - Ressource hardware pour l'estimation d'un macro-bloc



La procédure de vérification est présentée sur la Fig. 4.11

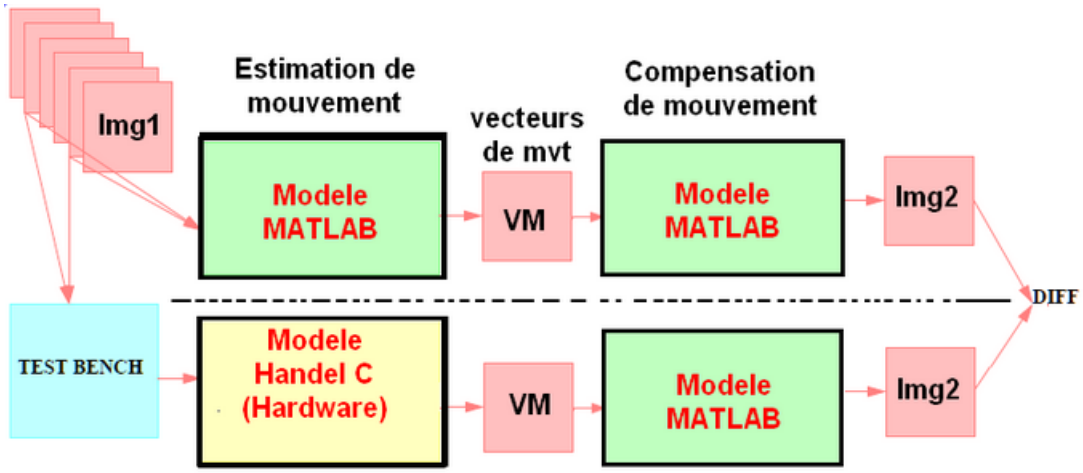


Fig. 4.11 : Procédure de vérification (Matlab et HandelC)

Procédure de vérification et de validation

A l'aide d'un script Matlab, les images couleur RVB (352 x 240 x 3) sont converties en image à niveau de gris (352 x 240 x 1). La taille de la donnée élémentaire (pixel) étant de 8bits, tandis que les mémoires de la carte FPGA acceptent des données sur 36 bits. Une

concaténation par 4 octets était nécessaire pour obtenir des mots de 32 bits, afin d'y arriver à une utilisation optimale de la mémoire, comme le montre la figure 4.12.

Après cette étape de préparation, le fichier de données est transmis à la carte en utilisant une liaison série RS232. En mode simulation on utilise les possibilités de communication par fichiers du HandelC (chanin, chanout) et la plateforme virtuale (PAL) texte dont l'en-tête étant le nombre de ligne, de colonnes et d'éléments de la trame d'image

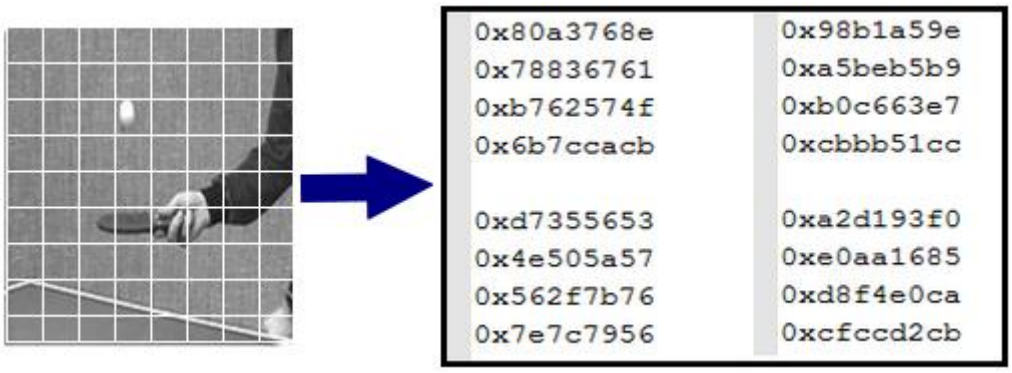
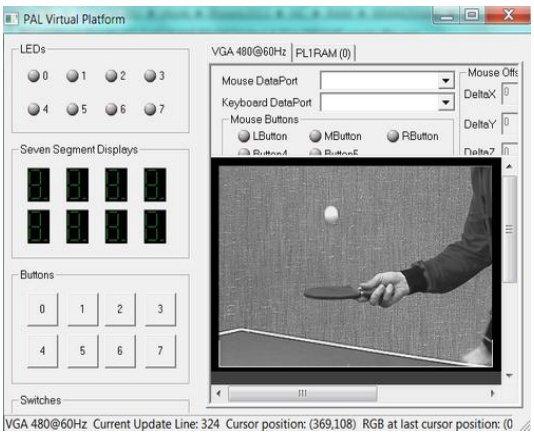


Fig. 4.12–Préparation de la trame d'image « player tennis » pour envoi à la RAM.

Les données sont envoyées à la RAM (figure 4.13) afin que l'unité de traitement puisse y avoir accès, au final, un vecteur de mouvement est généré, il est ensuite exporté sous Matlab est utilisé pour reconstruire l'image originale.

(a)



(b)

Fig. 4.13 (a) Transfert de l'image sur la mémoire de la carte RC203E (b) Vérification avec PAL virtual plateforme

Sur les différentes séries de tests effectués, les résultats obtenus dont très similaires à la simulation logicielle et montrent la validité de notre implémentation.

Conclusion générale et perspectives :

Les travaux présentés dans ce mémoire abordent les problèmes liés au développement d'applications sur plates-formes matérielle à travers l'implémentation d'un module d'estimation de mouvement pour un encodeur MPEG sur un circuit FPGA.

Dans la première partie, nous avons commencé par l'étude des différents modules d'une chaine de compression vidéo, et nous avons procédé à une simulation fonctionnelle et un profilage du code source d'un encodeur MPEG. Il ressort de cette étude que le module d'estimation de mouvement contribue pour beaucoup aux performances comme à la complexité. C'est l'étape la plus gourmande en termes de puissance calculatoire. L'utilisation d'un accélérateur matériel pour exécuter cette tâche est plus que nécessaire.

Nous avons par la suite présenté un état de l'art des techniques et méthodes générales d'estimation du mouvement. Nous avons rappelé les méthodes différentielles et fréquentielle, et nous avons détaillé les méthodes de mise en correspondance de blocs. Ces dernières se sont imposées en pratique ; dans certains standards de compression et restent la solution appropriée en termes de complexité, temps calcul pour une application de compression vidéo.

Dans la deuxième partie, une nouvelle méthode d'estimation du mouvement est présentée.

Cette méthode nommée TSFS (TwoSteps Full Search) s'appuie sur une recherche exhaustive à deux phases :

- Phase1 : Recherche globale rapide en évitant le chevauchement vertical et horizontal entre les différents blocs candidats.
- Phase 2 : Recherche exhaustive sur une petite zone de recherche autour du meilleur candidat trouvé dans la première phase.

Une étude est menée pour déterminer les paramètres de l'estimateur, en particulier, les tailles de différentes zones de recherche. Nous avons montré, à travers des résultats de simulation que le TSFS estime mieux le mouvement que les méthodes de mise en correspondance de blocs classiques, à l'exception de la méthode FS (recherche exhaustive) qui donne toujours la solution optimale. Les autres méthodes sont en moyenne deux fois plus rapides que le TSFS, néanmoins notre méthode réduit le temps de calcul de 80% par rapport au FS. Ainsi son faible coût de calcul et la bonne qualité de l'estimation justifie son utilisation.

Nous avons présenté une architecture matérielle pour l'implémentation de la méthode TSFS, composée de plusieurs unités classiques : unité de calcul, unité mémoire, unité de mise à jour

des valeurs de SAD. Toutes ces unités sont contrôlées par une unité de contrôle dont on a spécifié la machine à état pour gérer le séquencement des différentes phases. Nous avons élaboré le modèle en utilisant le langage HandelC en nous l'avons implémenté sur un FPGA Virtex2 de la plate-forme matérielle RC203E mise à notre disposition. Des tests et des vérifications nous ont permis de valider notre architecture et nos choix.

En ce qui concerne les perspectives, la première que nous pouvons entrevoir est d'élaborer un modèle VHDL de l'architecture proposée. Ceci afin d'optimiser l'utilisation de ressources et de monter en fréquence de fonctionnement.

L'architecture proposée fonctionne en série, mais avec le partitionnement des traitements en blocs, le TSFS se prête bien à une architecture parallèle. De plus l'approche en deux phases permet d'envisager l'application des techniques de pipelining. Il serait intéressant d'explorer ces deux voies afin d'y arriver à une architecture plus performante.

Annexes

Annexe A1 : Le langage Handel-C

Le langage Handel-C est développé par l'université d'Oxford, en Grande-Bretagne. Il permet de générer des circuits, mais n'est pas un langage de description de circuits, c'est un langage séquentiel. Les programmes écrits en Handel-C peuvent d'ailleurs être traduits en C-ANSI `à l'aide d'un compilateur disponible, et exécutés sur des machines séquentielles. Le compilateur produit une netlist qui doit ensuite être appliquée aux circuits reconfigurables à l'aide des outils commerciaux. Pour obtenir un circuit performant, il est nécessaire de paralléliser. Les outils de parallélisassions automatique ne sont pas suffisamment performants, les concepteurs de Handel-C ont donc choisi de laisser le programmeur exprimer les différentes formes de parallélisme : parallélisme de processus, parallélisme d'assignation et parallélisme d'expression.

Handel-C a permis de réaliser des applications impressionnantes, comme des jeux vidéo sans ordinateur, en branchant un moniteur VGA directement en sortie d'un circuit Xilinx qui pilotait `à la fois l'affichage et le jeu. Le même programme, compile sur une Sun Sparc-5 s'exécute 5 fois plus lentement**.**

C'est un langage qui permet de générer des circuits logiques, sans être pour autant un HDL (Hardware Description Langage). Il est plutôt destiné à compiler des algorithmes de haut niveau pour en faire des portes logiques (niveau hardware), sans que l'utilisateur ne se préoccupe de savoir exactement comment fonctionne le calculateur. C'est en fait un langage de programmation plutôt qu'un HDL. En un sens, le Handel-C est au hardware ce qu'un langage de haut niveau conventionnel est `à l'assembleur.

Les programmes en Handel-C

La syntaxe du Handel-C est basée sur celle du C dont elle reprend de nombreuses structures. Aussi les Programmes en Handel-C sont-ils intrinsèquement séquentiels. L'ordre d'écriture des instructions correspond exactement à leur ordre d'exécution. Bien sûr, comme tout langage qui se respecte, le Handel-C contient des structures de contrôle du flot du programme. Par exemple, l'exécution d'une partie de code peut être conditionnée par la valeur d'une expression, ou un bloc de code peut-être répété un certain nombre de fois grâce à une boucle.

Il est important de noter que les circuits logiques générés par le Handel-C sont exactement ceux nécessaire à l'exécution du programme. Il n'y a pas d'interpréteur comme en assembleur ; les portes logiques constituant le circuit Handel-C final sont des instructions assembleur du système Handel-C.

Les programmes parallèles

Puisque le compilateur Handel-C génère du hardware de bas niveau, il est possible d'accroître les performances du système en utilisant le parallélisme. Bien que le séquentiel soit inhérent au Handel-C, on peut (et c'est même parfois primordial) demander au compilateur de créer du hardware pour exécuter des instructions en parallèle. Et il s'agit d'un vrai parallélisme : deux instructions seront exécutées exactement au même instant par deux structures du hardware bien distinctes (contrairement aux ordinateurs classiques qui en donnent seulement l'illusion).

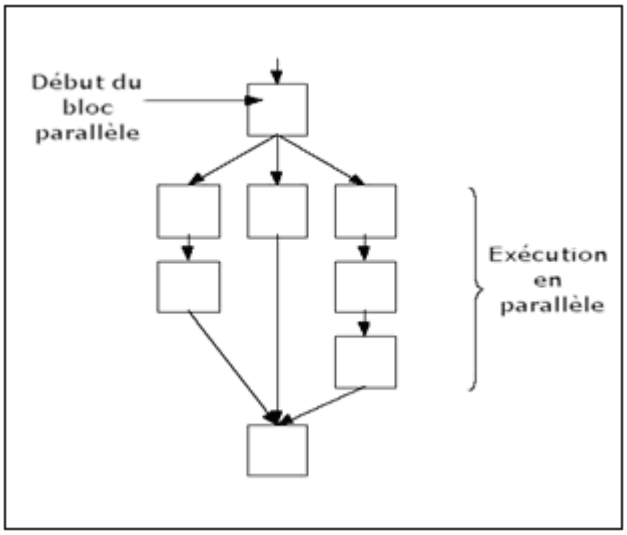


Figure 1 : Séparation et regroupage des branches parallèles

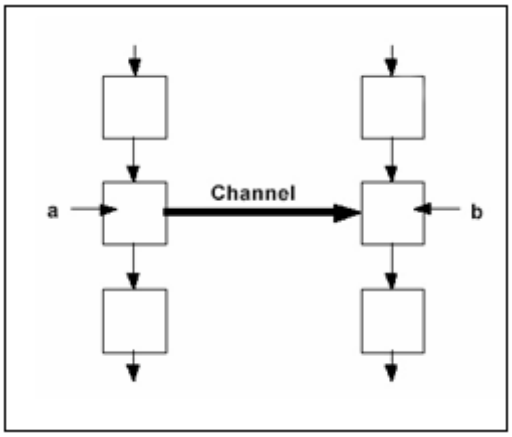


Figure 2: Communication par canal entre branches parallèles.

Si une branche arrive au niveau du canal avant l'autre, elle attend jusqu''à ce que cette dernière soit elle aussi prête pour le transfert de données.

Quand un bloc parallèle est rencontré, chacune de ses branches est exécutée simultanément au départ du bloc en question. Les flots d'exécution parallèles se rejoignent à la fin du bloc quand toutes les branches ont été exécutées. Toute branche se terminant "très rapidement" est obligée d'attendre la plus lente avant de continuer (c'est-à-dire avant que l'on ne sorte du bloc parallèle).

Les canaux de communication

Les canaux constituent le lien entre les branches parallèles. Une des branches fournit des données sur le canal tandis que l'autre la lit. Les canaux servent aussi à synchroniser les différentes branches parallèles car le transfert de données ne peut avoir lieu que si chacune des branches est prête pour le faire : si la branche émettrice n'est pas prête pour la communication alors la réceptrice doit attendre, et vice versa (voir **figure 2**).

Portée et partage des variables

La portée des déclarations est, comme en C traditionnel, basée autour des blocs de code (un bloc étant encadré par des accolades ...). En gros, cela signifie, d'une part, que les variables globales doivent être déclarées en dehors de tout bloc et, d'autre part, qu'une variable déclarée dans un bloc sera valide dans celui-ci ainsi que dans tous ses sous blocs. **La figure2** nous en donne une illustration.

Puisque les structures parallèles sont de simples blocs de code, une variable peut-être valide dans deux branches parallèles : il suffit par exemple d'avoir un bloc principal ou on déclare sa variable et d'avoir deux sous blocs parallèles. Ceci peut entraîner des conflits de ressource si plusieurs sous blocs parallèles accèdent en même temps à la variable en question. La syntaxe du Handel-C stipule qu'une variable ne peut pas être accédée par plus d'une branche parallèle à la fois.

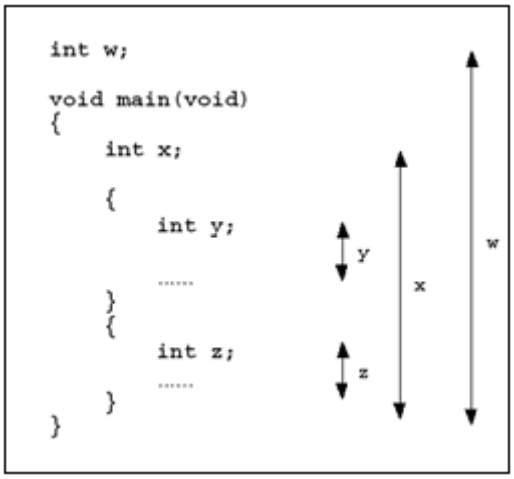


Figure 3 : Domaine de validité des variables

Différences fondamentales entre le Handel-C et le C

Lors du portage d'un programme du C vers le Handel-C, il convient de faire attention, car le Handel-C, bien qu'il soit basé sur la syntaxe du C, comporte tout de même quelques particularités importantes. La liste ci-dessous en pressente les principales :

✓ Les flottants (nombres à virgules) n'existent pas en Handel-C. Il n'y a que des entiers.

✓ Les pointeurs non plus.

✓ Dans un cycle d'horloge, les RAM et les ROM ne peuvent avoir qu'une seule entrée accédée. Ainsi deux branches parallèles ne peuvent pas accéder à une RAM en même temps par exemple.

✓ Lors de l'accès des tableaux, l'index doit être une constante définie explicitement avant la compilation.

Ayant un tableau tab, il est donc impossible d'écrire une boucle sur une variable n pour accéder à tab[n].

Annexe A2 : L'environnement logiciel DK/PDK

Le kit de développement RC203E est livré avec un environnement de d´développement basé sur le langage Handel-C (voir Annexe) : Le DK/PDK design suite de Celoxica. La dernière version (DK 5) est commercialisée sous le sigle Agility.

Fig.1 environnement DK5

La suite DK est un outil de design ESL (Electronic System Level), c'est `à dire qu'il fait abstraction du bas niveau au profit du développement d'architectures et d'algorithmes. En effet le langage Handel-C avec une syntaxe proche du langage C, est une solution alternative à la programmation bas niveau (VHDL) du circuit FPGA

L'environnement DK dispose d'un module de simulation. Il offre des algorithmes d'optimisation adaptes a une implémentation spécifique et convertit le code Handel-C en éléments logiques standards dans un fichier EDIF servant d'entrée à la suite d'outils de Xilinx (ISE). La figure 1 donne une représentation du e flot de conception.

DK offre une couche d'abstraction fournissant les API d'accès aux composants matériels de la plateforme : la librairie PAL (Platform Abstraction Layer).

L'intérêt de cette couche est d´écrire un code identique alors que les cartes sont différentes, l'application Handel-C peut alors être portée sans difficulté sur une autres carte que l'original. Il suffit de bien choisir la configuration adéquate.

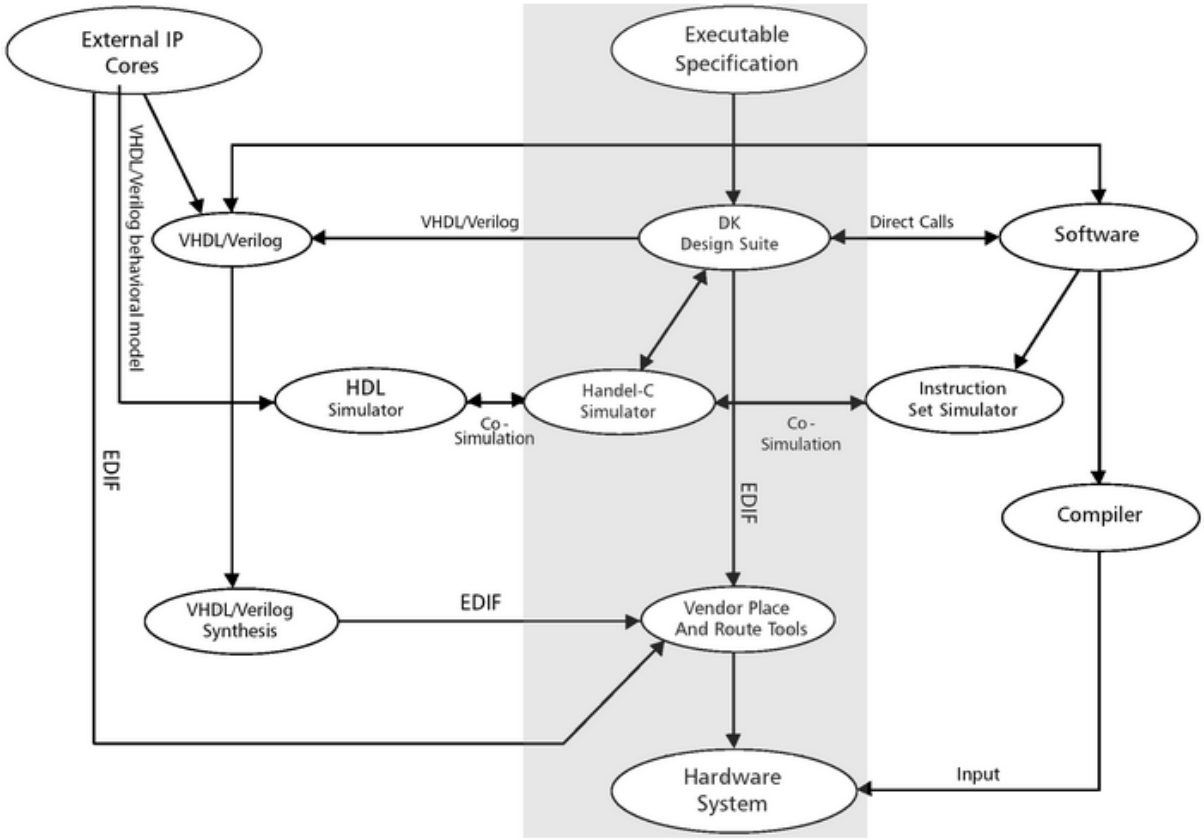


Fig.2

Comme le fait apparaitre la figure 2 la couche PAL établit le lien avec l'ensemble de pilotes spécifiques à la plateforme. Cela permet d'obtenir un niveau d'abstraction facilitant l'utilisation du matériel.

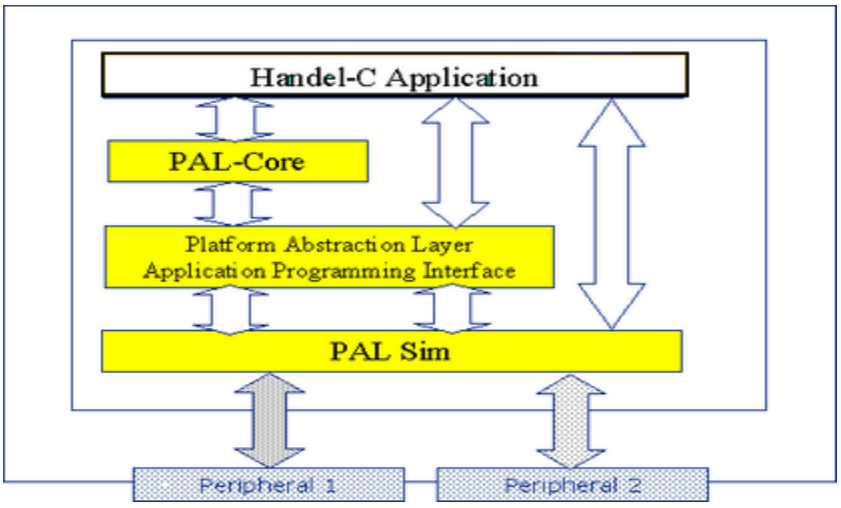


Fig.3

Le développement de l'application dans Handel-C passe par deux étapes : la simulation et la synthèse. La suite DK a un double rôle :

- En tant que simulateur, elle permet de détecter les erreurs syntaxiques et logiques, pour ensuite simuler l'application sur la plateforme de simulation virtuelle (PALSIM)
- En tant que synthétiseur : il va jusqu'à la création d'un fichier EDIF contenant les informations sur la Netlist. Le DK integre des scripts permettant de faire appel à ISE pour générer le « bitsteam » du FPGA cible.

AnnexeA3 : Le FPGA Virtex-II XC2V3000

La figure 1 représente une partie du FPGA. Il est principalement composé d'une matrice de CLBs (Configurable Logic Blocks).

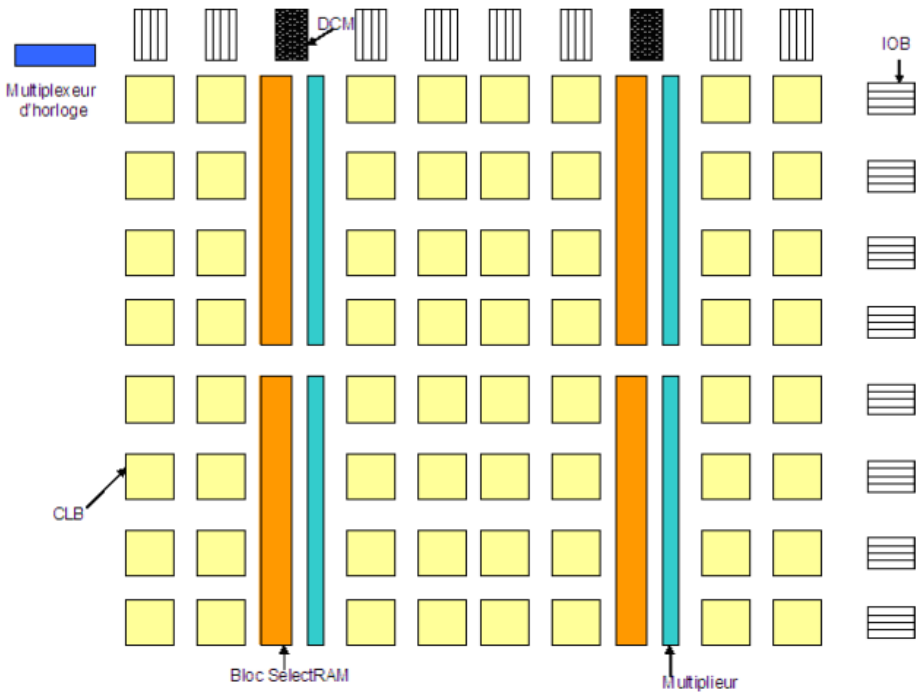


Fig. 1 - Architecture des Virtex-II

Chaque bloc configurable est compose de 4 cléments identiques appelés slices et de deux buffers (figure 2). Les slices sont identiques et contiennent :

- ❖ Deux générateurs de fonctions (composé de LUTs à quatre entrées).
- ❖ Deux éléments de mémorisation (bascules D).
- ❖ Des portes logiques.

- Des multiplexeurs.
- Une chaîne de cascade horizontale (portes OR).

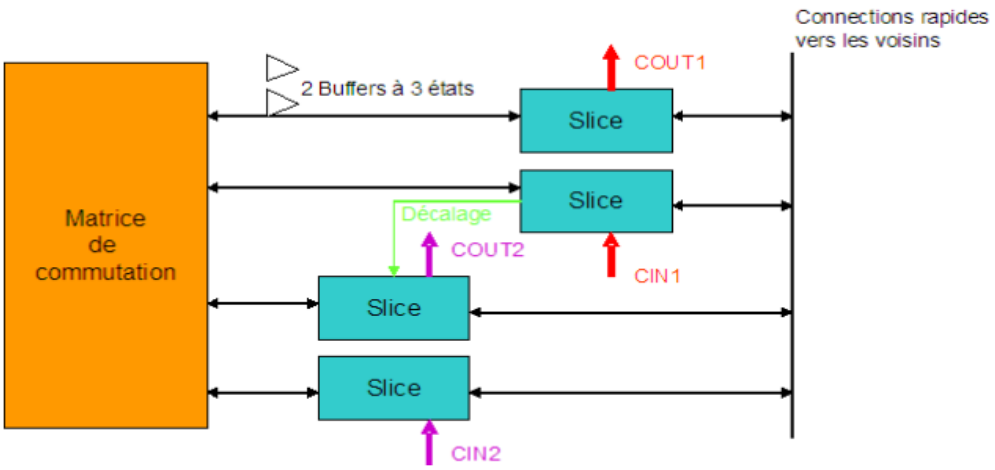


Fig. 2 : Structure d'un bloc logique configurable (CLB)

Les multiplexeurs permettent à chaque slice d'implémenter des fonctions possédant jusqu'à 8 entrées. Les quatre slices sont répartis en deux colonnes, chacune est traversée par une chaine logique indépendante l'une de l'autre (CIN1/COUT1, CIN2/COUT2) et par une chaîne commune de décalage.

Chaque bloc est connecté à une matrice de commutation pour accéder aux ressources de roulage général.

Les IOBs (Input Output Blocks) sont programmables et peuvent prendre le comportement d'entrée, de sortie ou encore bidirectionnel. Deux ou quatre IOBs sont connectés à une matrice de commutation pour accéder aux ressources de routage.

Les blocs mémoires sont de 18kb assignables de 16k * 1bit à 512*36bits. Chaque mémoire possède deux ports synchrones et indépendants l'un de l'autre. A chaque bloc mémoire est associé un multiplieur 18*18 optimisé pour des opérations sur le bloc mémoire. Chaque multiplieur et bloc mémoire est relié à quatre matrices de commutation pour accéder aux ressources de routage.

Les blocs DCM (Digital Clock Multiplexer) et le multiplexeur d'horloge global fournissent des fonctionnalités d'horloges évoluées. Au total il y a 12 DCM sur l'ensemble du FPGA.

La figure 3 indique les caractéristiques précises du Virtex-II XC2V3000, la taille réelle de la matrice de CLB est de 64*56 soit 3584 éléments.

	CLB					
gate	Array	Slices	Max RAM Kbit	Multiplier	SelectRAM Kbits	Max I/O
3M	64*56	14336	448	96	1728	720

Fig. 3 : Caractéristiques du XC2V3000.

Bibliographies

[1] : P.Symes ,Digital Video Compression ,October 2, 2003.

[2] : C. A. Poynton, A Technical Introduction to Digital Video. New York, NY: Wiley, 1996.

[3] : Implantation optimisée d'estimateurs de mouvement pour la compression vidéo sur platesformes hétérogènes multicomposants. Thèse de doctorat de Fabrice URBAN, Décembre 2007.

[4] : Woongshik You ; Joon-Young Jung ; DongJoon Choi ; O-Hyung Kwon ; Oh-Seok Kwon, UHDTV transmission based on broadcasting channel bonding,2013

[5] : N.Laveau ,Mouvement et Vidéo :Estimation, Compression et Filtrage,ENSM paris , 2005

[6] : J.Kronegg , Quantification et PSNR ,University of Geneva ,2003.

[7]: Zhou Wang, Alan Conrad Bovik, Hamid Rahim Sheikh, and Eero R Simoncelli. Image quality assessment : From error visibility to structural similarity. IEEE Transactions on Image Processing, 13(4) :600—612, April 2004.

[8]: C.Poynton ,YUV and luminance considered harmful , New York, NY ,1998.

[9] : Moez KTHIRI ,Etude et Implantation d'Algorithmes de Compression Vidéo optimisés H.264/AVC dans un Environnement Conjoint Matériel et Logiciel,2010.

[10] : F.Koriche ,Introduction a la compression des images,Université Montpellier II, France 2004.

[11]: Ahmed, N. ; Natarajan, T. ; Rao, K.R. "Discrete Cosine Transform",1974.

[12] : P.BAS Compression d'Images Fixes et de Séquences Vidéo cours ENSERG/INPG, Laboratoire des Images et des Signaux de Grenoble ,2004

[13]: revue Accelerating Video Platform Evolution, 2012

[14]: Thomas Wiegand, Gary J. Sullivan, Gisle Bjontegaard, and Ajay Lu-thra. Overview of the H.264/AVC video coding standard. IEEE Transactions on Circuits and Systems for Video Technology/July 2003.

[15]: Iain E.G. Richardson. H.264 MPEG-4 Video Compression : Video Coding for Next-generation Multimedia.

[16] : Ching-Yeh Chen ; Shao-Yi Chien ; Yu-Wen Huang ; Tung-Chien Chen ; Tu-Chih Wang ; Liang-Gee Chen, Analysis and architecture design of variable block-size motion estimation for H.264/AVC/2006.

[17] : http://forum.ripp-it.com/Codec-H264-X264-Encodage-en-x264-hp-et-h264-AVC-t8724.html : profils H.264 accédé 14/04/2013.

[18]: Yong-Jun Kim; Kyu-Yeul Wang; Sang-Seol Lee; Implementation of High Efficient CAVLC Encoder for H.264/AVC, 2010.

[19] : Chen-Han Tsai ; Chi-Sun Tang ; Liang-Gee Chen ;A flexible fully hardwired CABAC encoder for UHDTV H.264/AVC high profile video ,2012

[20] : http://see.xidian.edu.cn/vipsl/database_Video.html, accédé le 19/06/2013.

[21] : Introduction to Video Compression H.264, Dirk Farin,Uni versity of Mannheim Germany.

[22] : Wiegand, T. ; Schwarz, H. ; Joch, A. ; Kossentini, F. ; Sullivan, G.J. Rate-constrained coder control and comparison of video coding standards ,2003.

[23] : Shaw-Min Lei ; Ming-Ting Sun ,An entropy coding system for digital HDTV applications.

[24] : M.Dirk Robinson ,these ESTIMATION THEORETIC ANALYSIS OF MOTION IN IMAGE SEQUENCES,UNIVERSITY OF CALIFORNIA SANTA CRUZ ,2004.

[25] : Cristian GRAVA,Adequation algorithme - architecture pour le traitement multimedia embarqué ,2002.

[26] : B. K. P. Horn,B. G. Shunck, Determining optical flow, Artificial intelligence, 1981, Vol. 17,n°p. 185-203.

[27] : W. Enkelmann, Investigations of multigrid algorithms for the estimation of optical flow fields in image sequences,Proceedings of Comp. Vis. Graph. Image, 1988, Vol. 43, n° 1, p. 150-177.

[28] : B. Lucas,T. Kanade, An iterative image registration technique with an application to stereo video, Proceedings of DARPA Image Understanding Workshop, Washington, DC, 1981, p. 121-130.

[29] : H. Nagel,W. Enkelmann, An investigation of smoothness constraints for the estimation of displacement vector fields from image sequences, IEEE Transactions on Pattern Analysis and Machine Intelligence, 1986, Vol. 8, n° 1, p. 565-593.

[30] : J. L. Barron, D. J. Fleet,Beauchemin, Performance of optical flow techniques, International Journal of Computer Vision,1992, Vol. 12, n° 1, p. 43-77.

[31] L. Jacobson and H. Wechlcr,« Derivation of oplical flow using spatiotemporal frequency approach ». CVGIP, 38, 1987, pp.29-65.

[32] J. Wcng. « A theory of image matching », Proc. 3rd International Conference on Computer Vision, Osaka, Japon, 1990, pp.200-209.

[33] F.H. Adelson and J.R. Bergen, « Spatiotemporal energy models for the perception of motion ». Journal of the Optical Societyof America A., vol. 2, n°2,1985,pp. 284-299.

[34] : D.J. Hecger. Optical flow using spatiotemporal filters. International Journal of Computer Vision, pages 297-302, 1988.

[35] : D. Fleet and A. Jepson. Computation of component image velocity from local phase information. International Journal of Computer Vision, 5 :77-104, 1990.

[36] : Rapport de Stage : Estimation de mouvement subpixelique par blocs adaptée a la couleur avec modèle de mouvement, 14/12/2004.

[37] : M. Gallant, G. Cote, F. Kossenttni, « An Efficient Computation-Constrained Block-Based Motion Estimation Algorithm for Low Bit Rate Video Coding », IEEE Trans, on Image Processing, Vol 8, No. 12, pp. 1816-1823, Dec. 1999.

[38] : Thèse : Implantation optimisé d'estimateurs de mouvement pour la compression video sous plates-formes hétérogène multi-composant, 6/12/2007.

[39] : Rehan, M. ; Nashed, R.Efficient search area loading technique for block-based motion estimation and its FPGA implementation / 2009.