Université de Nice - Sophia Antipolis

École Doctorale STIC

Sciences et Technologies de l'Information et de la Communication

Rapport de Stage - DEA Image Vision

encadré par

Michel Barlaud, Marc Antonini et Éric Debreuve

et présenté par

Vincent Garcia

Estimation de mouvement subpixélique par blocs adaptée à la couleur avec modèle de mouvement

soutenu le 14 septembre 2004

Laboratoire d'accueil

Laboratoire I3S - Équipe CReATIVe

Année universitaire 2003 - 2004

Remerciements

Avant tout, je tiens à remercier le laboratoire I3S de Sophia Antipolis de m'avoir accepté au sein de l'une de ces équipes. Je remercie toute l'équipe CReATIVe pour son accueil si chaleureux créant une ambiance de travail très privilégiée.

Je remercie messieurs Michel Barlaud, Marc Antonini pour avoir dirigé mes travaux pendant toute la durée du stage.

Je remercie Eric Debreuve pour son encadrement, ses conseils et sa lecture.

Je remercie Marco pour m'avoir accueilli pendant les premières semaines de stage et m'avoir expliqué longuement le fonctionnement du codeur.

Je remercie Muriel et Thomas pour m'avoir aidé durant tout le stage avec leur expérience. Leur recul a permis de trouver de nombreuses erreurs dans le code me faisant ainsi gagner un temps très précieux.

Je remercie également Ariane pour sa bonne humeur permanente.

Je remercie Hervé et Céline pour la patience dont ils ont fait preuve lors de la lecture de mon rapport.

Je tiens également à remercier Sylvain, mon co-bureau, qui a eu l'endurance de me supporter durant tout le stage. Ne t'inquiète pas, tu en as seulement pour trois... ;-)

Je voudrais aussi remercier mon ami Tyler qui m'a aidé moralement par une présence quotidienne et des conseils avisés.

Je remercie tous ceux que je n'ai pas cités et qui ont pourtant contribué à l'achèvement du stage par leur présence, leurs conseils, leurs critiques et leur amitié.

Et enfin, j'achève cette partie en remerciant toute ma famille qui a était là pour moi dans des moments difficiles. Je les associe donc à ma réussite.

Table des matières

Introduction

Contexte

Le laboratoire I3S (Informatique Signaux et Système de Sophia Antipolis) est une unité de recherche créée en 1989 associant l'Université de Nice - Sophia Antipolis et le CNRS. Le laboratoire compte cinq thèmes de recherche :

- ACA : Algorithmique, Combinatoire et Applications
- ALM : Architectures Logicielles et Matérielles
- STL : Sciences et Techniques du Logiciel
- SIROCCO : SIgnal, RObotique, Communication, Contrôle et Optimisation
- IMAGES : Images numériques et vidéos

Le projet IMAGES se divise en deux équipes de recherche : ARIANA et CReATIVe. L'objectif du projet ARIANA est de fournir des outils de traitement des images, dans le but d'aider à la résolution des problèmes inverses dans le domaine de l'observation de la Terre et de la cartographie, en particulier aérienne et satellitaire.
Le projet CReATIVe (Compression, Reconstruction, Adaptées au Traitement d'Images et à la Vidéo) étudie, comme son nom l'indique, différents aspects du traitement d'images et de la vidéo en travaillant sur la segmentation par contours actifs ([1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20]), sur le codage vidéo ([21], [22], [23], [24], [25], [26]) ainsi que sur la compression géométrique de maillages surfaciques triangulaires ([27], [28], [29], [30], [31], [32], [33]).

Projets et objectifs

La quantité et la qualité des données visuelles (2D pour les images et 3D pour les vidéos) explose avec l'avènement et le développement des nouvelles technologies telles que les caméscopes et appareils photos numériques. Une limite semble se profiler dans la mesure où l'augmentation du nombre de données paraît plus rapide que la croissance des capacités de stockage (disques durs, CD, DVD, etc.). La seule option envisageable dans ce contexte est la compression de données.
Un des axes de recherche de l'équipe est le développement d'un codeur vidéo utilisant une transformée en ondelettes 3D (deux dimensions spatiales et une dimension temporelle). Les avantages d'une telle transformée sont d'abord de bonnes propriétés de décorrélation (qui en font un puissant outil pour la compression) ainsi que aspect de scalabilité (permettant d'adapter les caractéristiques de l'image au débit disponible ou à l'application souhaitée :

e.g. Internet).
Un tel codeur doit nécessairement prendre en compte les redondances spatiales et temporelles contenues dans la vidéo. La décorrélation spatiale est assurée par l'utilisation de le norme JPEG2000 et une transformée en ondelettes est également appliquée dans le domaine temporel. De manière à rendre cette dernière plus efficace, une estimation de mouvement doit être réalisée pour compenser la transformée en mouvement. Parmi les nombreuses méthodes d'estimation existantes, nous nous sommes penchés sur la plus répandue utilisée dans les standards vidéos (tels que MPEG2, MPEG4 et DivX), à savoir l'estimation de mouvement par mise en correspondance de blocs (block-matching).

Pour prendre en compte la diversité des méthodes de block-matching et les caractéristiques de chacune d'entre elles, nous avons réalisé une méthode d'estimation regroupant les principales techniques utilisées. Ce choix permet d'adapter la stratégie de mise en correspondance à notre codeur. Ma contribution dans le cadre du DEA Image-Vision consiste d'une part en l'implémentation et l'évaluation de ces techniques, et d'autre part dans l'étude de nouvelles techniques améliorant la qualité de l'estimation.

Plan

Dans un premier temps et dans un contexte général, je vous présente la transformée en ondelettes au fil de l'eau utilisée pour la compression. J'y expliquerai l'intérêt de la compensation en mouvement dans la transformée temporelle.

Dans une deuxième partie, je vous expose l'estimation de mouvement par mise en correspondance de blocs. Cette partie représente la majeure partie de mon travail. J'y étudierai l'influence des différentes stratégies de recherche, des critères de comparaison entre blocs ainsi que l'utilisation de l'interpolation d'images afin d'envisager des déplacements subpixéliques.

Enfin, dans une troisième partie, je vous propose un modèle de mouvement permettant de contraindre les vecteurs avant et arrière d'une même image lors de l'estimation du mouvement. Cette contrainte dite "bidirectionnelle" a l'avantage de faire économiser la moitié du débit réservé aux vecteurs dans le flux binaire de la vidéo et permet ainsi, à bas débit, d'augmenter la qualité globale de reconstruction de la vidéo.

Chapitre 1

Transformée en ondelettes

Face à l'accroissement massif du nombre et de la qualité de données, les réseaux de transmission arrivent à saturation. Un problème de stockage se pose maintenant malgré l'extension des capacités des disques durs et autres disques de stockage (DVD, CD-ROM). Dans ce contexte, la compression apparaît comme l'outil essentiel à la poursuite du progrès.

L'expansion d'internet nécessite que l'information puisse être lue par des réseaux hétérogènes. Le développement de standards de compression permet de définir une technique de codage et de décodage connue des différents systèmes. Les normes tentent de se développer en définissant un compromis entre la qualité, le débit cible et la complexité désirée. La transformée en ondelettes est très appréciée dans ce contexte et est exploitée dans la norme JPEG2000. Ses bonnes qualités de décorrélation et sa particularité multiéchelle en font un puissant allié d'une compression efficace. Son aspect multiéchelle permet un codage et un décodage progressif des données.

1.1 Introduction aux méthodes de compression

1.1.1 Schéma de compression

L'usage des signaux numériques devient de plus en plus courant avec le développement du multimédia et des applications interactives. La somme des données manipulées ou échangées est en perpétuelle progression. En outre, les signaux numériques sont codés sur un nombre imposant de bits. La compression s'effectue à l'aide de techniques ne concédant aucune perte ou en estimant que certains détails, fort peu perceptibles par l'usager, seront délibérément omis. Ainsi, deux grandes catégories de techniques se différencient : les méthodes dites « sans perte » et « avec perte » d'informations.

Le schéma de codage d'un signal est en général constitué d'une mise en forme du signal afin de l'adapter à la compression, d'une étape de quantification, d'une étape de codage entropique et finalement d'une adaptation au canal. L'étape de mise en forme se justifie par le fait que certains signaux, tels que les images ou les vidéos, sont très fortement corrélés. Afin de compresser efficacement les images, il est impératif de prendre en compte ce phénomène. Ainsi un schéma de compression classique se décline en deux étapes : une étape de décorrélation (réalisée soit par une méthode prédictive, soit par une transformée) suivie d'une étape de codage avec ou sans perte.

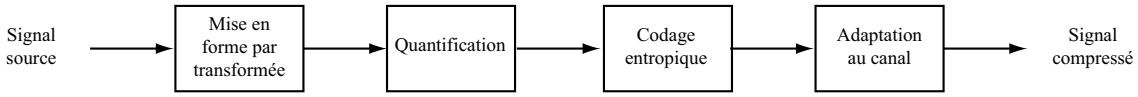


FIG. 1.1 – Schéma de compression

1.1.2 Aperçu de quelques méthodes de compression

Il existe deux grandes familles de méthodes de compression : les méthodes sans perte et les méthodes avec pertes. Cette section est un aperçu des quelques méthodes de compression couramment utilisées. L'utilisation des méthodes sans perte, dites conservatives, n'est que limitée dans la mesure où elles ne permettent qu'un faible taux de compression. Les méthodes avec pertes, dites non conservatives, ont la faculté d'accroître considérablement le taux de compression et de diminuer le débit du signal en deçà de la limite de Shannon.

Méthodes conservatives

Codage entropique

Le concept du codage entropique se fonde sur la théorie de l'information instituée par Shanon. La stratégie consiste à exploiter l'analogie entre la probabilité d'un événement et la quantité d'information à transmettre. Le codage entropique permet d'encoder différents symboles (luminance, chrominance, etc.) en leur attribuant un mot binaire. La longueur de ce mot dépend de la fréquence d'apparition du symbole considéré. La réalisation d'un code attribuant aux symboles les plus fréquents les mots les plus courts a pour conséquence de réduire le débit du signal.

Codage de Huffman : Le codage de Huffman est une méthode de codage entropique plébiscitée parmi les standards de compression d'images et vidéos. Le principe et d'encoder les différents symboles par des mots binaires de longueur variable, longueur dépendant de la probabilité d'apparition du symbole. Il utilise un code préfixe (i.e. un mot de code ne doit pas être le préfixe d'un autre mot) assurant l'unicité et la réversibilité du code.

Codage arithmétique : Le codage arithmétique [34] se singularise par sa capacité à coder chaque symbole sur un nombre non entier de bits. En réalité, il n'assigne pas un mot de code à chaque symbole mais il associe un point le l'intervalle $[0, 1[$ à un ensemble de symboles (cf. [35]). Le principe repose sur le découpage de l'intervalle $[0, 1[$. Chaque symbole se voit attribuer une partition de l'intervalle dont la taille est égale à sa probabilité d'occurrence. L'ordre de rangement est mémorisé pour être utilisé lors du décodage.
Le codage arithmétique est généralement plus performant que le codeur d'Huffman. Il tend vers la limite inférieure théorique mais cependant il est gourmand en ressource et nécessite de connaître a priori l'intégralité du signal avant de pouvoir procéder au codage.

Codage par plages de zéro : La méthode de codage par plages de zéros est une méthode permettant de compresser efficacement le signal si celui-ci contient un nombre important de plages de zéros. Le principe est assez simple. Le codeur prend en entrée un ensemble de données et compte le nombre de zéros entre deux symboles non nuls. La longueur de la plage de zéros peut ainsi être codée.

Méthodes non conservatives

Quantification scalaire : Le principe de la quantification est de projeter une variable continue sur une variable discrète qui prend ses valeurs sur un ensemble fini de points. En d'autres termes, étant donné un dictionnaire de symboles en nombre fini, à chaque symbole du signal est affecté un symbole du dictionnaire. Comme toute méthode avec pertes, le signal ainsi quantifié est dégradé et le processus est irréversible.

Quantification vectorielle : Le dictionnaire utilisé pour la quantification vectorielle utilise des ensembles de symboles, appelés vecteurs, permettant de coder plusieurs symboles simultanément. La quantification scalaire est un cas particulier de la quantification vectorielle avec des vecteurs de dimension un (i.e. utilisant un symbole par vecteur).

Ces méthodes de compressions peuvent être encore améliorées si l'on adapte le signal à leur caractéristiques. Les transformées trouvent ici leur intérêt.

1.1.3 Introduction aux transformées

Le codage par transformée a pour but de modéliser le signal à partir de fonctions mathématiques. Au lieu de coder directement le signal source, il s'avère souvent plus avantageux de restructurer les données de manière à les rendre plus adaptées aux différentes méthodes de compression. Des théorèmes portant sur la théorie du codage mettent en évidence l'efficacité optimale des diverses méthodes de compression pour des signaux bien particuliers. Un point fondamental marque le lien entre toutes les méthodes adoptant un codage par transformée. Il repose sur le fait qu'en traitement d'images la plupart des signaux numériques présentent d'importantes corrélations. Le rôle essentiel des transformées est de décorréler la source de données, tandis que celui du codeur est d'éliminer l'information redondante.

Je vous présente très brièvement ci-dessous quelques transformées classiques utilisées en traitement d'images :

Transformée de Karhunen-Loève - TKL : Pour un signal stationnaire au sens large et une compression linéaire, la transformée de Karhunen-Loève est une transformée optimale du point de vue de la décorrélation et de la concentration de l'énergie sur un nombre limité d'échantillons. Malgré ses performances, la transformée de Karhunen-Loève se voit souvent préférer la DCT à cause de la lourdeur de sa mise en œuvre.

Transformée de Fourier - TF : Afin de séparer les informations des fréquences différentes, la transformée de Fourier s'applique dans le domaine temporel et se projette dans le domaine fréquentiel. Elle analyse le contenu de la source en révélant le spectre des fréquences.
La transformée de Fourier s'adapte aux signaux linéaires et stationnaires mais elle n'est pas satisfaisante en compression d'images. Une simple quantification dans le domaine fréquentiel fait apparaître des discontinuités (phénomène de Gibbs). Cependant, l'existence d'un algorithme rapide permettant de calculer la transformée en fait un outil particulièrement intéressant.

Transformée en Cosinus Discrète - DCT : La transformée en cosinus discrète est très populaire et est intégrée dans des standards comme JPEG, MPEG1-2, H261, H263, H264.
La popularité de la DCT est liée à l'existence d'algorithmes rapides utilisant la FFT (Fast Fourier Transform). Elle s'applique sur les différents blocs de l'image plutôt que sur l'intégralité de l'image. La DCT a pour effet de réduire les corrélations présentes dans le signal. Cependant, son usage présente des dégradations inter-blocs, aussi appelés « effets de blocs », qui peuvent entraîner une gêne visuelle surtout pour la compression bas-débit.

Transformée en Ondelettes - TO : De la même façon que la transformée de Fourier peut se définir comme étant une projection du signal sur la base des exponentielles complexes, la transformée en ondelettes est la projection du signal sur la base des fonctions ondelettes. Découvertes par Grossman et Morlet, les ondelettes sont apparues en compression d'images suite aux travaux de Daubechies et Mallat. Elles permettent de traiter de petites irrégularités locales dans des signaux globalement lisses et réguliers. La transformée en ondelettes bénéficie de plusieurs avantages sur les transformées précédemment présentées. La transformée a de bonnes propriétés de décorrélation spatio-fréquentielle et de régularité qui en font un outil particulièrement adapté à la compression.

Il est important de souligner qu'une transformée ne réalise pas de compression. Seul le codeur appliqué en sortie de la transformée compresse le signal. En résumé, une transformée efficace est une transformée inversible (indispensable pour le décodage d'images haute-définition), qui décorrèle de manière efficace l'information et qui concentre l'énergie sur un minimum de coefficients.

Je vous présente dans la suite de ce chapitre le projet dans lequel je me suis inséré, à savoir la transformée en ondelettes 2D+t compensée en mouvement.

1.2 Transformée en ondelettes 2D+t compensée en mouvement

Les codeurs vidéos standards tels que MPEG4 et H264 ne prennent en compte la corrélation temporelle qui existe dans une vidéo que lors de l'estimation de mouvement.

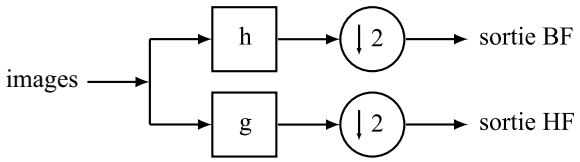


FIG. 1.2 – Transformée en ondelettes classique

L'emploi d'une transformée temporelle semble être un choix judicieux et indispensable de façon à exploiter au mieux la redondance temporelle présente dans une vidéo.

1.2.1 Généralités sur la transformée en ondelettes

Dans le domaine de la compression d'images, la transformée en ondelettes repose sur le filtrage du signal original par deux bancs de filtres permettant d'obtenir deux sorties nommées sous-bandes. Il s'agit dans les deux cas d'un filtrage suivi d'un sous-échantillonnage par deux. Le sous-échantillonage utilisé avec un transformée spatiale implique des sous-bande dimensionnellement deux fois plus petites. Avec une transformée temporelle, les images sont de même taille mais en nombre deux fois plus faible. La première sous-bande est constituée du signal d'entrée à une résolution deux fois plus faible et est appelée basse-fréquence (BF). L'autre, appelée haute-fréquence (HF), permet de reconstruire le signal original à partir de la sous-bande BF (voir figure 1.2).
La transformée spatiale étant assurée par la norme JPEG2000, nous étudions la troisième dimension de la transformée que nous utilisons : la transformée temporelle.

Notons x_n les images de la séquence d'entrée. Le calcul des sous-bandes haute-fréquence h_n et basse-fréquence l_n pour la transformée temporelle se fait de la façon suivante :

$$h_k = \beta_0 x_{2k} + \sum_{i \in D_h} \beta_i \left(x_{2k-i} + x_{2k+i} \right) \tag{1.1}$$

$$l_k = \eta_0 x_{2k+1} + \sum_{i \in D_l} \eta_i \left(x_{2k+1-i} + x_{2k+1+i} \right) \tag{1.2}$$

où les $(\beta_i)_{i \in D_h}$ et les $(\eta_i)_{i \in D_l}$ sont respectivement les coefficients du filtre passe-haut et passe-bas de la transformée.

Les sous-bandes obtenues contiennent des images de coefficients h_n et l_n , de même taille que les images de la vidéo d'entrée. Il est ainsi possible de traiter les coefficients comme des images classiques.

Un autre avantage de la transformée en ondelettes réside dans la possibilité de redécomposer récursivement une sous-bande. On parle alors d'analyse multirésolution (voir figure 1.3), ou de décomposition sur plusieurs niveaux. L'opération la plus fréquente est de redécomposer la sous-bande BF obtenue au niveau supérieur afin d'améliorer la décorrélation. On parle alors d'une décomposition dyadique.

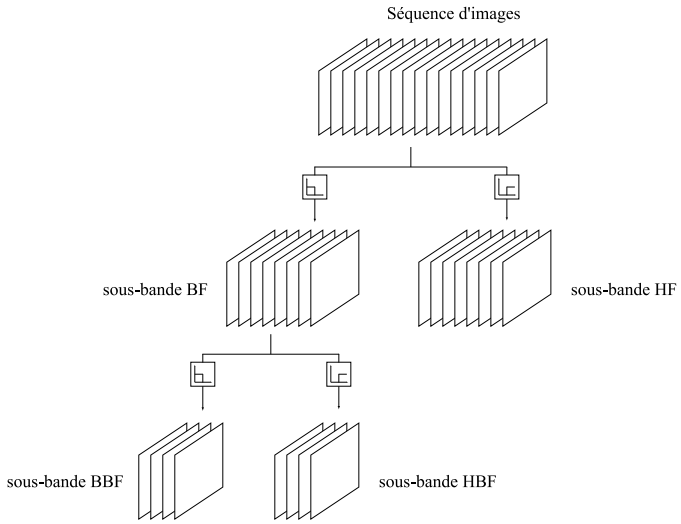


FIG. 1.3 – Analyse multirésolution

1.2.2 Transformée temporelle au fil de l'eau

Le problème inhérent à la transformée temporelle est qu'il faudrait stocker en mémoire la séquence dans son intégralité pour la traiter d'un bloc. Ceci n'est pas envisageable, en particulier pour des séquences assez longues. Une solution possible est de découper la séquence en groupes d'images (GOP, Groups Of Pictures) de longueur variable suivant la taille du filtre appliqué. Chaque GOP est alors transformé indépendamment.
Cependant le découpage en GOP introduit des artéfacts visibles sur la vidéo reconstruite après compression et décompression. En effet, non seulement chaque GOP est traité indépendamment de ses voisins provoquant nécessairement une discontinuité, mais les images situées aux bords des GOP sont symétrisées pour les besoins du calcul ce qui accentue la discontinuité. La conséquence est une dégradation visible des images reconstruites aux bords des GOP. C'est pourquoi une autre solution est proposée : la transformée au fil de l'eau.
La transformée au fil de l'eau est calculée au fur et à mesure de l'acquisition des images d'entrée. Le GOP est décalé image par image permettant d'assurer la continuité des sous-bandes produites.

1.2.3 Compensation du mouvement

La transformée temporelle permet de décorréler efficacement une séquence vidéo dans le cas le plus général. Cependant, cette décorrélation suit strictement l'axe du temps et ne tient pas compte du mouvement présent dans la vidéo (mouvement des objets et/ou de la caméra). En conséquence, des points d'images successives seront associés pour le calcul de la transformée alors qu'ils n'ont rien en commun mis à part leurs coordonnées sur l'image. Ce fait entraîne non seulement une sous-optimalité de la décorrélation, mais aussi des effets visibles dans le cas d'une reconstruction après codage avec pertes (pertes généralement situées dans la sous-bande haute fréquence contenant les détails). La nécessité de

tenir compte du mouvement est donc clairement établie. C'est le rôle de la compensation de mouvement, dont l'importance a été reconnue très tôt [36].

Afin d'associer un point d'une image à ses correspondants dans les images voisines, il faut disposer de pointeurs vers ces points, les vecteurs d'estimation du mouvement. Pour chaque point de chaque image sont établis des vecteurs vers ses antécédents dans les images précédentes et des vecteurs vers ses successeurs dans les images suivantes. Ces vecteurs interviennent alors dans le calcul de la transformée temporelle.

Ma contribution commence à ce niveau. En effet, le premier but de mon stage a été de créer un estimateur de mouvement permettant d'obtenir ces vecteurs mouvement par diverses méthodes en vue de les comparer. Le second a été d'utiliser un modèle de mouvement linéaire pour une estimation de mouvement avec contrainte bidirectionnelle. Ainsi, l'ensemble de mon travail est exposé dans les deux chapitres suivants.

Chapitre 2

Estimation du mouvement

L'estimation du mouvement est un problème incontournable dans le domaine du traitement de séquences d'images. Le but est d'estimer le mouvement des objets contenus dans une vidéo sous forme d'un champ dense de vecteurs.
Les applications d'une telle estimation sont nombreuses et variées et nous pouvons noter par exemple la segmentation d'objets en mouvement, la compensation du mouvement utilisée en compression vidéo (normes MPEG 2, MPEG 4, H264), etc.
L'estimation de mouvement est excessivement coûteuse en temps de calcul. Cela peut prendre jusqu'à 80% des ressources matérielles de la machine. Une estimation de faible qualité est souvent source de problèmes importants comme par exemple des effets de blocs visuellement gênants. Ainsi, depuis plus de vingt ans, son étude reste un sujet de recherche important.
Dans ce chapitre, je présenterai l'estimation du mouvement de manière générale en illustrant son fonctionnement. Puis je réaliserai un inventaire des principales techniques de mise en correspondance de blocs et conclurai sur l'évaluation des performances de ces dernières.

2.1 Block-matching : mise en correspondance de blocs

2.1.1 Principe

Le principe général du bloc-matching est d'exploiter les redondances temporelles existant entre des images consécutives. Pour cela, considérons une séquence vidéo dans laquelle nous voulons estimer le mouvement des différents objets qui y sont contenus. Pour simplifier l'estimation, nous ne considérons que le mouvement présent entre deux images successives : l'image courante et l'image de référence. Chaque image est subdivisée en blocs de taille égale (généralement de 8×8 ou 16×16 pixels, voir figure 2.1) et chaque bloc est considéré comme étant un objet indépendant. Nous faisons l'hypothèse que le mouvement des pixels est uniforme à l'intérieur d'un bloc. De manière informelle, l'algorithme consiste, pour un bloc de l'image courante, à choisir un bloc dans l'image de référence (passée ou future) et à calculer un critère comparaison entre ces deux blocs. L'opération est réitérée en choisissant un autre bloc jusqu'à ce que tous les blocs d'une zone déterminée de l'image de référence (appelée « fenêtre de recherche ») aient été testés ou jusqu'à un critère d'arrêt arbitraire. Le bloc le plus semblable est ainsi identifié dans l'image de référence

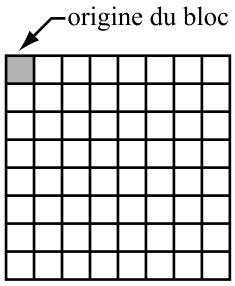


FIG. 2.1 – Illustration d'un bloc de taille 8×8 pixels

pour chaque bloc de l'image courante. Nous obtenons de cette manière pour tout bloc un vecteur déplacement caractéristique du mouvement de ce dernier.
L'utilisation d'une fenêtre de recherche permet de limiter le nombre de blocs de référence que l'algorithme doit tester. La taille de la fenêtre dépend d'un déplacement maximal autorisé fixé par l'utilisateur.
Initialement, on peut vouloir placer l'origine d'un bloc en son centre. Cependant, il est beaucoup plus astucieux de le situer dans le coin supérieur gauche (voir figure 2.1) pour deux raisons : premièrement, les blocs ont en général des cotés de taille paire ce qui implique qu'il n'y a pas de pixel central ; deuxièmement, les tests d'appartenance à la fenêtre sont moins coûteux en temps de calcul avec cette convention.

Le nombre et l'ordre dans lequel les blocs sont testés ainsi que les conditions d'arrêt ont des influences majeures sur l'efficacité de l'algorithme de recherche. Je détaillerai dans ce chapitre quelques méthodes proposées dans la littérature pour optimiser cette recherche.

2.1.2 Critères de comparaison

Le « critère de comparaison » précédemment mentionné pour trouver le bloc correspondant est une mesure objective de la dissemblance entre les valeurs des pixels contenus dans chaque bloc. Il en existe plusieurs qui prennent en compte ou non la couleur. De manière générale, la plupart des articles traitant de l'estimation de mouvement considèrent que les blocs sont caractérisés par la luminance. Ceci est dû au fait que l'œil humain est davantage sensible à l'intensité lumineuse qu'à la couleur. Cependant, ignorer les chrominances peut alors poser quelques problèmes dont un exemple sera donné dans la section évaluant les critère de comparaison. Je vous présente les différents critères de comparaison couramment utilisés en traitement d'image et la manière dont la couleur a été inclue. Mais avant, introduisons les notations nécessaires à la bonne compréhension de la suite.

Notations

Tout d'abord, nous ne considérons que des blocs carrés de même dimension. Les critères de comparaison sont calculés entre deux blocs : le bloc de référence noté B_r et le bloc courant noté B_c. Chacun est identifié par son origine comme vu figure 2.1. D'autre part, il est important de noter que les blocs sont codés en YUV (luminance Y, chrominance U, chrominance V). Ainsi, la luminance du pixel (i, j) du bloc de référence est donnée par

$B_r(i,j,1)$ et les chrominances par $B_r(i,j,2)$ et $B_r(i,j,3)$ avec $i,j \in [1,m]$. Si l'on considère que le bloc est centré en (x,y) et que l'image de référence notée I_r est également au format YUV, cela correspond à la formulation plus couramment utilisée $I_r(x+i,y+j,1)$ et nous pouvons écrire l'égalité suivante :

$$B_r(i,j,1) = I_r(x+i,y+j,1)$$

Nous considérerons pour la présentation des différents critères la forme $B_r(i,j,.)$ qui ne prend en compte que le bloc et les pixels qu'il contient. En effet, ceci possède un triple avantage : le premier est l'allègement des notations ce qui facilite la lisibilité des formules. Le deuxième avantage est directement lié à la programmation orientée objet. En détachant le bloc de l'image, cela permet de considérer chaque bloc comme un objet à part entière. De cette manière, le calcul du critère n'opère que sur les deux blocs considérés sans utiliser les différentes images et également sans manipuler des indices généralement compliqués. Enfin, les cas problématiques de position des blocs (origine externe à l'image, bloc partiellement contenu dans l'image) sont gérés à la construction des ces derniers ce qui permet d'éviter des difficultés généralement rencontrées avec l'approche classique.

Nous allons utiliser pour certains critères les moyennes des blocs que nous notons $\overline{B_r}$ et $\overline{B_c}$. Ces moyennes sont relatives à l'information dont on veut calculer la moyenne. Par exemple, la moyenne du bloc de référence si l'on ne considère que la luminance sera obtenue comme suit :

$$\overline{B_r(1)} = \frac{\sum_{i=1}^{w}\sum_{j=1}^{w} B_r(i,j,1)}{w^2}$$

Enfin, pour alléger les notations, nous supposons dans les doubles sommes que les indices i et j parcourent l'intervalle $[1,w]$. Ainsi, à titre d'exemple, la formule de la moyenne ci-dessus se réécrit :

$$\overline{B_r(1)} = \frac{\sum_{i}\sum_{j} B_r(i,j,1)}{w^2}$$

Sum of Square Differences - SSD

Ce critère n'est autre que la somme des différences élevée au carré entre les pixels correspondant des deux blocs. L'expression de la SSD est de la forme :

$$SSD(B_c,B_r) = \sum_i \sum_j [B_c(i,j,1) - B_r(i,j,1)]^2$$

où i et j parcourent respectivement les lignes et les colonnes des blocs. Ce critère est un des plus simples existant.
Cependant, ce critère n'est pas adapté aux images en couleurs du fait de l'unique utilisation de la luminance. En effet, deux blocs peuvent être très similaires en luminance et beaucoup moins semblables en couleur. Pour pallier à ceci, nous avons défini un critère $SSDColor$ qui prend en compte la couleur :

$$SSDColor(B_c,B_r) = \sum_i \sum_j \sum_{c=1}^{3} [B_c(i,j,c) - B_r(i,j,c)]^2$$

On pourra noter dans ce critère que la luminance et les chrominances sont pondérées de la même manière. Nous pourrions imaginer donner des pondérations différentes pour favoriser la luminance ou les chrominances.

Zero-mean Normalized Sum of Square Differences - ZNSSD

Ce critère est une variante du critère SSD précédemment défini. Il permet de prendre en compte les moyennes des blocs et de normaliser le résultat :

$$ZNSSD(B_c, B_r) = \frac{\sum_i \sum_j [(B_c(i,j,1) - \overline{B_c(1)}) - (B_r(i,j,1) - \overline{B_r(1)})]^2}{\sqrt{\sum_i \sum_j (B_c(i,j,1) - \overline{B_c(1)})^2 \sum_i \sum_j (B_r(i,j,1) - \overline{B_r(1)})^2}}$$

Nous définissons comme précédemment le critère $ZNSSDColor$:

$$ZNSSDColor(B_c, B_r) = \frac{\sum_i \sum_j \sum_{c=1}^{3} [(B_c(i,j,c) - \overline{B_c(c)}) - (B_r(i,j,c) - \overline{B_r(c)})]^2}{\sqrt{\sum_i \sum_j \sum_{c=1}^{3} (B_c(i,j,c) - \overline{B_c(c)})^2 \sum_i \sum_j \sum_{c=1}^{3} (B_r(i,j,c) - \overline{B_r(c)})^2}}$$

Absolute Value - AV

Ce critère est quasiment identique au critère SSD. Nous ne considérons pas ici le carré de la différence mais la valeur absolue de la différence.

$$AV(B_r, B_c) = \sum_i \sum_j |B_r(i,j,1) - B_c(i,j,1)|$$

$$AVColor(B_r, B_c) = \sum_i \sum_j \sum_{c=1}^{3} |B_r(i,j,c) - B_c(i,j,c)|$$

Ce critère a pour particularité de considérer toutes les différences de la même manière alors que le critère SSD pénalise davantage les grandes erreurs. Par exemple, avec le critère SSD, une différence de un à dix reprises est moins pénalisante qu'une différence de dix à une reprise. Pour le critère AV la pénalité est identique dans les deux cas.

Des travaux complémentaires élaborés pendant ce stage nous ont conduit à définir d'autres critères basés sur la variance et la moyenne. Néanmoins, n'ayant pas abouti à des résultats satisfaisants, nous ne nous attarderons pas davantage.

2.1.3 Prédictions avant et arrière (forward and backward)

L'algorithme du bloc matching permet d'estimer le mouvement des blocs entre deux images à des temps consécutifs. Le mouvement ainsi calculé va permettre de prédire l'image $n+1$ en combinant l'information contenue dans les pixels présents dans l'image n et les vecteurs mouvement. Nous avons vu précédemment que nous utilisions deux blocs. Cependant, nous avons délibérément omis de spécifier de quelles images sont extraits chacun des blocs. Ainsi, deux types de prédiction sont possibles : la prédiction avant (forward) et la prédiction arrière (backward).

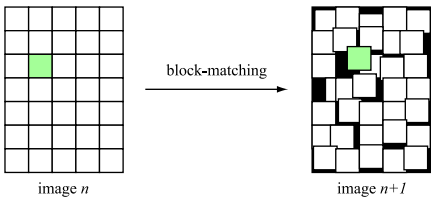


FIG. 2.2 – Block-matching avec prédiction forward

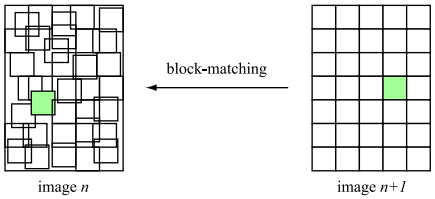


FIG. 2.3 – Block-matching avec prédiction backward

La prédiction forward consiste à diviser l'image n en blocs et à trouver leur position dans l'images $n+1$. Comme le montre la figure 2.2, le block-matching avec prédiction forward répond pour chaque bloc à la question "Où va le bloc courant ?".

En revanche, la prédiction backward, illustrée sur la figure 2.3, divise l'image $n+1$ en blocs et cherche leur position dans l'image n. Dans ce cas la question est "D'où vient le bloc courant ?".

Les résultats sont très différents. Il apparaît par construction que l'image prédite par prédiction forward présente des « trous ». En effet, de manière générale les blocs n'ont pas un mouvement unique pour toute l'image. Lors de la prédiction certains blocs se recouvrent et laissent apparaître de zones n'ont prédites et par conséquent noires (voir figure 2.2). Ce problème est entièrement résolu par l'utilisation de la prédiction backward qui ne laisse entrevoir aucun « trou » dans la prédiction.

2.2 Parcours du voisinage

Il existe de nombreuses méthodes de block-matching dans l'ensemble des articles [37], [38], [39], [40], [41], [42] cherchant à optimiser l'efficacité et la rapidité de l'algorithme. Rappelons que étant donné un bloc dans l'image courante, l'objectif est de trouver le bloc dans l'image précédente ou suivante (selon le type de prédiction utilisé) qui lui correspond au mieux selon des critères précédemment définis. Nous vous présentons dans cette partie diverses méthodes largement utilisées, de la plus simple et la plus ancienne, à savoir la recherche exhaustive, à des méthodes très récentes. Je vous expose l'évolution des techniques au cours du temps, les intérêts et les inconvénients qu'elles représentent.

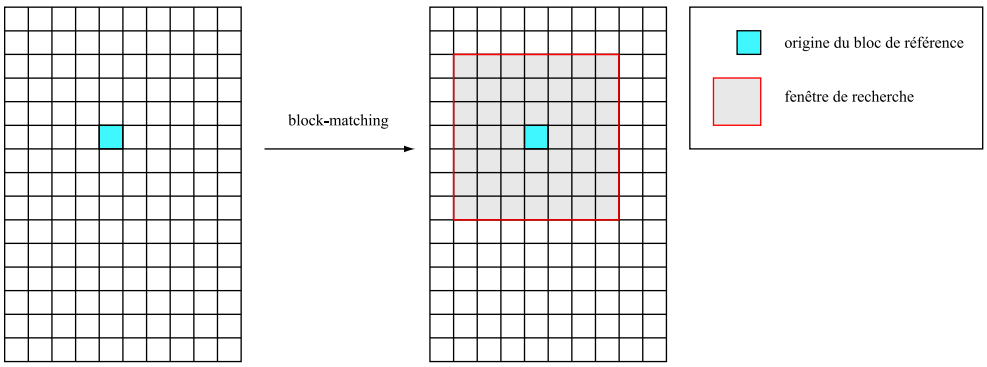


FIG. 2.4 – Full Search Algorithm

2.2.1 Algorithme de recherche exhaustive - Full Search Algorithm

Le block-matching « Full Search » consiste à parcourir de manière exhaustive l'ensemble de la fenêtre de recherche. En procédant de cette manière, le bloc retourné par l'algorithme sera celui qui minimise le critère de comparaison. Pour simplifier la compréhension de l'algorithme visible sur la figure 2.4, un bloc est identifié par son origine.
La fenêtre de recherche est ici définie par un déplacement $[-3,3] \times [-3,3]$ c'est-à-dire que le déplacement maximal autorisé est de trois dans toutes les directions. Habituellement, le déplacement maximal est de sept mais aucune norme n'est définie. En effet, un déplacement maximal petit pénalise les grands déplacements et augmente naturellement l'erreur de prédiction dans le cadre d'une vidéo présentant de tels déplacements. Cependant, peu de blocs seront testés et l'algorithme sera alors rapide. A l'inverse, un grand déplacement autorisé permet de donner plus de liberté aux vecteurs mouvement mais augmente de manière quadratique le nombre de blocs testés. Il y a donc un compromis entre qualité et rapidité de l'algorithme. D'autre part, dans certains cas, contraindre le déplacement est une façon d'ajouter un a priori sur le déplacement recherché.

2.2.2 Algorithme de recherche multirésolution à trois étapes - Three Step Search Algorithm

La recherche exhaustive étant trop coûteuse, plusieurs algorithmes dits rapides ont vu le jour et le « Three-Step Search » présenté dans [37] a été précurseur dans ce domaine. 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.

La figure 2.5 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 (pour la norme L^∞) 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

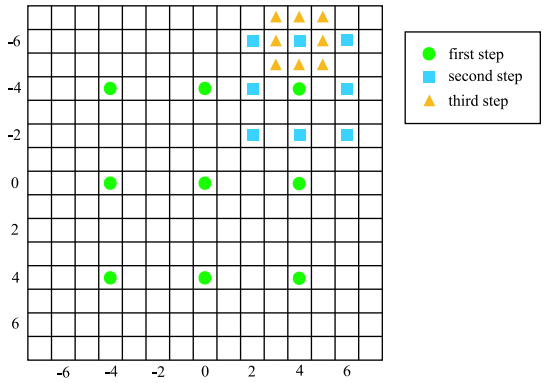


FIG. 2.5 – Three Step Search Algorithm

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 point 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.

La première remarque que nous pouvons faire concerne le nombre de points visités. Pour une fenêtre de recherche de $[-7,7] \times [-7,7]$ pixels, là où l'algorithme exhaustif teste $15^2 = 225$ points, l'algorithme Three Step n'en parcourt que 25 soit 9 fois moins. Nous détaillerons la discussion rapport au nombre de points testés pour chaque algorithme dans une section suivante où nous exposons les différents résultats obtenus. En effet, il est important de se rendre compte que la proposition décrite précédemment par rapport à la décroissance du critère n'est pas tout à fait exacte. Les algorithmes rapides permettent de converger vers un minimum local et non nécessairement global. Les conséquences de cela seront également exposées.

La seconde remarque repose sur le concept même de l'algorithme. Tel qu'il a été proposé, cet algorithme permet seulement de parcourir une fenêtre de recherche de $[-7,7] \times [-7,7]$ pixels alors que l'algorithme exhaustif n'a aucune contrainte algorithmique par rapport à la fenêtre de recherche. Pour pallier à ceci, nous avons implémenté un algorithme basé sur le "Three Step Search" que nous nommons « N Step Search ». Nous choisissons comme déplacement initial un nombre (généralement une puissance de deux) et à chaque pas, nous divisons le déplacement par deux. Par exemple, avec un déplacement initial de huit, l'algorithme fera quatre pas avec comme déplacements huit, quatre, deux et un et la fenêtre de recherche correspondante sera de $[-15,15] \times [-15,15]$ pixels. De cette manière, des fenêtres plus grandes peuvent être explorées et nous n'excluons pas la possibilité d'avoir un déplacement initial autre qu'une puissance de deux. Par exemple, pour un déplacement initial de six, les trois pas correspondent aux déplacements de six, trois et un. On notera que le « Three Step Search Algorithm » est alors un cas particulier de notre algorithme où le déplacement initial est de quatre. L'utilisation de la division euclidienne assure que le dernier pas se fera avec un déplacement pixélique. Toutefois, notre algorithme ne permet pas d'explorer toutes les tailles de fenêtre. Il est donc important d'en considérer une réellement

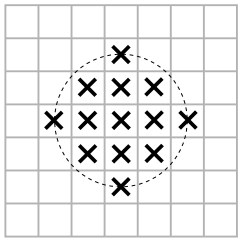


FIG. 2.6 – Un modèle de recherche approprié : un disque de rayon deux pixels.

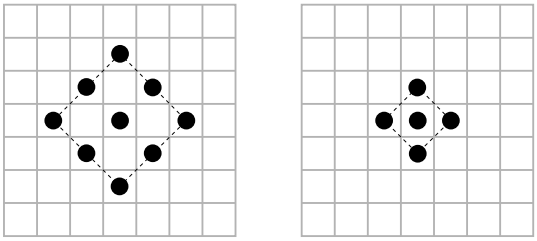


FIG. 2.7 – Modèles de recherche dérivés de la figure 2.6 utilisés dans l'algorithme « Diamond Search »

exploitable par tous les algorithmes et nous choisissons la fenêtre de $[-7, 7] \times [-7, 7]$ pixels.

Notez pour finir qu'une nouvelle méthode « Three Step Search » a été publié dans [38] et est nommée « New Three Step Search ». Cette méthode possède un critère d'arrêt en prenant en compte, lors d'un premier pas, les huit voisins de l'origine. Ceci permet à l'algorithme de converger plus rapidement en cas de faible déplacement. Cependant, nous avons préféré implémenter de plus récentes méthodes pour comparer les différentes approches.

2.2.3 Algorithme de recherche sur une grille en diamant - Diamond Search Algorithm

L'algorithme « Diamond Search » (DS) présenté dans [39] utilisent deux modèles de recherche illustrés dans la figure 2.7 dérivés du modèle de diamant illustré dans la figure 2.6. Le premier modèle appelé « Large Diamond Search Pattern » (LDSP) comprend neuf points dont huit points sur le bord du diamant situés à une distance de deux pour la norme L^1 et un point au centre pour composer une forme de diamant. Le second modèle nommé « Small Diamond Search Pattern » (SDSP) contient cinq points dont quatre sont sur le bord (situés à une distance de un pour la norme L^1) et un au centre.

L'algorithme du block-matching par « Diamond Search » se résume de la manière suivante :

Première étape : Le LDSP est centré sur l'origine du bloc courant et les neuf blocs

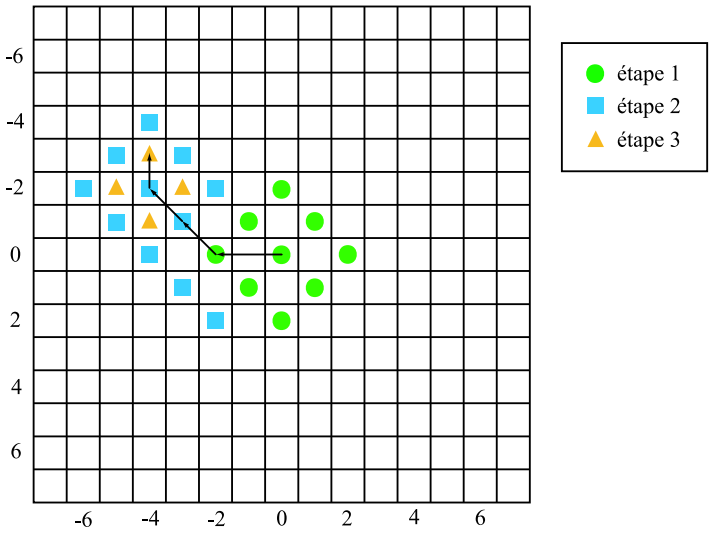


FIG. 2.8 – Diamond Search Algorithm

sont testés. Si le bloc minimisant la distorsion est celui du centre, l'algorithme passe directement à la troisième étape, sinon, il continue avec la seconde.

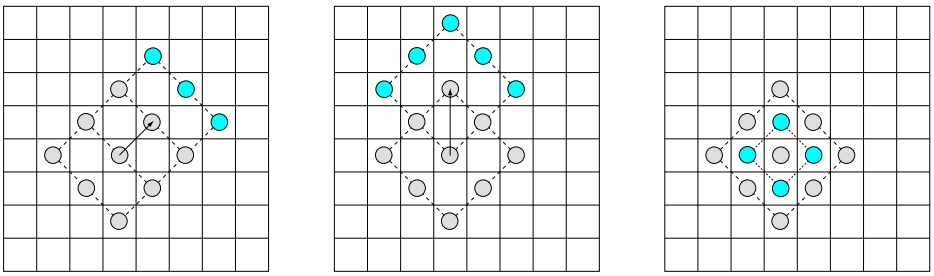
Deuxième étape : L'origine du bloc minimisant le critère et calculé précédemment est re-positionné pour être le centre d'un nouveau LDSP. Les blocs du modèle sont alors testés successivement. Si le nouveau bloc minimisant le critère est le bloc central, on continue avec la troisième étape. Sinon, on répète la deuxième étape.

Troisième étape : On échange le modèle LDSP avec le modèle SDSP et on teste les blocs. Le bloc obtenu lors de cette étape représente la solution finale pour obtenir le vecteur mouvement du bloc.

La figure 2.8 illustre le déroulement de l'algorithme en indiquant la différence entre les trois étapes.

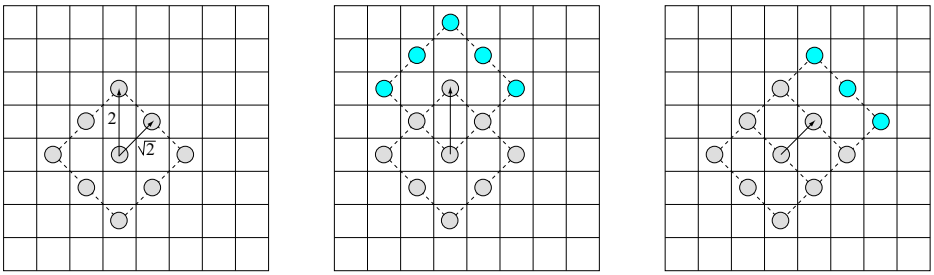
Lors de l'appel récursif de la deuxième étape, nous pouvons remarquer sur la figure 2.8 que les modèle LSDP se recouvrent partiellement. Ainsi, dans un souci d'optimisation, nous considérons que lors de cette étape trois cas sont possibles et représentés sur la figure 2.9. Dans le premier cas (figure 2.9 (a)), le bloc minimisant le critère se trouve sur le bord du LDSP. Pour compléter le modèle lors de cette étape, il est nécessaire de tester seulement cinq blocs placés comme l'indique le schéma. De même, dans le deuxième cas (figure 2.9 (b)), le bloc se trouve dans un coin du diamant et seuls trois blocs doivent être testés. Enfin, dans le dernier cas ((figure 2.9 (c)), les quatre blocs de la troisième étape de l'algorithme sont testés comme expliqué précédemment.

L'algorithme DS permet, comme nous le verrons dans la suite du rapport, d'optimiser la recherche dans le sens où moins de points sont visités par rapport au « Three Step Search » tout en augmentant la qualité de l'estimation.



(a) cas 1 : bloc sur le bord (b) cas2 : bloc dans un coin (c) cas 3 : bloc au centre

FIG. 2.9 – Optimisation du « Diamond Search Algorithm »



(a) (b) (c)

FIG. 2.10 – Problèmes liés au « Diamond Search »

2.2.4 Algorithme de recherche sur une grille hexagonale - Hexagon-Based Search Algorithm

Les modèles de recherche ont une influence importante sur la rapidité et la qualité de l'estimation de mouvement. Ainsi, l'algorithme DS permet d'obtenir de meilleurs résultats que les algorithmes utilisant des modèles carrés. Nous avons vu qu'avec l'utilisation du LDSP, les points considérés sont à une distance de 2 pour la norme L^1. Cependant, si nous voulons maintenant utiliser la norme L^2, nous remarquons que les points du modèle sont situés à une distance de 2 ou de $\sqrt{2}\approx 1.4142$ (voir figure 2.10 (a)). Le voisinage n'étant pas homogène risque de privilégier certaines directions. De plus, l'optimisation proposée pour le DS, consistant à ne tester que le nombre de blocs nécessaires à la complétion du diamant lors de la boucle sur la deuxième étape, fait intervenir un nombre variable de blocs selon l'emplacement du bloc minimisant la distorsion (voir figure 2.10 (b) et (c)).

Ainsi, l'algorithme « Hexagon-Based Pattern » (HEXBS) proposé dans [41] tente de trouver une solution à ces remarques, ou du moins propose une alternative à l'algorithme DS tout en gardant la même philosophie. En effet, l'algorithme utilise, comme dans le DS, deux modèles de recherche illustrés dans la figure 2.11 : un modèle large (« Large Hexagonal Search Pattern » - LHSP) et un modèle petit (« Small Hexagonal Search Pattern » - SHSP). La première remarque que l'on peut faire est que le SHSP est identique au SDSP. Deuxièmement, le LHSP contient sept points alors que le LDSP en contient neuf. Cela permet d'entrevoir la diminution du nombre de points testés par l'algorithme hexagonal.

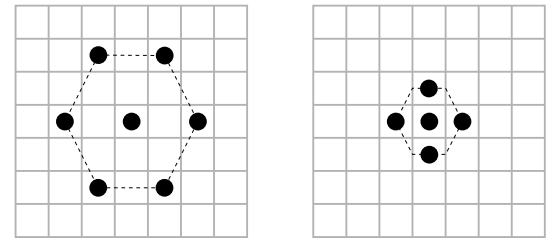


FIG. 2.11 – Modèles de recherche utilisés pour l'algorithme « Hexagon-Based Search »

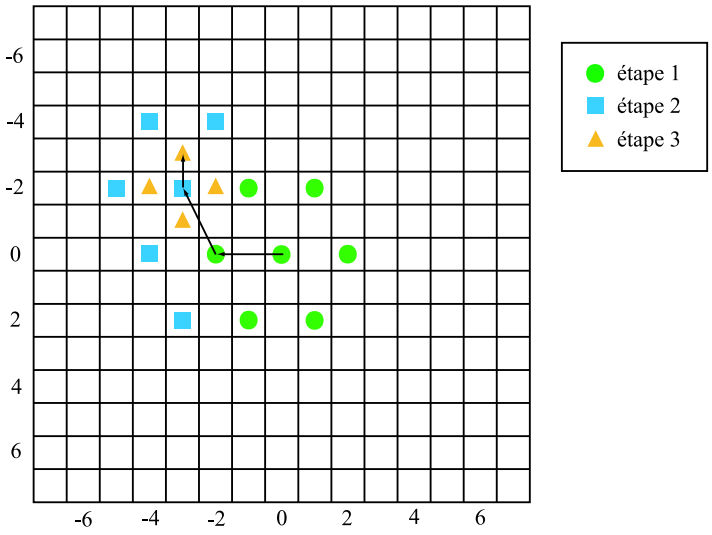


FIG. 2.12 – Hexagon-Based Search Algorithm

Enfin, les points situés sur le LHSP sont situés à une distance de 2 ou de $\sqrt{5}\approx 2.2361$. L'hexagone constitue ainsi une meilleure approximation du cercle que le carré ; cela représente un avantage certain si la norme L^2 est utilisée. Enfin, nous pouvons noter que l'hexagone est une figure qui permet de recouvrir l'espace de manière optimale et nous pensons ainsi pouvoir accroître les performances de l'estimateur de mouvement.

L'algorithme HEXBS, dont un exemple de déroulement est donné figure 2.12, est en tout point similaire au DS mis à part l'utilisation des modèles de recherche LHSP et SHSP à la place des modèles LDSP et SDSP.

On retrouve également dans le HEXBS l'optimisation présente dans le DS. Nous pouvons voir sur la figure 2.13 que, quelque soit l'emplacement du bloc minimisant le critère, seuls trois blocs devront être testés pour l'itération suivante. Ceci représente un avantage par rapport à l'algorithme DS au niveau de l'implémentation car il n'y a pas de cas particulier, tout est géré identiquement et nous sommes certains que chaque itération de la deuxième étape n'ajoute que le test de trois blocs.

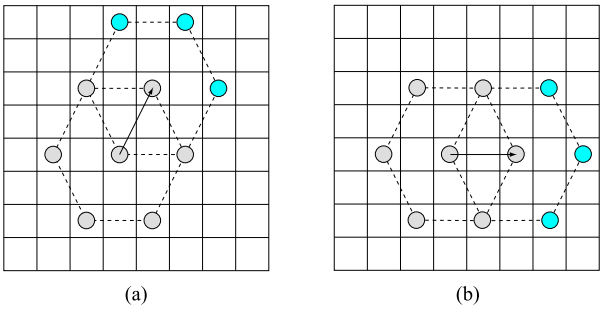


FIG. 2.13 – Optimisation de l'algorithme « Hexagon-Based Search »

Nous verrons plus loin que l'algorithme HEXBS permet de diminuer le nombre de blocs testés par rapport à l'algorithme DS. Cependant, nos tests mettent en évidence le fait qu'il est en général moins efficace au niveau qualitatif.

2.3 Estimation du mouvement subpixélique

Les techniques proposées jusqu'ici permettent de réaliser un estimation du mouvement de façon plus ou moins efficace, tant au niveau de la qualité qu'au niveau de la rapidité de l'algorithme. Cependant, le déplacement obtenu par ces méthodes fait intervenir des vecteurs aux coordonnées entières ce qui contredit totalement la continuité de l'espace dans lequel nous évoluons. Pour prendre en compte cet aspect, l'interpolation des images et son application au block-matching permettent de se rapprocher du monde réel.

2.3.1 Interpolation subpixélique - principe général

Les champs de vecteurs résultants des différentes méthodes font apparaître naturellement des coordonnées entières car nous considérons que le déplacement se fait sur les pixels. En effet, le pixel est par définition le plus petit élément représentable dans une image et il parait évident que l'estimation du mouvement entre deux images ne peut donner que des déplacements pixéliques. Cependant, le déplacement réel d'un objet ne se fait pas en fonction de l'échantillonnage du capteur d'acquisition et il en résulte que le mouvement fait généralement intervenir des déplacements non entiers. De ce fait, l'expression du mouvement par des vecteurs ayant des coordonnées non entières devient une nécessité pour une estimation de meilleure qualité. L'interpolation subpixélique trouve ici son intérêt.

Nous disposons d'une image composée de pixels. Chaque pixel est en fait la synthèse des couleurs d'une zone de l'espace. Il n'est en effet pas possible d'imaginer la représentation du monde sous forme continue du fait de la masse d'information à traiter et de la capacité limitée de stockage que nous possédons. Ainsi, une image est une approximation de la projection d'une scène tridimensionnelle sur un plan bidimensionnel.
L'interpolation subpixélique consiste, étant donné les pixels de l'image, à « imaginer » les pixels qui seraient présents si l'image avait été prise avec une résolution plus importante.

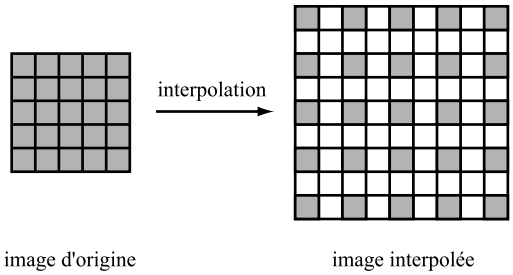


FIG. 2.14 – Interpolation au demi pixel

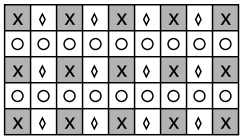


FIG. 2.15 – Interpolation bilinéaire au demi pixel

Par exemple, l'interpolation avec une précision au demi pixel revient à insérer un pixel entre deux pixels consécutifs de l'image comme illustré sur la figure 2.14.
Cette figure illustre bien le travaille à réaliser pour interpoler l'image. Bien évidemment la figure présentée ne tient compte que des niveaux de gris. Avec une image en couleur, l'interpolation est à appliquer sur les trois chrominances en mode RGB et sur la luminance et les deux chrominances en mode YUV.

2.3.2 Méthodes d'interpolation subpixélique

Nous présentons ici les différentes méthodes d'interpolation de manière synthétique. Leur implémentation a déjà été effectuée et rendue compte dans [43].

Interpolation bilinéaire

La méthode d'interpolation bilinéaire est l'une des plus simples existant mais elle reste imprécise pour des précisions dépassant le demi pixel. Elle consiste à faire la moyenne des deux pixels voisins situés à égale distance.

La figure 2.15 illustre l'interpolation d'une image au demi pixel. Cette dernière est réalisée en deux étapes :

1. L'interpolation agit premièrement sur les lignes de l'image contenant des pixels réels (marqués en gris et par un « × » sur la figure). Les pixels notés « ◊ » sont obtenus en calculant la moyenne des deux pixels « × » voisins sur la même ligne ;
2. Un parcours sur les colonnes est ensuite effectué. Les pixels notés « ○ » sont obtenus en calculant la moyenne des deux pixels voisins sur la même colonne (les voisins tous deux marqués comme « × » ou comme « ◊ »).

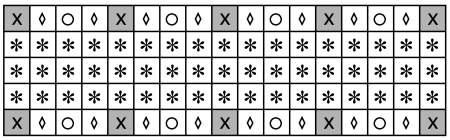


FIG. 2.16 – Interpolation bilinéaire au quart de pixel

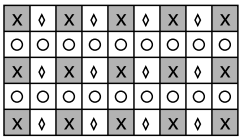


FIG. 2.17 – Interpolation H26L au quart de pixel - étape 1 : interpolation au demi de pixel

De même, nous pouvons expliquer le fonctionnement de l'interpolation au quart de pixel avec la figure 2.16.

Comme pour le demi pixel, l'interpolation au quart de pixel se déroule en deux étapes : premièrement l'interpolation en ligne (celles qui contiennent les pixels réels), puis deuxièmement l'interpolation en colonne à partir des lignes complétées. De plus, chaque interpolation nécessite deux opérations que je détaille dans le cas de l'interpolation des lignes :

1. Les pixels notés « ∘ » sont obtenus en calculant la moyenne des deux pixels « × » voisins (à une distance de deux pixels) sur la même ligne.
2. Les pixels notés « ◊ » sont obtenus en calculant la moyenne des deux pixels voisins directs sur la même ligne (« × » et « ∘ »).

L'interpolation en colonne est réalisée de la même manière en procédant en deux étapes.

Ce procédé très simple peut naturellement être étendu au huitième de pixel, voire à toute précision qui est une puissance de deux. Cependant, dès le huitième de pixel, les résultats sont assez peu satisfaisants et d'autres méthodes d'interpolation doivent être employées.

Interpolation utilisant la norme H26L

H26L est une norme de codage qui décrit des méthodes d'interpolation pour le quart et le huitième de pixel.

Considérons pour commencer l'interpolation au quart de pixel. Cette interpolation se déroule en deux étapes. Dans un premier temps, l'interpolation va se faire sur les demi pixels avec l'utilisation d'un filtre numérique.
L'algorithme effectue un parcours sur les lignes d'indice paire (précisons que la première ligne a pour indice 0 et est considérée comme paire). Tous les pixels notés « ◊ » sur la figure 2.17 sont obtenus en calculant la moyenne pondérée des six voisins pixélique (pixels gris et notés par un « × » sur la figure 2.17) situés à gauche et à droite sur la même ligne. Les pondérations sont les suivantes :

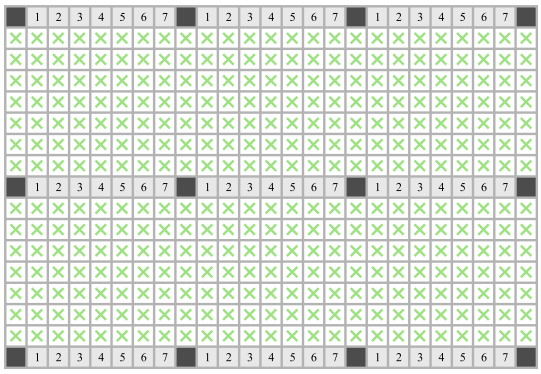


FIG. 2.18 – Interpolation H26L au huitième de pixel

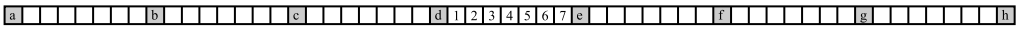


FIG. 2.19 – Représentation d'une portion de ligne

$$(\frac{1}{32}, \quad \frac{-5}{32}, \quad \frac{20}{32}, \quad \frac{20}{32}, \quad \frac{-5}{32}, \quad \frac{1}{32})$$

Lorsque tous les demi pixels des lignes d'indice paire ont été calculés, nous utilisons le même filtre en colonne pour calculer les lignes d'indice impaire.
Nous pouvons à ce stade effectuer deux remarques. Premièrement le résultat est naturellement arrondi à l'entier le plus proche contenu dans l'intervalle $[0, 255]$ généralement utilisé pour décrire la valeur d'un pixel. Deuxièmement, si le filtre déborde de l'image, le pixel situé sur le bord de l'image cumule toute les pondérations des pixels situés à l'extérieur.

L'image est à ce stade interpolée à la précision du demi pixel. La seconde étape est d'appliquer à cette image interpolée une interpolation bilinéaire au demi pixel. Nous obtenons de cette manière une image interpolée au quart de pixel.

Étudions maintenant l'interpolation au huitième de pixel. Une unique étape est nécessaire pour réaliser cette interpolation. Le principe reste le même que pour toutes les méthodes d'interpolation : l'interpolation complète dans un premier temps les lignes contenant des pixels réels, puis une interpolation sur les colonnes est réalisée à partir des lignes ainsi complétées. De même, comme lors de la première étape de l'interpolation H26L au quart de pixel, un filtre numérique est utilisé. Cependant, étant donné que sept pixels sont interpolés entre deux pixels réels (voir figure 2.18), le filtre va différer selon la position du pixel entre les pixels réels. La figure 2.19 permet de fixer les notations employées pour définir, dans le tableau 2.1, les filtres utilisés par l'algorithme. Sur cette figure les pixels à interpoler sont représentés par des numéros et les pixels réels le sont par des lettres.
Les filtres ainsi définis sont utilisés dans une seconde étape pour réaliser l'interpolation sur les colonnes. Cela permet de calculer les lignes manquantes correspondant aux croix vertes représentées sur la figure 2.18. À ce niveaux, toute l'image a alors été interpolée.

Position \ Poids	a	b	c	d	e	f	g	h
1	$\frac{-3}{512}$	$\frac{12}{512}$	$\frac{-37}{512}$	$\frac{485}{512}$	$\frac{71}{512}$	$\frac{-21}{512}$	$\frac{6}{512}$	$\frac{-1}{512}$
2	$\frac{-3}{256}$	$\frac{12}{256}$	$\frac{-37}{256}$	$\frac{229}{256}$	$\frac{71}{256}$	$\frac{-21}{256}$	$\frac{6}{256}$	$\frac{-1}{256}$
3	$\frac{-6}{512}$	$\frac{24}{512}$	$\frac{-76}{512}$	$\frac{387}{512}$	$\frac{229}{512}$	$\frac{-60}{512}$	$\frac{18}{512}$	$\frac{-4}{512}$
4	$\frac{-3}{256}$	$\frac{12}{256}$	$\frac{-39}{256}$	$\frac{158}{256}$	$\frac{158}{256}$	$\frac{-39}{256}$	$\frac{12}{256}$	$\frac{-3}{256}$
5	$\frac{-4}{512}$	$\frac{18}{512}$	$\frac{-60}{512}$	$\frac{229}{512}$	$\frac{387}{512}$	$\frac{-76}{512}$	$\frac{24}{512}$	$\frac{-6}{512}$
6	$\frac{-1}{256}$	$\frac{6}{256}$	$\frac{-21}{256}$	$\frac{71}{256}$	$\frac{229}{256}$	$\frac{-37}{256}$	$\frac{12}{256}$	$\frac{-3}{256}$
7	$\frac{-1}{512}$	$\frac{6}{512}$	$\frac{-21}{512}$	$\frac{71}{512}$	$\frac{485}{512}$	$\frac{-37}{512}$	$\frac{12}{512}$	$\frac{-3}{512}$

TAB. 2.1 – Filtres utilisés pour l'interpolation au huitième de pixel avec la norme H26L

Nous verrons dans la section 2.4 que cette méthode d'interpolation donne de meilleurs résultats que la méthode bilinéaire. Néanmoins, seules deux précisions d'interpolation sont définies dans la norme et l'interpolation au demi pixel ne l'est pas. Cela restreint quelque peu les possibilités. Dans la suite nous présentons une méthode qui permet d'interpoler à toute précision entière donnée.

Interpolation utilisant les courbes de Bézier

Les deux méthodes présentées précédemment permettent d'interpoler une images aux précision du demi, du quart et du huitième de pixel. Cependant, les qualités obtenues et la restriction du niveau de la précision ont abouti à l'étude des courbes de Bézier pour l'interpolation d'image.

Les courbes de Bézier permettent de construire des courbes complexes qui ont généralement pour conséquence l'augmentation du degré de la courbe. Pour remédier à cela, l'assemblage de plusieurs arcs de Bézier de degré fixe formant une courbe composite semble être une bonne alternative à ce problème. Cette liaison de courbes est appelée B-Spline.
Le problème qui nous intéresse peut se formuler de la manière suivante : nous disposons de $n+1$ points $\{p_0, ..., p_n\}$ que nous voulons interpoler par n arcs de degrés m. Ces arcs doivent être reliés entre eux suivant une continuité C^{m-1}. Nous considérons alors le cas particulier des splines cubiques où les arcs sont de degrés trois. La B-spline se définit alors comme suit :

$$S_i(t) = Q_{i-1}Bs^4_{i-3}(t) + Q_iBs^4_{i-2}(t) + Q_{i+1}Bs^4_{i-1}(t) + Q_{i+2}Bs^4_i(t)$$

où les Q_i sont les points de contrôle de la B-Spline et les $Bs^4_i(t)$ définissent les fonctions de base, ou fonction de pondération, des points de contrôle de la courbe B-Spline de degré trois pour chaque arc $S_i(t)$:

$$\begin{aligned}
Bs^1_i(t) &= \begin{cases} 1 & \text{si } t_i \leq t \leq t_{i+1} \\ 0 & \text{sinon} \end{cases} \\
Bs^2_i(t) &= \frac{t-t_i}{t_{i+1}-t_i}Bs^1_i(t) + \frac{t_{i+2}-t}{t_{i+2}-t_{i+1}}Bs^1_{i+1}(t) \\
Bs^3_i(t) &= \frac{t-t_i}{t_{i+2}-t_i}Bs^2_i(t) + \frac{t_{i+3}-t}{t_{i+3}-t_{i+1}}Bs^2_{i+1}(t) \\
Bs^4_i(t) &= \frac{t-t_i}{t_{i+3}-t_i}Bs^3_i(t) + \frac{t_{i+4}-t}{t_{i+4}-t_{i+1}}Bs^3_{i+1}(t)
\end{aligned}$$

où les points partagent ici trois points de contrôle.

Nous pouvons constater d'après l'expression d'un arc que les fonctions de base dépendent de la paramétrisation entre les points de passage. Cette paramétrisation n'est autre que l'expression de l'abscisse curviligne en un point quelconque de la courbe. Si tous les couples (p_i, p_{i+1}) possèdent un même $\Delta t_i = ||t_{i+1} - t_i||$, les vecteurs nodaux des points de passage se réduisent à $\overrightarrow{T} = \{t_i = i, i \in [0, ..., n]\}$. Nous pouvons reparamétrer chaque arc sur l'intervalle $[0, ..., 1]$ et rendre ainsi les fonctions de base $Bs_i^{m+1}(t)$ identiques d'un arc à l'autre. L'interpolation résultante sera donc de type B-Spline uniforme.

D'un point de vue matriciel, l'équation d'un arc s'écrit comme suit :

$$\forall i, S_i(t) = \begin{bmatrix} Bs_{i-3}^4(t) & Bs_{i-2}^4(t) & Bs_{i-1}^4(t) & Bs_{i-3}^4(t) \end{bmatrix} \cdot \begin{bmatrix} Q_{i-1} \\ Q_i \\ Q_{i+1} \\ Q_{i+2} \end{bmatrix}$$

De sorte que pour chaque arc, il existe une matrice M_i telle que :

$$\forall i, S_i(t) = \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \cdot M_i \cdot \begin{bmatrix} Q_{i-1} \\ Q_i \\ Q_{i+1} \\ Q_{i+2} \end{bmatrix}$$

Dans le cas d'une B-Spline uniforme et après avoir reparamétré chaque arc d'intervalle (t_i, t_{i+1}) sur l'intervalle $[0, 1]$, il résulte de l'uniformité du vecteur nodal que M_i est la même matrice sur chacun des arcs :

$$\forall i, M_i = M = \frac{1}{6} \begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

Pour résoudre l'interpolation de n points de passage P_i (ici les pixels de l'image) par une courbe B-Spline de $n + 2$ points de contrôle (les Q_i), il est nécessaire de fixer les premiers et derniers points de contrôle de manière à ce que le système ne possède pas plus d'inconnues que d'équations. L'une des heuristiques possible est celle des « points fantômes » :

$$\begin{cases} Q_0 = \frac{1}{2} \cdot (Q_{-1} + Q_1) \Rightarrow Q_{-1} = 2Q_0 - Q_1 \\ Q_{n-1} = \frac{1}{2} \cdot (\cdot Q_{n-2} + Q_n) \Rightarrow Q_n = 2Q_{n-1} - Q_{n-2} \end{cases}$$

d'où la définition des conditions d'interpolation aux points extrêmes :

$$\begin{cases} 6p_0 = 6Q_0 \\ 6p_{n-1} = 6Q_{n-1} \end{cases}$$

Le système d'interpolation est alors défini comme suit :

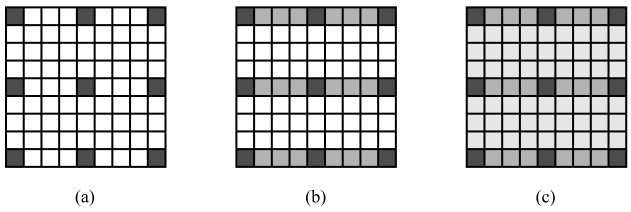


FIG. 2.20 – Interpolation B-Spline

$$\begin{bmatrix} 6 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ 1 & 4 & 1 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & 4 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & 0 & 1 & 4 & 1 \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & 6 \end{bmatrix} \cdot \begin{bmatrix} Q_0 \\ Q_1 \\ \vdots \\ \vdots \\ \vdots \\ Q_{n-2} \\ Q_{n-1} \end{bmatrix} = 6 \cdot \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ \vdots \\ \vdots \\ p_{n-2} \\ p_{n-1} \end{bmatrix}$$

L'interpolation de n points de passage par une courbe B-Spline uniforme est donc réalisée en inversant la matrice M. Nous calculons ainsi n des $n+2$ points de contrôle nécessaires :

$$M \cdot Q = 6P \text{ et donc } Q = 6M^{-1}P$$

Les deux points de contrôle extrêmes sont calculés à l'aide des points fantômes vus précédemment. Enfin, lorsque tous les points de contrôle sont connus, les arcs B-spline $S_i(t)$ sont déduits successivement.

L'application de cette méthode à l'interpolation d'image est réalisée en deux étapes. Premièrement, les lignes contenant des pixels réels (marqués en sombre, figure 2.20 (a)) sont complétées par interpolation sur les lignes (figure 2.20 (b)). Deuxièmement, l'interpolation est réalisée en colonne et cela permet de combler l'ensemble des lignes vides (figure 2.20 (c)).

La première remarque que nous pouvons faire est qu'il est possible avec cette méthode d'interpoler une image avec n'importe quelle précision ; cela reste impossible avec les deux méthodes précédentes. La seconde remarque concerne l'efficacité au sens qualitatif de l'interpolation. Nous verrons dans la section 2.4 que les résultats obtenus sont de meilleure qualité mais au détriment des temps de calcul.

2.3.3 Adaptation subpixélique des méthodes de block-matching

Principe général

Le block-matching subpixélique ne diffère que très peu du block-matching pixélique. Nous rappelons quelques idées principales pour être rigoureux. Pour cela rappelons que le block-matching fait intervenir deux images : l'image courante et l'image de référence. Dans une prédiction backward seule l'image de référence doit être interpolée et la méthode de

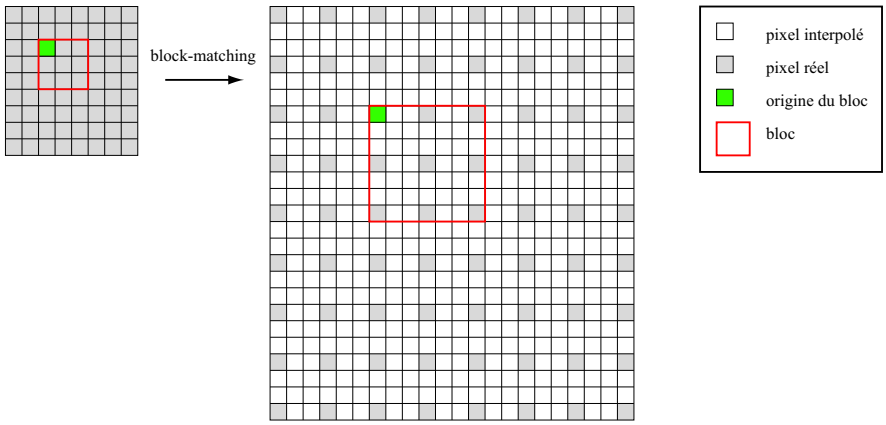


FIG. 2.21 – Interpolation au tiers de pixel - positionnement des blocs

block-matching subpixélique fait ainsi intervenir des blocs de taille différente. Ceci ne paraît pas être évident mais il s'agit de la seule option acceptable. En effet, il ne faut pas perdre de vue que l'intérêt de l'utilisation de l'interpolation est de pouvoir obtenir des vecteurs mouvements ayant des coordonnées non nécessairement entières. Les blocs considérés dans l'image courante doivent être les mêmes que ceux utilisés dans les méthodes pixéliques. Pour cette raison, cette image n'est pas interpolée. D'autre part, cela est un avantage dans la mesure où l'interpolation est coûteuse en temps de calcul. Une économie d'une interpolation est ainsi réalisée.
Maintenant que nous avons détaillé l'utilisation d'images et de blocs de tailles différentes, nous allons nous attarder sur la construction des blocs dans l'image de référence. N'oublions pas que l'algorithme de block-matching compare des blocs. Il est donc nécessaire qu'ils soient comparables. Pour cela, les blocs de l'image de référence sont construits comme indiqué sur la figure 2.21.

L'origine du bloc de référence est déduite de la position du bloc de l'image courante en multipliant par la précision de l'interpolation. Cette origine est le centre de la fenêtre de recherche, elle-même de taille multipliée par la précision, toujours pour des raisons de cohérence vis-à-vis des méthodes pixéliques. La fenêtre étant fixée, les blocs de référence sont composés des pixels de l'image de référence mais on en considère qu'un sur trois si la précision de l'interpolation est de trois. Il en résulte que les blocs de référence sont composés, en prenant l'exemple de la figure 2.21, de neuf pixels mais la taille du bloc n'est pas de 3×3 mais de 7×7. Ceci assure que les blocs comparés par l'algorithme sont composés du même nombre de pixels et qu'ils représentent des superficies similaires sur les images.
Les algorithmes de block-matching sont également adaptés pour prendre en compte la caractéristique subpixélique de la recherche (prise en compte de bloc de taille différentes, adaptation des méthodes).

Problèmes rencontrés

En étudiant plus en détail la figure 2.21, nous pouvons remarquer une information importante relative à la taille des images d'origine et interpolée. L'image d'origine a pour dimensions 9×8 et l'image interpolée 25×21. Ainsi, la taille de l'image d'origine n'est pas trivialement multipliée par la précision de l'interpolation. Cela est dû au fait qu'il n'est possible d'interpoler qu'entre deux pixels réels (indiqués par des carrés grisés sur l'image interpolée). Pour une interpolation au tiers de pixel[1], l'algorithme doit interpoler deux pixels entre deux pixels réels. Ainsi, la taille de l'image interpolée peut être donnée en fonction de la taille de l'image d'origine et de la précision utilisée :

$$\begin{aligned} h_{interpolee} &= (h_{origine} - 1) * precision + 1 \\ l_{interpolee} &= (l_{origine} - 1) * precision + 1 \end{aligned}$$

où $h_{origine} \times l_{origine}$ est la dimension de l'image d'origine et $h_{interpolee} \times l_{interpolee}$, celle de l'image interpolée.

Cette simple remarque qui peut paraître anodine nous a permis de résoudre les quelques problèmes posés lorsque nous avons adapté les algorithmes de block-matching au cas subpixélique.

La taille des blocs sur l'image interpolée doit également être considérée avec précaution. Nous avons déjà vu que la taille des blocs doit permettre de les comparer aux blocs de l'image courante. Ils possèdent une taille relative au nombre de pixels dont ils sont composés et une taille relative au recouvrement d'une partie de l'image courante. Cela pose des problèmes si ces blocs ne sont pas considérés avec une grande attention. On notera enfin que la taille de recouvrement des blocs suit la même règle que la taille de l'image interpolée.

Maintenant que le contexte de l'estimation du mouvement par block-matching ainsi que l'interpolation subpixélique ont été détaillés, nous allons présenter l'adaptation des différents algorithmes au cas subpixélique.

Adaptation du « Full Search Algorithm »

Cet algorithme est le plus simple de tous à adapter. En effet, rappelons que tous les blocs contenus dans la fenêtre de recherche sont testés de manière exhaustive. Le block-matching exhaustif équivalent en subpixélique teste de la même manière tous les blocs contenus dans la fenêtre de recherche. Cependant, nous voyons sur la figure 2.22 que cette fenêtre contient beaucoup plus de pixels entraînant ainsi un temps de recherche qui augmente sensiblement. À titre d'exemple, pour une fenêtre de recherche de taille $[-7, 7] \times [-7, 7]$, 225 blocs étaient testés en pixélique mais 841 en subpixélique avec une précision au demi pixel. Cependant, nous verrons par la suite que le gain obtenu en subpixélique est réellement significatif.

[1]Notation équivalente à « interpolation avec précision au tiers de pixel »

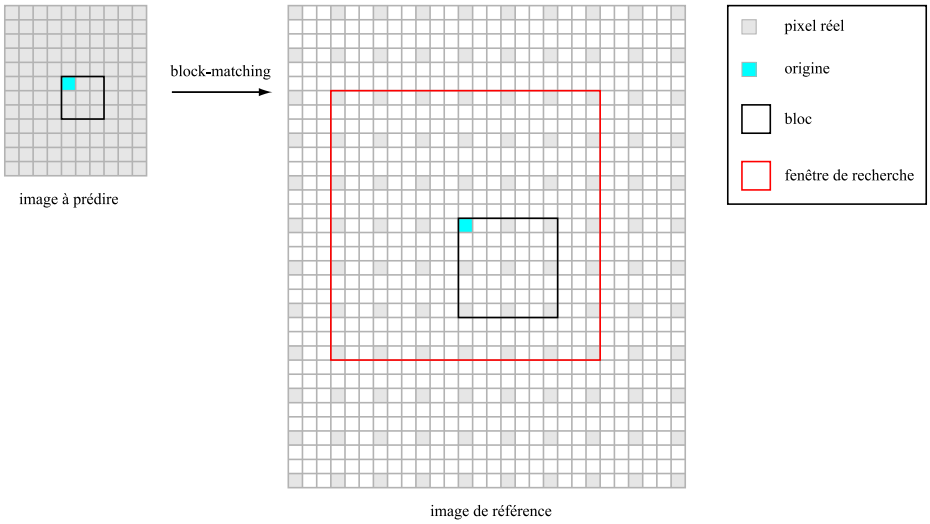


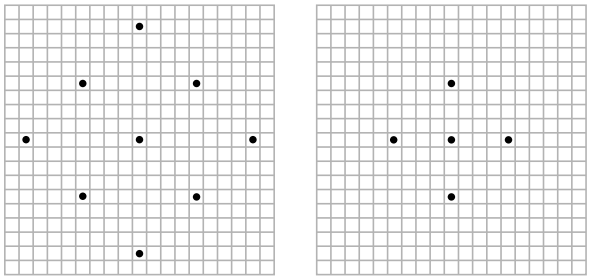
FIG. 2.22 – Block-matching subpixélique

Adaptation du « Three Step Search Algorithm »

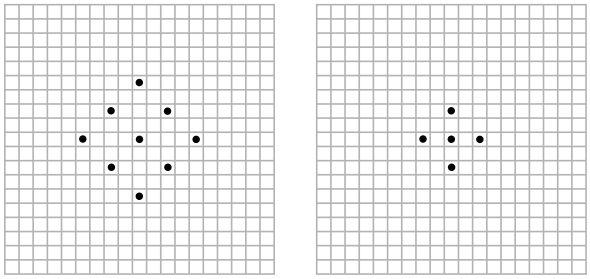
Dans la section 2.2 nous avons implémenté une méthode générale nommée « N Step Search » qui ne se restreint pas uniquement à trois pas. L'utilisation de notre méthode nous permet d'adapter l'algorithme au cas subpixélique de manière immédiate.
Dans le « N step » classique, nous définissons un déplacement initial d que nous utilisons pour définir les neufs blocs testés. Après sélection du *meilleur* bloc (au sens minimisation du critère de comparaison), l'opération est renouvelée récursivement en divisant successivement le déplacement d par deux jusqu'à ce que d soit égal à un. Pour son adaptation au cas subpixélique, il suffit de multiplier le déplacement initial par la précision et d'utiliser l'algorithme sans le modifier en prenant garde de construire les blocs comme indiqué précédemment.
On notera que cette adaptation est implicitement la succession du « N step » pixélique et de la partie strictement subpixélique. Ainsi, il est tout à fait possible que l'algorithme retourne les mêmes vecteurs que dans le cas pixélique si ces derniers sont les meilleurs qu'il puisse trouver. Cet aspect est important car cela prouve que les deux algorithmes sont bien « similaires » dans leur conception. Cet aspect a été pris en compte pour l'adaptation de tous les algorithmes assurant des résultats au moins aussi bon que les méthodes pixéliques.

Adaptation du « Diamond Search Algorithm »

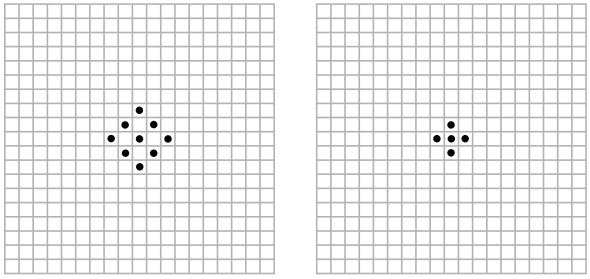
L'implémentation du « N Step Search » en subpixélique nous a permis de voir l'importance d'obtenir des résultats similaires au cas pixélique si le déplacement est réellement pixélique. De ce fait, nous proposons un algorithme subpixélique qui utilise plusieurs couples de modèles de recherche.
Rappelons que l'algorithme DS utilise dans le cas pixélique deux modèles de recherche. Si nous voulons réaliser un block-matching subpixélique de précision donnée, l'algorithme boucle sur la précision en débutant avec une précision pixélique jusqu'à la précision sou-



(a) modèles LDSP et SDSP avec précision pixélique



(b) modèles LDSP et SDSP avec précision au demi pixel

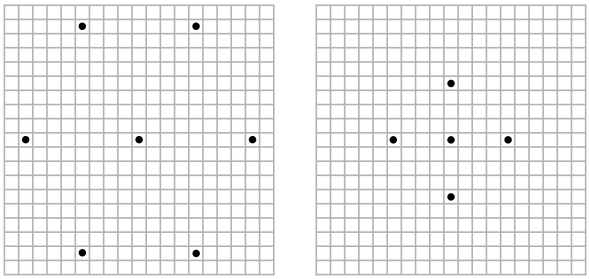


(c) modèles LDSP et SDSP avec précision au quart de pixel

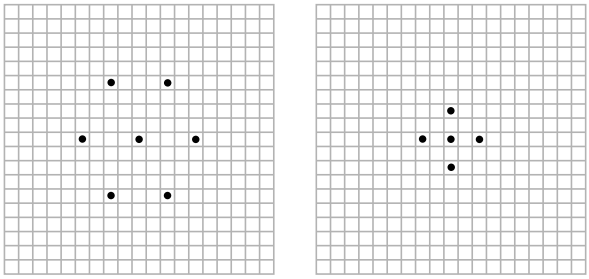
FIG. 2.23 – Modèles de recherche utilisés par l'algorithme DS subpixélique à la précision quart de pixel

haitée. À chaque itération, l'algorithme classique est utilisé avec un jeu de modèles de recherche créés en fonction de la précision.
Si une précision au quart de pixel est souhaitée, l'algorithme commence par interpoler l'image de référence au quart de pixel puis il calcule la fenêtre de recherche et enfin il itère l'algorithme classique avec les précisions au pixel, demi pixel et quart de pixel. Les modèles de recherche utilisés pour chaque précision sont donnés figure 2.23.

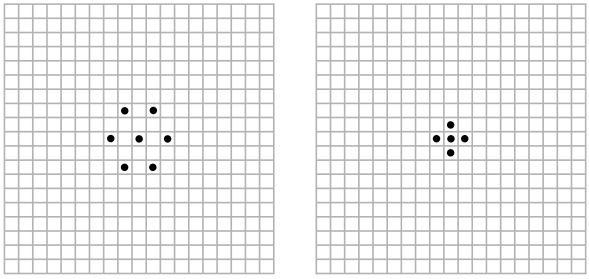
Il est naturellement possible de générer automatiquement les deux modèles de recherche LDSP et SDSP en fonction de la précision. Ainsi, deux modèles génériques ont servi à construire les six modèles présentés sur cette figure. Cette généralisation permet de contrôler les erreurs et de rendre l'algorithme plus robuste.



(a) modèles LHSP et SHSP avec précision pixélique



(b) modèles LHSP et SHSP avec précision au demi pixel



(c) modèles LHSP et SHSP avec précision au quart de pixel

FIG. 2.24 – Modèles de recherche utilisés par l'algorithme HEXBS subpixélique à la précision quart de pixel

Adaptation du « Hexagon-Based Search Algorithm »

Rappelons que le block-matching pixélique « Hexagon-Based Search » est une modification (au niveau la forme des modèles de recherche) du « Diamond Search ». De la même manière, la version subpixélique de l'algorithme « Hexagon-Based Search » est un remaniement du « Diamond Search » subpixélique.

Les modèles employés pour une précision au quart de pixel sont illustrés figure 2.24. Chaque couple de modèles (LHSP,SHSP) est utilisé pour un niveau de précision donnée. Enfin, il existe également des modèles génériques permettant d'engendrer les modèles utilisés à chaque niveau de précision.
Les remarques par rapport à la robustesse de l'algorithme avec l'utilisation de modèles génériques sont naturellement identiques aux remarques faites précédemment.

Dans cette section nous avons présenté l'estimation de mouvement par block-matching. De plus nous avons vu que la présence de nombreux paramètres comme par exemple le critère de comparaison, la méthode de block-matching, la précision de l'interpolation, permettent d'envisager différentes approches du problème. Pour clarifier l'ensemble du procédé, nous allons évaluer dans la partie suivante l'ensemble des paramètres mis en œuvre.

2.4 Évaluation

Pour estimer le mouvement, nous avons vu que quatre paramètres sont mis en jeu : le critère de comparaison, la méthode de recherche, la précision et la méthode d'interpolation.

L'analyse de chacun des paramètres doit être réalisée individuellement. Trois sont fixés et le quatrième est testé en prenant diverses valeurs. Définissons alors clairement les conditions de tests et la valeur des critères d'évaluation quand ces derniers sont fixes.

2.4.1 Conditions d'évaluation

Il est important de bien définir l'ensemble des conditions d'évaluation car les résultats obtenus dépendent directement de ces dernières. Les négliger poserait des problèmes de validation de nos résultats.

Matériel informatique et logiciels

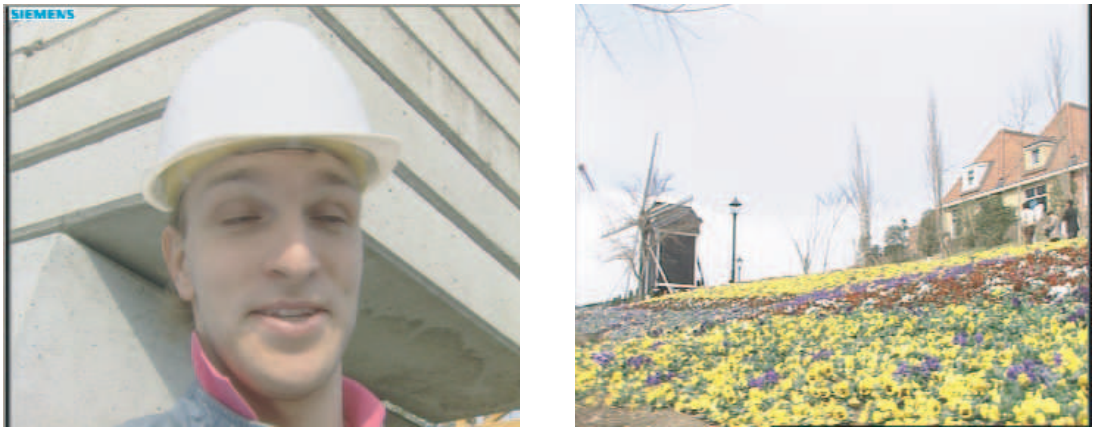
L'estimateur de mouvement a été réalisé en C++ à l'aide de Visual C++. L'ordinateur utilisé est un PC 2GHz avec 1Go de mémoire vive et équipé de Windows XP Professionnel.

Séquences de test

Afin de réaliser une évaluation complète, nous mettons en œuvre différents types de tests. En particulier, nous utilisons des séquences vidéo couramment employées : les séquences « Foreman » et « Flower and Garden » présentées sur la figure 2.25. Ces séquences se complètent par les mouvements qu'elles contiennent. En effet, la vidéo de « Foreman » représente un personnage centré sur la scène avec un fond fixe et dont le visage bouge. La vidéo « Flower and Garden » est une translation latérale de caméra filmant une scène où différents objets sont placés à différentes profondeurs relativement à la caméra. La conséquence de cette translation est que les différents objets ont des mouvements apparents différents : rapides pour les objets proches et quasi-nul pour les objets lointains. De plus, cette vidéo est particulièrement intéressante dans la mesure où elle contient des objets extrêmement texturés (e.g. les fleurs) ainsi que des objets homogènes (e.g. le ciel). Enfin, cette séquence possède également des personnages en mouvement compliquant encore davantage le mouvement global de la vidéo.

Paramètres utilisés par défaut

Pour comparer les différents paramètres mis en jeu, nous ne faisons varier qu'un seul paramètre, les autres sont fixés à des valeurs prédéfinies qui sont les suivantes :



(a) (b)

FIG. 2.25 – Séquences de test - (a) Foreman (b) Flower and Garden

- critère de comparaison : SSD ;
- méthode de block-matching : full search.
- précision d'interpolation : demi pixel ;
- méthode d'interpolation : B-Spline ;

Par ailleurs, d'autres variables sont utilisées mais nous n'étudierons pas leur influence ici. Ces variables sont les suivantes et ont pour valeur :

- taille des blocs des images : [16pixels×16pixels] ;
- taille des blocs de comparaison : [16pixels×16pixels] ;
- déplacement horizontal et vertical maximal autorisé : 7 pixels.

2.4.2 Critères d'évaluation

Nous utilisons pour comparer les différents paramètres plusieurs critères d'évaluation couramment utilisés dans les articles traitant de l'estimation du mouvement. Ces critères peuvent être séparés en deux catégories : les critères **qualitatifs** et les critères de **complexité**.

Mean Square Error - MSE

Ce critère donne une représentation de la qualité de l'estimation en calculant l'erreur quadratique moyenne de la luminance entre l'image prédite I_p et l'image d'origine I_o que l'algorithme cherche à prédire. Si nous considérons que les images ont pour dimension $[h \times l]$ (avec h la hauteur et l la largeur de l'image) et que les indices des pixels sont contenus dans $[1..h] \times [1..l]$, alors la formule du calcul de la MSE est :

$$MSE = \frac{\sum_{i=1}^{h} \sum_{j=1}^{l} [I_o(i,j,1) - I_p(i,j,1)]^2}{l \cdot h}$$

où $I_o(i,j,1)$ est la valeur d'intensité du pixel (i,j) de l'image I_0. Naturellement plus la MSE est faible, plus l'estimateur est efficace en terme de qualité, et inversement.

Nous utilisons également lors de l'étude des critères de comparaison une version de la MSE prenant en compte la couleur. Comme dans le cas de la SSD, nous sommons simplement les MSE en luminance et en chrominance pour obtenir :

$$MSE_{couleur} = \frac{\sum_{c=1}^{3}\sum_{i=1}^{h}\sum_{j=1}^{l}[I_o(i,j,c) - I_p(i,j,c)]^2}{3 \cdot l \cdot h}$$

Ce critère n'est pas utilisé dans la littérature, pas plus d'ailleurs que les critères de mise en correspondance couleur. Cependant, au vu des résultats obtenus, il nous a semblé intéressant de les présenter.

Peak Signal to Noise Ratio - pSNR

Ce critère est également très couramment utilisé pour estimer la qualité de l'estimateur. Sa formule se déduit de la MSE définie précédemment :

$$pSNR = 10 \cdot log_{10}(\frac{255^2}{MSE})$$

Par déduction d'après les remarques précédentes, la qualité de l'estimateur augmente avec le pSNR et inversement.

De même que pour la MSE, nous utilisons également une version couleur du pSNR :

$$pSNR_{couleur} = 10 \cdot log_{10}(\frac{255^2}{MSE_{couleur}})$$

Nombre moyen de blocs testés par bloc

Nous avons vu que la stratégie des différentes méthodes d'estimation consiste à trouver pour chaque bloc de l'image à prédire le bloc minimisant le critère de comparaison tout en tentant d'en tester le moins possible. Une bonne manière de se rendre compte de l'efficacité de l'algorithme est d'étudier le nombre moyen de blocs testés.

Temps de calcul

Nous considérons deux temps de calcul qui vont comparer la complexité de l'interpolation et de l'estimateur. Ces temps sont essentiels selon l'application ciblée par le codeur (e.g. codeur temps-réel).

Les différents critères d'évaluation étant maintenant définis, nous allons étudier l'influence des paramètres de l'estimateur.

2.4.3 Critères de comparaison

Les critères de comparaison généralement employés sont la SSD et plus récemment la valeur absolue (AV). Nous avons implémenté trois critères (SSD, ZNSSD et AV) ainsi que leur variante prenant en compte les chrominances. Nous allons maintenant étudier les résultats obtenus avec chacun d'entre eux.

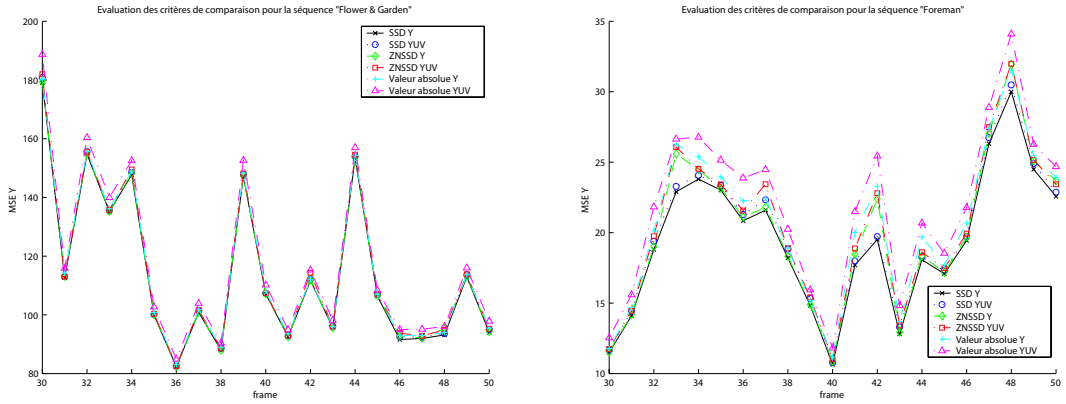


FIG. 2.26 – Évaluation des critères de comparaison. Variation de la MSE en luminance en fonction des images

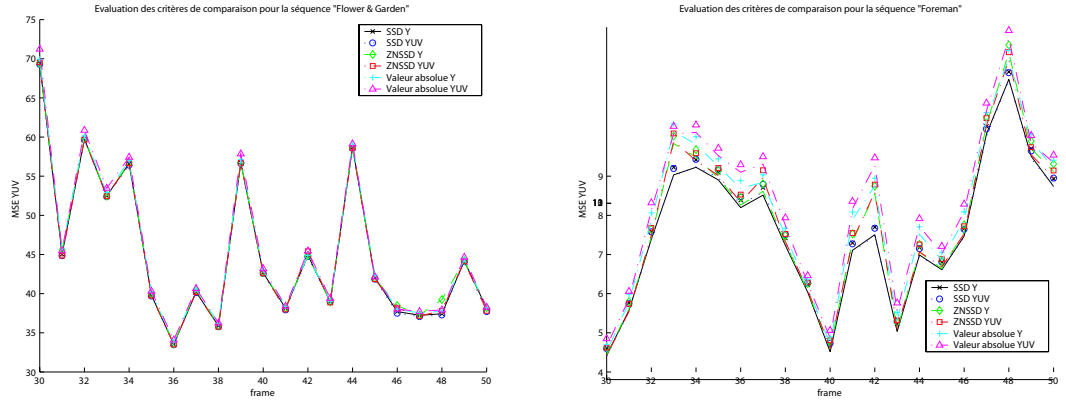


FIG. 2.27 – Évaluation des critères de comparaison. Variation de la MSE en couleur en fonction des images

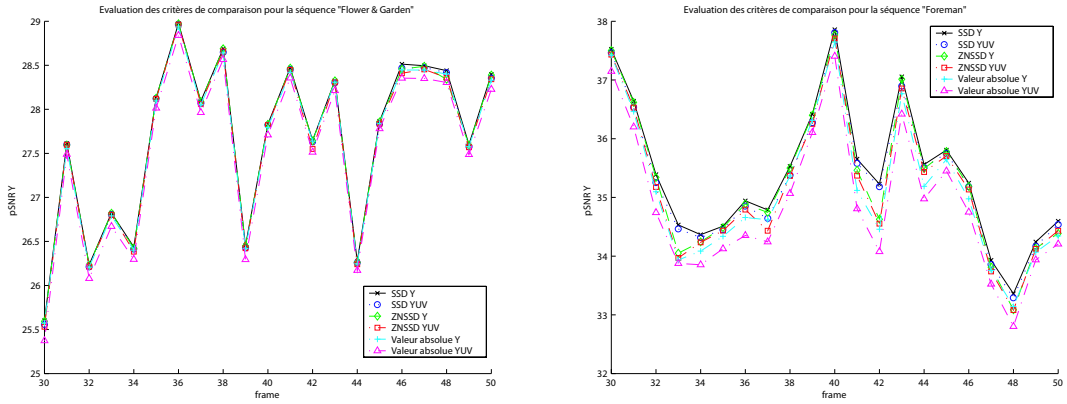


FIG. 2.28 – Évaluation des critères de comparaison. Variation du pSNR en luminance en fonction des images

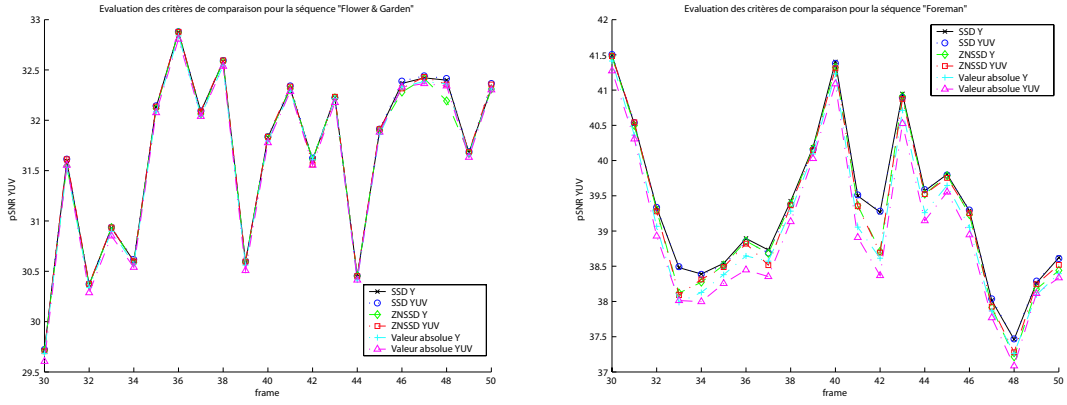


FIG. 2.29 – Évaluation des critères de comparaison. Variation du pSNR en couleur en fonction des images

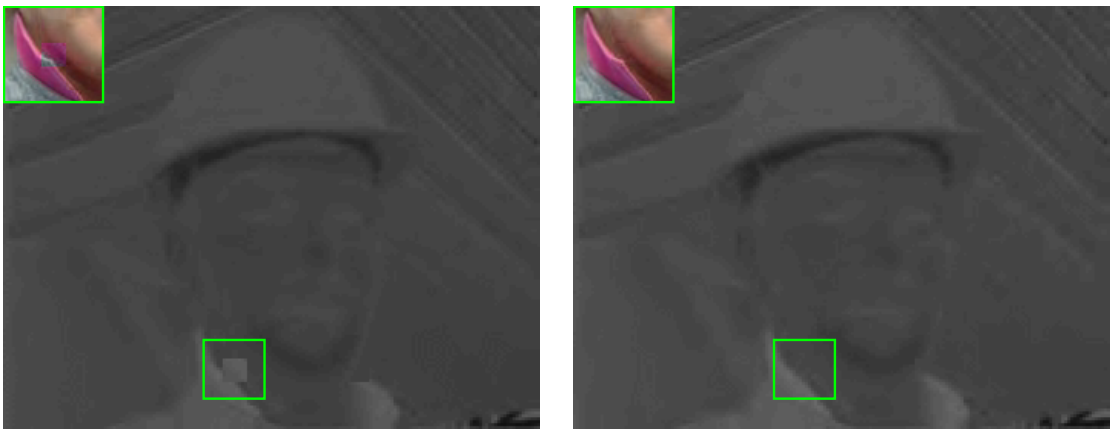
Évaluation \ Critères	SSD Y	SSD YUV	ZNSSD Y	ZNSSD YUV	AV Y	AV YUV
MSE Y	**114.18**	114.81	114.49	115.23	115.15	117.90
MSE YUV	45.22	**45.08**	45.40	45.21	45.54	45.80
PSNR Y (dB)	**27.663**	27.639	27.650	27.623	27.625	27.527
PSNR YUV (dB)	31.666	**31.680**	31.646	31.666	31.634	31.613
Nombre de blocs visités	706.79	706.79	706.79	706.79	706.79	706.79
Durée block-matching (s)	**13.23**	14.64	14.82	22.36	13.90	16.98

TAB. 2.2 – Qualité moyenne de l'estimation en fonction du critère de comparaison pour la séquence « Flower and Garden »

On peut noter tout d'abord que sur la séquence « Foreman », les différences sont plus importantes entre les critères mais certaines propriétés intéressantes sont illustrées avec la séquence « Flower and Garden ». Le critère couramment employé dans les articles est le pSNR et nous utiliserons dans toute la suite la version luminance comme référence pour les évaluations. Nous pouvons alors classer les critères du moins performant au meilleur : AV couleur, ZNSSD couleur, AV, SSD couleur, ZNSSD, SSD. Cet ordre est légèrement différent avec la séquence « Foreman » : AV couleur, AV, ZNSSD couleur, ZNSSD, SSD couleur, SSD. Ces résultats sont cohérents dans la mesure où la MSE et le pSNR sont directement liés au critère SSD. En effet, en analysant plus en détails la formule on peut se rendre compte que la MSE n'est autre que la SSD moyennée par rapport au nombre de pixels. Les critères de comparaison et de qualité étant les mêmes, il est naturel de trouver les meilleurs résultats pour le critère SSD. Le critère ZNSSD est également lié à la MSE et au pSNR mais de façon moins directe et de fait les résultats sont moins significatifs. Enfin, le critère valeur absolue n'est plus lié aux deux critères d'évaluation ; cela explique sa faible "efficacité". Si nous avions pris comme critère de qualité la valeur absolue, il semble que ce même critère aurait été le meilleur en tant que critère de comparaison. Les résultats obtenus sont néanmoins acceptables avec moins de 0.1 dB de perte par rapport au critère SSD sur la séquence « Flower and Garden » et moins de 0.5 dB avec la séquence « Foreman ».
Si nous considérons une évaluation des critères ne prenant en compte que la luminance, un critère de comparaison classique présente toujours de meilleurs résultats que sa version couleur. En revanche, avec l'évaluation prenant en compte la couleur, les écarts diminuent sensiblement et, dans certains cas, ils peuvent s'inverser (SSD et ZNSSD). Qualitativement les implémentations couleur sont identiques parfois même plus efficaces que les critères luminance. L'intérêt de l'utilisation des critères couleurs est visible sur la figure 2.30.

Nous pouvons y remarquer qu'un bloc qui semblait proche au niveau luminance cause un dégradation forte au niveau visuel. Ce type d'artefacts est résolu dès lors de l'utilisation d'un critère couleur (figure 2.30(b)).

Nous allons maintenant étudier la complexité des critères. Le nombre de blocs visités en moyenne est naturellement identique pour la simple raison que le critère de comparaison n'influe pas sur la stratégie de recherche. En revanche, les temps de calcul diffèrent légèrement et cela est dû à la complexité du critère. En effet, la SSD, critère le plus simple, est



(a) critère luminance (b) critère couleurs

FIG. 2.30 – Évaluation des critères de comparaison. Intérêt des critères couleurs

le plus rapide à calculer. En revanche, la ZNSSD est plus rapide que la valeur absolue car cette dernière nécessite l'évaluation d'un booléen coûteux en temps de calcul. Par ailleurs l'implémentation couleur reste plus lente car elle complexifie encore davantage les calculs. Pour toutes ces raisons, nous utilisons les critères SSD et SSD couleur plus efficaces et plus rapides.

2.4.4 Parcours du voisinage

Le choix de la méthode de block-matching est essentiel. La qualité et la rapidité de l'estimation du mouvement en dépendent. Nous présentons dans cette section l'évaluation des différents algorithmes présentés.

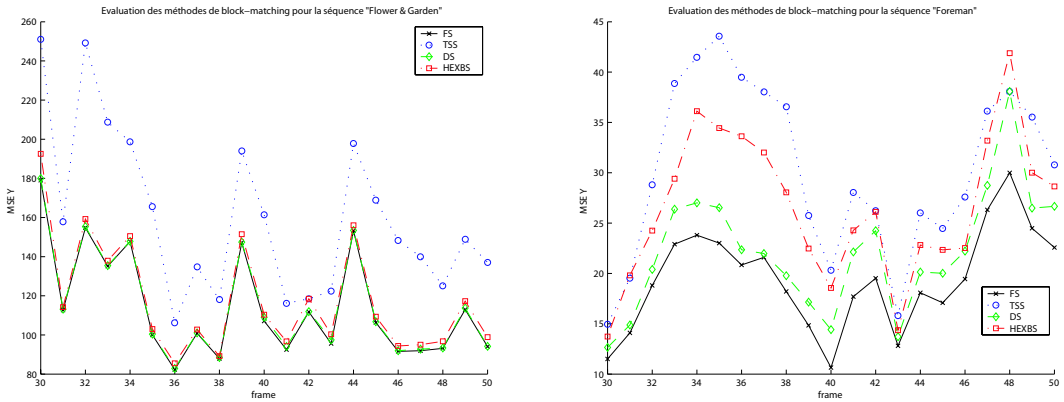


FIG. 2.31 – Évaluation de la méthode de block-matching : Variation du pSNR en fonction des images.

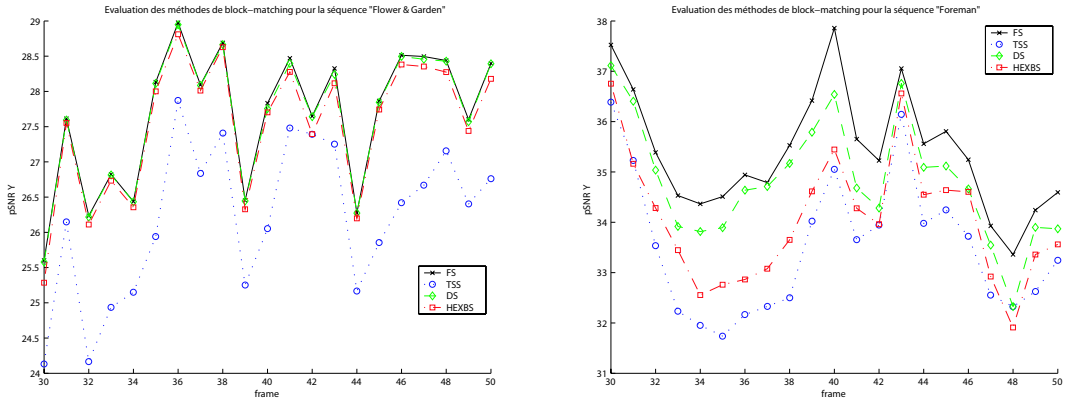


FIG. 2.32 – Évaluation de la précision de l'interpolation : Variation de la MSE en fonction des images.

Si nous classons les différentes méthodes de block-matching selon le pSNR, nous obtenons par ordre décroissant : « Full Search », « Diamond Search », « Hexagon-Based Search » et « Three-Step Search ». Il est évident que toute méthode rapide ne peut surpasser qualitativement la méthode exhaustive. Les figures 2.31, 2.32 et le tableau 2.4.4 montrent que les pertes sont de 1.88 dB pour le « Three-Step Search », de 1.33 dB pour le « Hexagon-Based Search » et de seulement 0.56 dB pour le « Diamond Search ». Cette dernière méthode est ainsi la méthode rapide qui donne qualitativement les meilleurs résultats. On notera que ces données sont relatives à la séquence « Foreman ». Pour la séquence « Flower and Garden », les résultats sont moins significatifs mais le DS reste la meilleure méthode après le FS et le TSS la moins performante.

Méthode BM / Évaluation	FS	TSS	DS	HEXBS
MSE Y	**19.44**	30.29	**22.19**	26.60
PSNR Y (dB)	**35.38**	33.50	**34.82**	34.05
Nombre de blocs visités	760.9	33.8	**29.5**	**23.5**
Durée block-matching (s)	13.225	0.642	**0.561**	**0.477**

TAB. 2.3 – Qualité moyenne de l'estimation en fonction de la méthode de block-matching pour la séquence « Foreman »

Si nous étudions maintenant la rapidité de la recherche et le nombre moyen de blocs testés, nous remarquons que la différence entre les algorithmes rapides et l'algorithme exhaustif est considérable. En effet, le FS visite 760.9 blocs en 13.225 secondes alors que, par exemple, le HEXBS n'en visite que 23.5 en 0.477 secondes soit un rapport de près de 32. De plus, nous remarquons également que les algorithmes récents HEXBS et DS sont plus rapides et visitent moins de blocs que le TSS qui date de 1981. Naturellement, le nombre de blocs visités et la durée du block-matching sont liés et nous pouvons classer les algorithmes par ordre croissant en rapidité : FS, TSS, DS et HEXBS.

Le choix de la méthode dépend aussi de l'application visée par l'estimation. Pour une estimation de bonne qualité, l'algorithme FS donne les meilleurs résultats mais au prix d'un temps de calcul plus élevé. En revanche, pour une estimation ayant des contraintes de temps, le DS semble être une bonne alternative car il n'affecte la qualité de l'estimation que de manière raisonnable tout en diminuant grandement les temps de recherche.

2.4.5 Précision de l'interpolation

La précision de l'interpolation généralement employée est une puissance de deux. Cette limitation est en partie due à l'utilisation des différentes méthodes d'interpolation. Pour évaluer l'influence de l'interpolation sur l'estimation du mouvement, nous avons ainsi testé l'interpolation au pixel, au demi, au quart et au huitième de pixel. Les résultats obtenus sont les suivants.

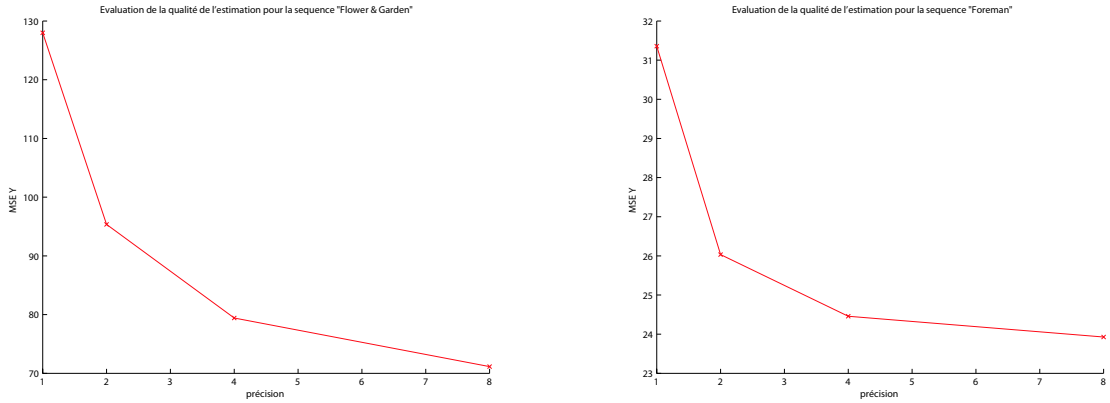


FIG. 2.33 – Évaluation de la précision de l'interpolation : Variation de la MSE en luminance en fonction de la précision

Évaluation \ Précision	1/1	1/2	1/4	1/8
MSE Y	145.57	110.42	93.98	**87.71**
PSNR Y (dB)	26.5	27.7	28.4	**28.7**
Nombre de blocs visités	**204.3**	760.9	2934.5	11522.3
Durée block-matching (s)	**3.6**	13.2	54.6	227.1
Durée interpolation (s)	**0**	30.2	48.6	86.7

TAB. 2.4 – Qualité moyenne de l'estimation en fonction de la précision pour la séquence « Flower and Garden »

La qualité de l'estimation est significativement augmentée avec la précision de l'interpolation indiquant ainsi que le déplacement est subpixélique. Cela se vérifie pour toute précision et sur toute une séquence d'images. Comme nous pouvons le voir sur le tableau 2.4.5, ce gain est en moyenne de 1.2 dB pour l'interpolation au demi pixel et de 2.2 au

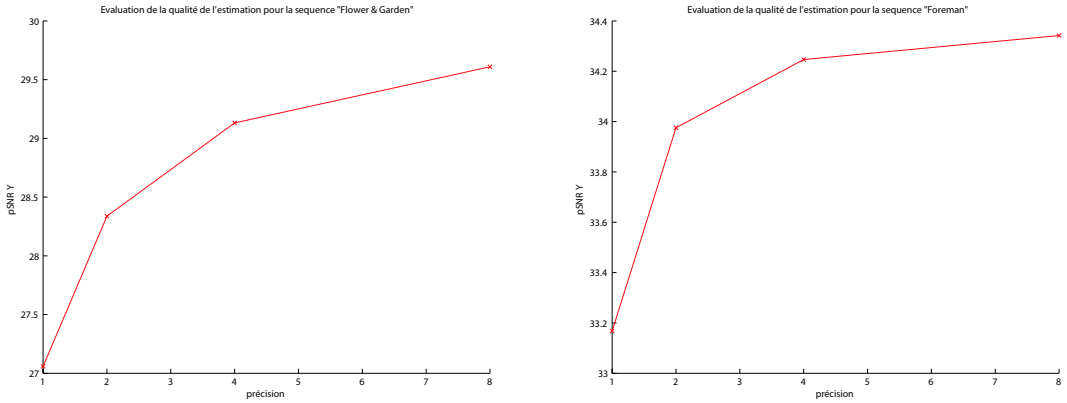


FIG. 2.34 – Évaluation de la précision de l'interpolation : Variation du pSNR en luminance en fonction de la précision

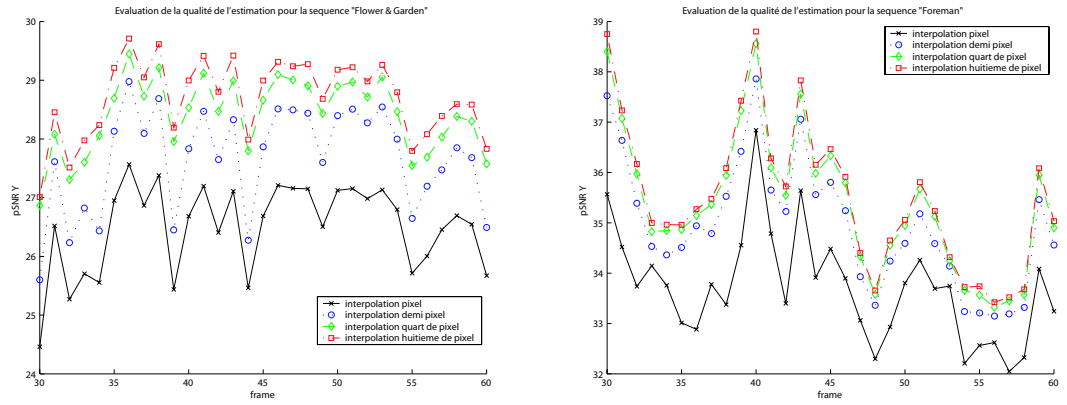


FIG. 2.35 – Évaluation de la précision de l'interpolation : Variation du pSNR en luminance en fonction des images

huitième de pixel. Cependant, les figures 2.33 et 2.34 indiquent que ce gain semble être de moins en moins important entre deux précisions successives. Il semble avoir un comportement logarithmique en fonction de la précision. Étudions maintenant l'impact de l'interpolation au niveau de la complexité.

D'après les figures 2.36 et 2.37, la durée du block-matching et le nombre de blocs visités moyen augmente de manière quasiment quadratique. De plus, le temps nécessaire à l'interpolation (voire figure 2.38) s'ajoute à l'algorithme d'estimation de mouvement et il peut être très important selon la méthode d'interpolation choisie.
Ainsi, nous choisissons l'interpolation au demi ou au quart de pixel afin d'augmenter sensiblement le pSNR sans toutefois trop pénaliser l'estimation en temps de calcul.

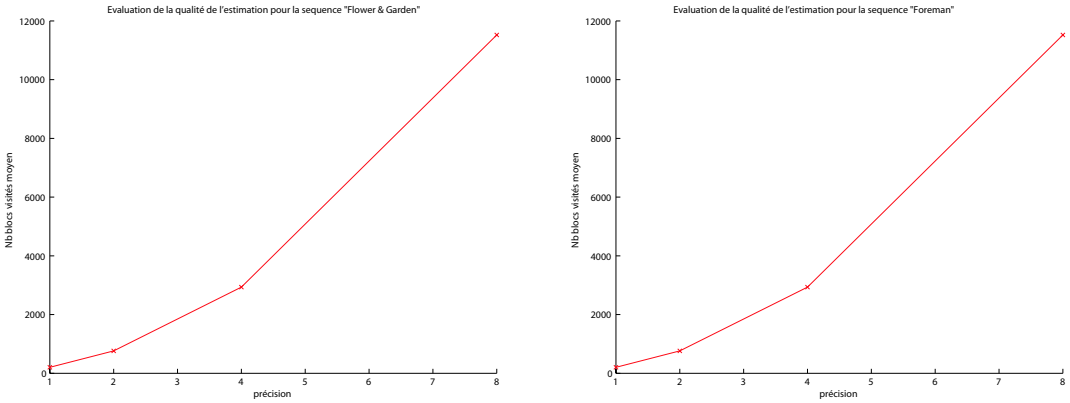


FIG. 2.36 – Évaluation de la précision de l'interpolation : Variation du nombre de blocs testés moyen en fonction de la précision

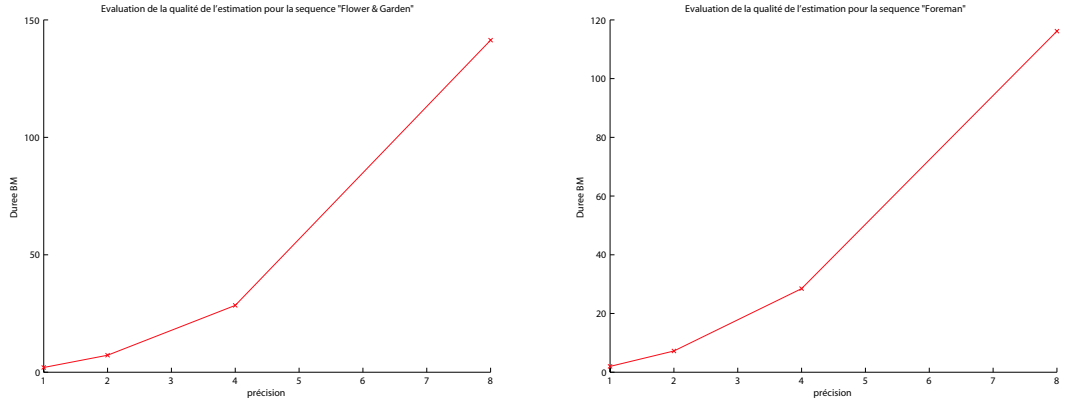


FIG. 2.37 – Évaluation de la précision de l'interpolation : Variation de la durée du block-matching en fonction de la précision

2.4.6 Méthode d'interpolation

L'interpolation et le block-matching subpixélique permettent de gagner plusieurs décibels. Il reste néanmoins à évaluer les différentes méthodes d'interpolation. Cette évaluation pose un problème, sans réelle gravité, car il est indispensable de comparer des données comparables. L'interpolation H26L ne permet pas l'interpolation au demi pixel. Pour cette raison, nous utilisons une interpolation au quart de pixel pendant toute l'évaluation.

Les figures 2.39 et 2.40 ainsi que le tableau 2.4.6 indiquent que la méthode utilisant les B-Splines donne les meilleurs résultats. Cependant, la perte observée n'est que de 0.16 dB pour l'interpolation bilinéaire et de seulement 0.04 dB pour H26L. Si nous observons maintenant les temps nécessaires à l'interpolation avec les diverses méthodes, nous nous rendons compte que la différence la plus significative se situe au niveau de la rapidité d'exécution. En effet, l'interpolation utilisant les B-Splines est 49 fois plus lente que l'interpolation bilinéaire et 42 fois plus que H26L. Cette différence est dûe à la complexité de

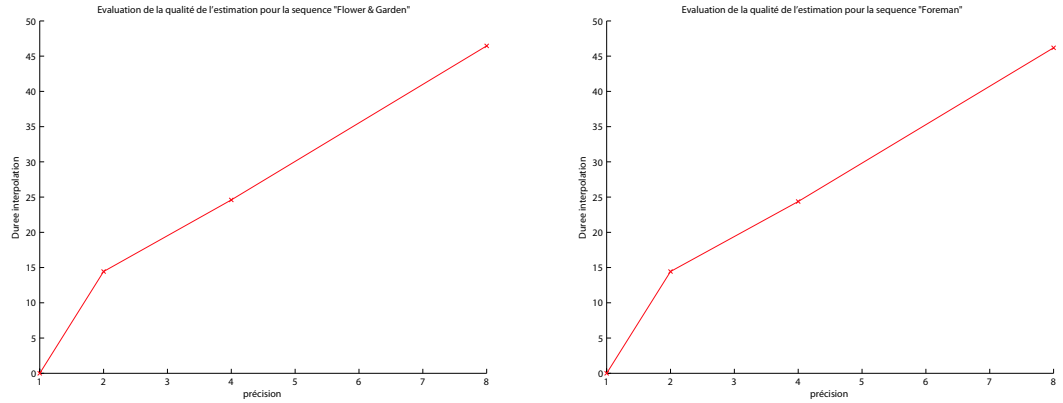


FIG. 2.38 – Évaluation de la précision de l'interpolation : Variation de la durée d'interpolation en fonction de la précision

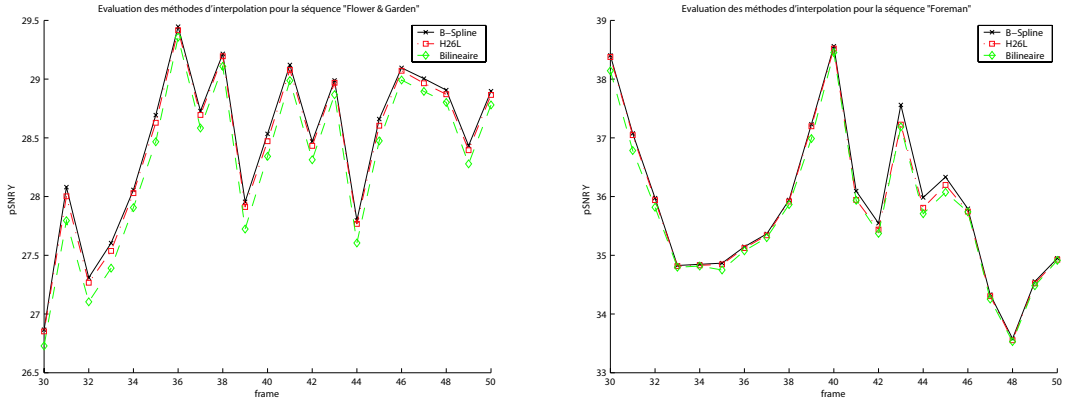


FIG. 2.39 – Évaluation de la précision d'interpolation : Variation de la MSE en fonction des images.

Évaluation \ Interpolation	Bilinéaire	H26L	B-Spline
MSE Y	97.23	94.48	**93.61**
PSNR Y (dB)	28.31	28.43	**28.47**
Durée interpolation (s)	**0.98**	**1.15**	48.50

TAB. 2.5 – Qualité moyenne de l'estimation en fonction de la méthode d'interpolation pour la séquence « Foreman »

la méthode B-Spline. En revanche, la différence entre H26L et la méthode bilinéaire n'est que de 17 centièmes de seconde ce qui reste tout à fait minime compte tenu du fait que cela permet de gagner 0.12 dB.

Le choix de la méthode d'interpolation dépend aussi de l'application visée. Ainsi la

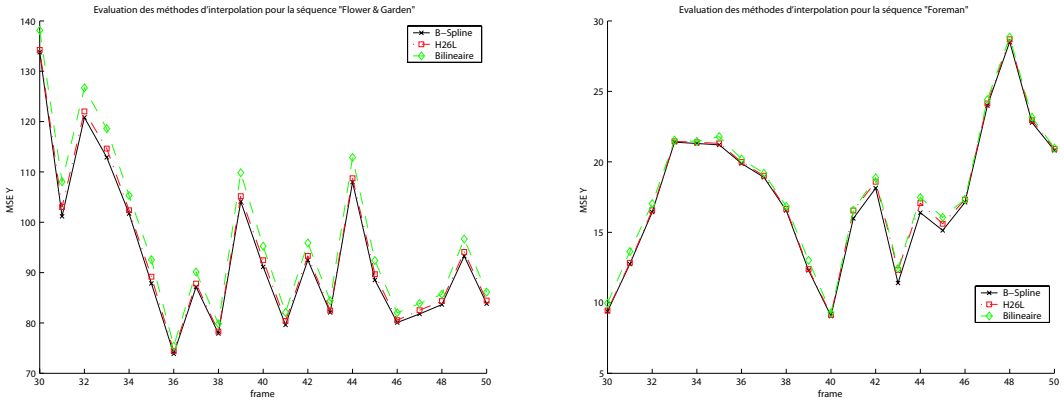


FIG. 2.40 – Évaluation de la méthode d'interpolation : Variation du pSNR en fonction des images.

méthode qui utilise les courbes de Bézier donne les meilleurs résultats. Cependant, le gain apporté n'est que relativement acceptable si l'on considère que le temps de calcul est multiplié par un coefficient d'environ quarante en comparaison des deux autres algorithmes. En revanche, pour des temps de calculs quasiment identiques, l'interpolation H26L est qualitativement la méthode la plus acceptable.

2.5 Conclusion

L'estimation de mouvement est un problème couramment rencontré en traitement d'images et de vidéos. De nombreuses techniques existent et nous nous sommes attachés à étudier celles qui utilisent une mise en correspondance de blocs. À l'intérieur même des méthodes de block-matching existent de nombreuses méthodes agissant sur différents aspects de la recherche : la stratégie de recherche, les critères de comparaison et l'interpolation d'images pour obtenir des déplacements subpixéliques. L'implémentation des diverses méthodes ont permis de les comparer entre elles et d'exhiber un ensemble de méthodes adaptées au codeur développé par l'équipe.
Nous avons remarqué qu'un choix devait être réalisé. Si nous privilégions la qualité (critère SSD couleur, méthode de recherche exhaustive, précision au huitième de pixel obtenue avec une interpolation par B-Splines), nous pénalisons grandement les temps de calcul. En revanche, l'utilisation d'une méthode « Diamond Search » au quart de pixel avec interpolation utilisant la norme H26L permet de diminuer de façon considérable les temps de calcul mais en obtenant une qualité moindre. Cependant, la qualité obtenue est très acceptable et est meilleure que la plupart des estimations du mouvement implémentées jusqu'à lors. En effet, nous avons ainsi testé ces deux possibilités pour l'estimation de la séquence « Flower and Garden » et nous avons obtenu les résultats suivants. Pour la méthode privilégiant la qualité, l'estimation est réalisée en 313 secondes et donne un pSNR de 28.067dB. Pour la méthode optimisant la rapidité et la qualité, l'estimation est réalisée en 2.04 secondes pour un pSNR de 27.709dB. La perte de qualité est de 0.358dB. Néanmoins, le temps d'exécution est divisé par un facteur 150 ce qui est considérable. Mais cela ne change

pas le résultat. Le type d'application et la qualité recherchée vont discriminer l'un des deux choix. Dans notre cas, nous utilisons la méthode rapide pour les différents tests et la méthode qualitative pour valider les résultats.

Chapitre 3

Modèle de mouvement linéaire pour l'estimation du mouvement

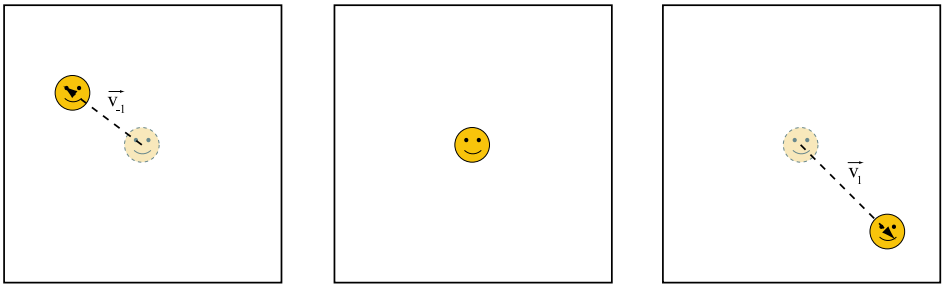
3.1 Principe

L'estimation du mouvement est utilisée pour la compensation du mouvement dans la transformée temporelle. Centrée sur une image, cette transformée utilise des images situées avant et après dans le domaine temporel. Le nombre de ces images dépend directement de la longueur du filtre utilisé pour le calcul de la transformée temporelle. Nous utilisons en pratique un filtre de longueur trois. En d'autres termes, si la transformée est centrée sur l'image n, les images $n-1$, n et $n+1$ sont employées pour le calcul. Deux estimations de mouvement de vecteurs sont alors nécessaires (voir figure 3.1 (a) et (b)) : une estimation entre les images n et $n-1$ et une autre entre les images n et $n+1$.
En conséquence, deux champs de vecteurs sont créés. En rappelant que l'objectif du projet est de créer un codeur vidéo, on se rend compte que ces vecteurs sont insérés dans le flux binaire de la vidéo, et qui plus est sans avoir subi la moindre perte. Ils peuvent représenter 5% du débit total de la vidéo à haut débit (2Mbits par seconde) et jusqu'à 30% à bas débit (500 Kbits par seconde).
L'idée de l'utilisation d'un modèle de mouvement linéaire pour l'estimation de mouvement (nous nommons ceci la contrainte bidirectionnelle) provient de l'hypothèse de linéarité du mouvement interne dans une vidéo. En effet, sur un suffisamment petit nombre d'images, le mouvement des objets est globalement linéaire. L'hypothèse semble raisonnable dans notre cas.

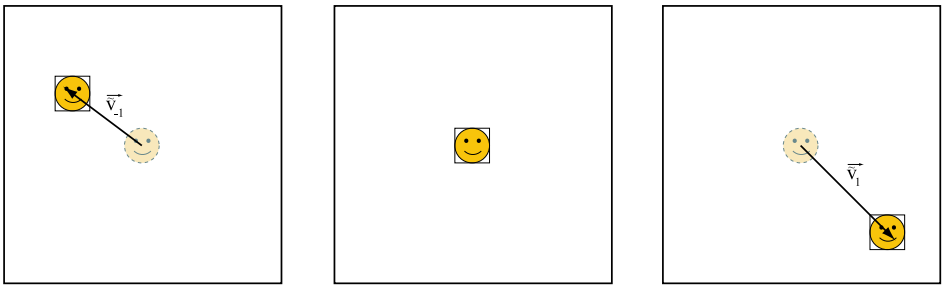
Le principe de l'estimation avec contrainte est extrêmement simple. Trois images sont considérées (les images aux temps $n-1$, n et $n+1$, voir figure 3.1 (c)) et on force l'égalité suivante :

$$\overrightarrow{\tilde{v}_1} = -\overrightarrow{\tilde{v}_{-1}}$$

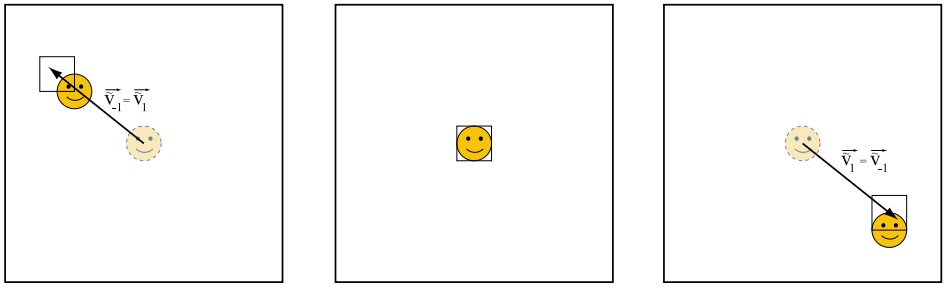
avec $\overrightarrow{\tilde{v}_1}$ le vecteur mouvement d'un bloc b entre les images n et $n+1$ et $\overrightarrow{\tilde{v}_{-1}}$ le vecteur mouvement du même bloc b entre les images n et $n-1$.
Le nom de "contrainte bidirectionnelle" prend ici tout son sens. Les champs de vecteurs ainsi générés sont nécessairement opposés. Ainsi, seul un des deux champs est suffisant



(a) Mouvement réel dans la vidéo



(b) Estimation du mouvement sans contrainte bidirectionnelle



(c) Estimation du mouvement avec contrainte bidirectionnelle

FIG. 3.1 – Estimation du mouvement avec et sans contrainte bidirectionnelle

dans le flux binaire dans la mesure où le second est obtenu en prenant son opposé.

Le but n'est pas ici de diminuer le débit total de la vidéo puisque ce dernier est fixe. L'objectif est de diminuer d'un facteur deux celui des vecteurs. Le débit ainsi économisé peut être alloué aux sous-bandes qui sont codées avec pertes. L'intérêt de la contrainte est clairement réservée au codage bas-débit permettant d'améliorer la qualité du codage des sous-bandes. En revanche, son utilité est limitée à haut débit dans la mesure où le gain obtenu est très faible.

En résumé, la contrainte bidirectionnelle permet de diminuer, dans le flux binaire, la part des vecteurs mouvement. Cette économie de bits augmente la qualité de reconstruction des sous-bandes mais diminue la qualité de la prédiction. Il y a donc un compromis entre efficacité et qualité. Dans le cas bas débit, la contrainte est extrêmement intéressante alors que dans le cas haut débit son utilité reste relative.

3.2 Adaptation de l'estimateur

L'estimateur de mouvement présenté au chapitre précédant ne prenait en compte que deux images. L'ajout d'une troisième nécessite l'adaptation de l'ensemble du code est des méthodes.

3.2.1 Méthodes de block-matching

Les méthodes ne varient que très légèrement. Dans l'estimation non contrainte, pour chaque bloc, des vecteurs mouvement sont testés selon un certain ordre qui dépend d'une stratégie de recherche et de manière à obtenir le meilleur vecteur pour un critère de correspondance donné. Dans le cas contraint, deux vecteurs opposés (un pour l'estimation entre n et $n-1$ et l'autre pour l'estimation entre n et $n+1$) sont recherchés par bloc. Il suffit alors d'utiliser la même stratégie pour un des deux vecteurs et de prendre son opposé pour le second. Cette méthode est extrêmement simple à mettre en œuvre à partir de la version non contrainte. La seule difficulté (qui reste très raisonnable) réside dans le test d'appartenance des deux vecteurs à la fenêtre de recherche.
Une conséquence de la symétrie des vecteurs est que pour les blocs situés sur les bords gauche et droit, les vecteurs mouvement ont uniquement une composante verticale car sinon un des deux vecteurs sortirait du cadre de l'image. De même, pour les blocs situés sur les bords haut et bas, seule la composante horizontale des vecteurs est non nulle.

3.2.2 Critères de comparaison

Les critères de comparaison ne sont pas épargnés par l'adaptation à la contrainte. Dans le cas non contraint, le critère calcul une certaine fonction utilisant les valeurs des deux blocs. Dans le cas contraint, trois blocs doivent être pris en compte. Nous considérons pour cela que le bloc central a autant d'importance que l'union des deux autres blocs. Nous réalisons une pseudo moyenne sur ces blocs sans en être véritablement une. Un exemple étant souvent plus explicite qu'un long paragraphe, les équations ci-dessous montrent cette adaptation dans un cas simple :

$$SSD(B_c, B_r) = \sum_i \sum_j [B_c(i,j,1) - B_r(i,j,1)]^2 \quad (3.1)$$

$$SSD(B_c, B_{-r}, B_r) = \sum_i \sum_j [B_c(i,j,1) - \frac{1}{2}(B_{-r}(i,j,1) + B_r(i,j,1))]^2 \quad (3.2)$$

La première équation appartient au cas non contraint. La seconde est l'adaptation au cas contraint de la première équation. De plus, B_{-r} et B_r y représentent respectivement les blocs contenus dans les images $n-1$ et $n+1$.

Dans ce cas, la formule n'est que la double somme de l'erreur entre le bloc central et la moyenne des deux autres blocs. En revanche, dans des cas plus complexes comme la ZNSSD, on ne peut se contenter d'effectuer une moyenne. Le terme $B_r(i,j,c) - \overline{B_r(c)}$ du cas non contraint est alors remplacé par le terme $\frac{1}{2}[(B_{-r}(i,j,c) - \overline{B_{-r}(c)}) + (B_r(i,j,c) - \overline{B_r(c)})]$. Ce n'est plus une simple moyenne des deux blocs mais une moyenne des blocs dont la moyenne est ramenée pour chaque bloc à zéro. Tous les critères de comparaison suivent ce schéma d'adaptation de manière à conserver les propriétés du cas non contraint.

La description de l'estimation contrainte étant réalisé, je vous expose maintenant les résultats que nous obtenons avec cette méthode.

3.3 Résultats numériques

L'évaluation de l'estimation de mouvement avec contrainte bidirectionnelle ne peut se faire qu'en la comparant avec l'estimation non contrainte décrite dans le chapitre 2. Nous voulons juger la qualité des vecteurs dans le deux cas. Pour cela, nous réalisons une estimation classique entre les images n et $n-1$ pour obtenir un premier champ de vecteurs. Nous réalisons ensuite une estimation contrainte entres les images $n-1$, n et $n+1$ et obtenons un second champ de vecteurs. Nous comparons alors les reconstruction de l'images n à partir de l'image $n-1$ en utilisant chacun des deux champ de vecteurs. Les paramètres utilisés pour cette évaluation sont les suivants :

- Méthode : block-matching exhaustif (full search)
- Critère : SSD
- Précision : pixel
- Taille de la fenêtre de recherche : $\pm$ 7

La figure 3.2 et le tableaux 3.3 nous permettent de faire plusieurs conclusions sur cette évaluation. Premièrement, la contrainte bidirectionnelle diminue la qualité de l'estimation. Cela semble obligatoire car sinon les vecteurs trouvés avec la contrainte serait plus "optimaux" que sans la contrainte. Cependant la différence est de moins de 2dB pour la séquence « Foreman » et de seulement 0.5dB pour « Flower and Garden ». La différence constatée entre les deux séquences vidéo est liée au mouvement propre contenu dans ces dernières. En effet, la contrainte bidirectionnelle utilise l'hypothèse de linéarité du mouvement. Hors, cette hypothèse est tout a fait vérifiée dans la séquence « Flower and Garden »

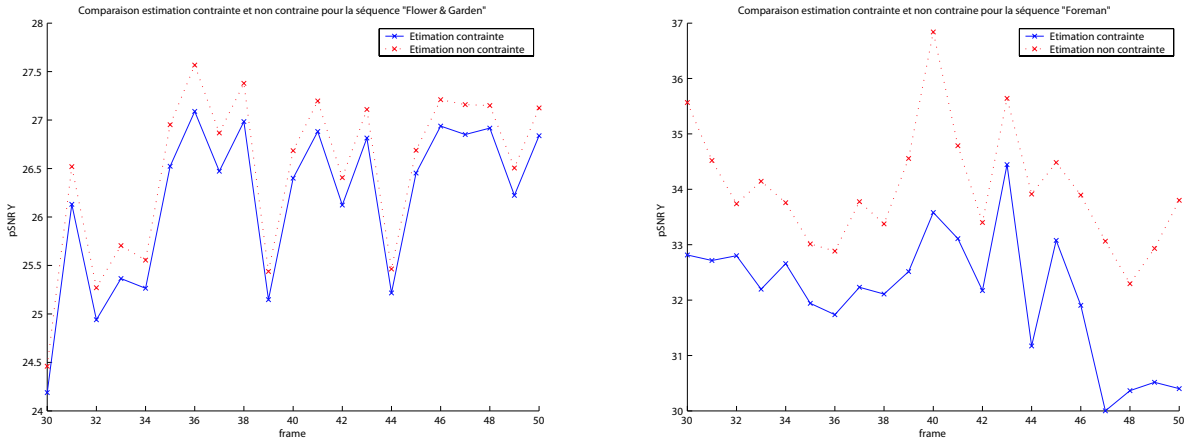


FIG. 3.2 – Comparaison des de l'estimation de mouvement avec contrainte bidirectionnelle par rapport à l'estimation classique

Estimation du mouvement / Évaluation	sans contrainte	avec contrainte
MSE Y	148.52	159.62
MSE YUV	57.42	61.52
PSNR Y (dB)	**26.496**	**26.179**
PSNR YUV (dB)	30.607	30.307
Nombre de blocs visités	**204.28**	**184.56**
Durée block-matching (s)	**1.972**	**4.997**

TAB. 3.1 – Qualité moyenne de l'estimation du mouvement avec ou sans contrainte bidirectionnelle « Flower and Garden »

dans la mesure où le mouvement est issu d'une translation de la caméra. En revanche, le mouvement présent dans la séquence « Foreman » est beaucoup plus chaotique. L'importance d'utiliser des séquences aux caractéristiques différentes est donc clairement établit. Cela permet de voir le comportement d'un algorithme dans un cas favorable et dans un cas défavorable. On peut ainsi conclure que l'utilisation de l'estimation bidirectionnelle a pour conséquence de diminuer le pSNR de 0.5 à 2 dB.
Deuxièmement, les temps de calcul et nombre de blocs visités moyens varient entre les deux estimation. Encore une fois, ces résultats ne sont pas surprenant. En effet, le nombre de blocs visités dans le cas contraint est moins important pour une raison déjà exposée. Dans la mesure où deux vecteurs sont simultanément optimisés, ils arrivent plus souvent qu'un des deux vecteurs sorte de la fenêtre de recherche (sur les bords de l'image) entraînant l'annulation du teste du couple de vecteurs. Enfin, malgré la diminution du nombre de blocs, la durée du block-matching augmente sensiblement avec la contrainte car les critères de comparaison sont alors plus complexes.

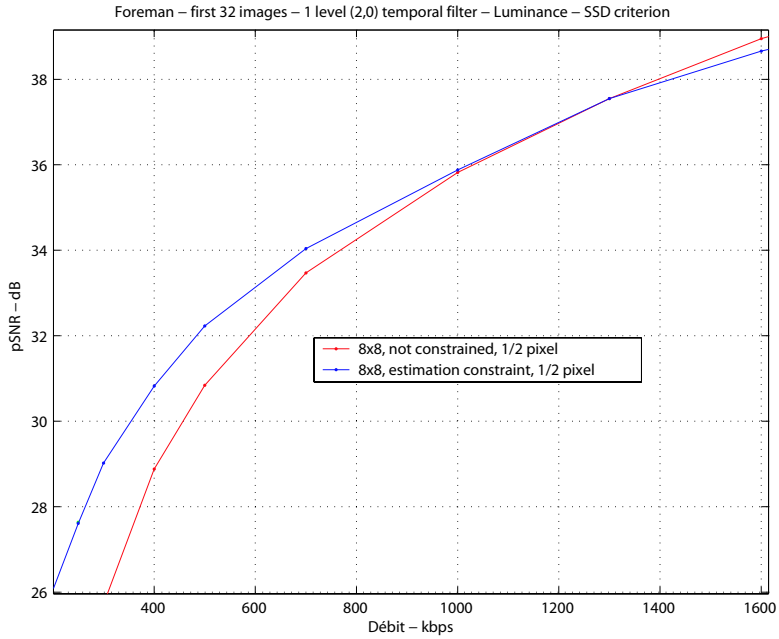


FIG. 3.3 – Influence de la contrainte bidirectionnelle sur la reconstruction globale de la vidéo

Après codage et décodage de la vidéo à plusieurs débits (voir figure 3.3), nous nous apercevons que le codeur avec contrainte bidirectionnelle est plus efficace à bas débit que le codeur sans contrainte (gain de 2dB à 400Kbps) et moins efficace à haut débit. Ce résultat n'est absolument pas surprenant et nous l'avions prédit dès le début. La valeur d du débit pour laquelle les deux courbes s'intersectent (appelé débit critique, environ 1.2 Mbps sur la figure 3.3) est le débit au-delà duquel il n'est plus avantageux d'utiliser les vecteurs contraints. En revanche, n'oublions pas que la contrainte augmente la durée du block-matching et donc même à cette valeur, l'estimation non contrainte est plus intéressante.
Enfin, la valeur d varie entre 500Kbps et 1.3Mbps selon la séquence vidéo utilisée et, de ce fait, aucune valeur fixe ne peut être utilisée comme seuil.

3.4 Conclusion

L'utilisation d'un modèle de mouvement donne, à bas débit, de bons résultats en diminuant la part des vecteurs (ce qui diminue également la qualité de l'estimation) tout en augmentant globalement la qualité de reconstruction de la vidéo.
Un seuil existe néanmoins et est caractérisé par un débit critique d à partir duquel l'effet de la contrainte est néfaste par rapport au cas non contraint.
Cependant, globalement le modèle de mouvement est un outil intéressant permettant d'augmenter la qualité générale de la vidéo pour un faible débit fixé.

Conclusion générale

La compression vidéo est d'ores et déjà un outil largement utilisé et cela grace aux DVD, DivX, caméscopes numériques et même avec les appareils photos numériques capables d'enregistrer des séquences vidéo avec une qualité tout à fait acceptable. Internet a également joué un rôle important pour ce développement en offrant un outil de diffusion massif de vidéos. La compression vidéo est ainsi devenue un besoin pour de nombreuses personnes. L'étude de nouvelles fonctionnalités est un sujet de recherche constant comme par exemple la scalabilité permettant d'obtenir une vidéo à différentes dimensions et à différents niveaux de qualité à partir d'un seul fichier. Les supports visés par une telle application sont par exemple les ordinateurs, les PDA, les téléphones mobiles, Internet, le WAP, etc.

L'équipe CReATIVe développe un codeur vidéo utilisant une transformée en ondelettes 3D. Les deux dimensions spatiales sont gérées par l'utilisation de la norme JPEG2000. En revanche, l'essentiel des travaux est porté sur la transformée temporelle. La construction des sous-bandes nécessite la compensation des images en mouvement et cette dernière est réalisée avec une estimation du mouvement par mise en correspondance de blocs. L'estimation de mouvement a été le sujet de mon stage.
L'estimation du mouvement est un problème classique en traitement d'images et est déjà intégré à de nombreux standards vidéo. Cependant, cette estimation n'utilise en général qu'une faible partie des travaux publiés. En effet, une estimation de mouvement utilise différentes techniques agissant à des niveaux complètement différents. L'objectif de mon stage a été d'inventorier et d'étudier les techniques les plus couramment utilisées, ainsi que d'autres techniques plus récentes et prometteuses, dans le but de les comparer et d'envisager une estimation de mouvement plus efficace et bien adaptée à notre codeur.

Ma contribution consiste en l'étude et à l'implémentation en C++ de méthodes de recherche et de critères de comparaison. Nous avons également montré l'intérêt de l'interpolation d'images qui permet d'obtenir des vecteurs mouvement subpixéliques ainsi que celui de l'utilisation de modèles de mouvement.
Les différentes méthodes de recherche (recherche exhaustive, recherche en trois étapes, recherche utilisant une grille en forme de diamant ou d'hexagone) permettent de gérer le compromis entre rapidité et qualité. Les critères de comparaison (SSD, ZNSSD, VA, et leur version couleur) jouent sur la selection des blocs en calculant une valeur de corrélation entre les blocs de référence et les blocs candidats. L'interpolation d'images permet d'envisager des mouvements subpixéliques et les méthodes d'interpolation (bilinéaires, H26L et courbes de Bézier) permettent d'obtenir les images interpolées plus ou moins rapidement

et avec une qualité d'interpolation sensiblement différente. Enfin, l'utilisation d'un modèle de mouvement, se traduisant dans notre cas par une contrainte bidirectionnelle, permet de diminuer le débit des vecteurs mouvement et d'augmenter, de ce fait, la qualité de reconstruction de la vidéo.

L'implémentation des toutes ces techniques dans une architecture claire m'ont permis de comparer l'influence de chaque paramètre et d'en tirer des conclusions quant à l'efficacité de chacun en terme de rapidité et de qualité. Les conclusions sont les suivantes : pour une application n'ayant aucune contrainte en temps de calcul, je propose d'utiliser une méthode de recherche exhaustive (full search) utilisant le critère SSD couleur, une précision au huitième de pixel obtenue avec une interpolation basée sur les splines et enfin sans contrainte bidirectionnelle. En revanche, pour une application exigeant une exécution rapide, je suggère une recherche utilisant une grille en forme de diamant (diamond search), le critère SSD et l'interpolation au demi ou au quart de pixel obtenue avec la norme H26L. L'utilisation du modèle de mouvement (i.e. de la contrainte bidirectionnelle) dépendra du débit cible recherché par l'utilisateur. À faible débit, l'utilisation du modèle permet d'obtenir un gain significatif (jusqu'à 2dB) sur la reconstruction de la vidéo.
Mon travail a permis de mettre en évidence et de corriger certains problèmes au sein même du code. Les résultats sont obtenus plus rapidement avec un large choix de possibilités. L'implémentation des méthodes de recherche rapides ainsi que la comparaison des méthodes d'interpolation permettent d'avoir des résultats proches de ceux que l'on pourrait obtenir avec les algorithmes qualitativement les plus performants (de 0.5 à 1dB de perte) et ce en un temps beaucoup plus court (150 fois plus rapide). De cette manière, les tests sur le codeur vidéo peuvent être réalisés très fréquemment améliorant ainsi l'avancement des travaux.

Les perspectives de l'estimation de mouvement sont l'implémentation de nouvelles méthodes de recherche, de la recherche de nouveaux critères de qualité plus accès sur la perception d'images impliquant ainsi de nouveaux critères de comparaison. L'aspect interpolation subpixélique peut, quant à elle, être accélérée pour obtenir des résultats qualitativement proche de l'interpolation par spline mais avec une vitesse proche de H26L. Enfin, une des perspectives les plus porteuses est, a mon avis, l'utilisation de modèle de mouvement plus complexes que le mouvement linéaire permettant d'augmenter encore les performances du codeur à bas débit.

J'espère que mon travail aura été apprécié par l'équipe. En ce qui me concerne, j'ai pris beaucoup de plaisir à travailler avec des personnes si agréables, disponibles et aux compétences si importantes. J'ai beaucoup appris de chacun des membres de cette équipe et j'espère leur avoir été utile.

Table des figures

Liste des tableaux

Bibliographie

[1] O. Amadieu, E. Debreuve, M. Barlaud, and G. Aubert, "Inward and outward curve evolution using level set method," in *International Conference on Image Processing*, (Kobe, Japan), 1999.

[2] S. Jehan-Besson, M. Barlaud, and G. Aubert, "Video object segmentation using eulerian region-based active contours," in *International Conference on Computer Vision*, (Vancouver, Canada), 2001.

[3] S. Jehan-Besson, M. Barlaud, and G. Aubert, "Dream^{2}s : Deformable regions driven by an eulerian accurate minimization method for image and video segmentation. application to face detection in color video sequences," in *ECCV*, (Copenhagen, Denmark), 2002.

[4] S. Jehan-Besson, M. Barlaud, and G. Aubert, "DREAM^{2}S : Deformable regions driven by an eulerian accurate minimization method for image and video segmentation," *IJCV*, vol. 53, no. 1, pp. 45–70, 2003.

[5] S. Jehan-Besson, *Modèles de contours actifs basés régions pour la segmentation d'images et de vidéos*. PhD thesis, Université de Nice Sophia Antipolis, France, 2003.

[6] E. Debreuve, M. Barlaud, G. Aubert, and J. Darcourt, "Space time segmentation using level set active contours applied to myocardial gated SPECT," *IEEE Transactions on Medical Imaging*, vol. 20, pp. 643–659, juillet 2001.

[7] F. Precioso and M. Barlaud, "B-spline active contours with handling of topology changes for fast video segmentation," *Eurasip Special issue : Image analysis for multimedia interactive services - PART II*, vol. 2002, pp. 555–560, June 2002.

[8] F. Precioso and M. Barlaud, "Regular B-spline active contours for fast video segmentation," in *International Conference on Image Processing*, (Rochester, NY), September 2002.

[9] F. Precioso, M. Barlaud, T. Blu, and M. Unser, "Smoothing B-spline active contour for fast and robust image and video segmentation," in *International Conference on Image Processing*, (Barcelona, Spain), September 2003.

[10] F. Precioso, M. Barlaud, T. Blu, and M. Unser, "Robust real-time segmentation of images and videos using a smoothing-spline snake-based algorithm," (to appear) 2004.

[11] G. Aubert, M. Barlaud, O. Faugeras, and S. Jehan-Besson, "Image segmentation using active contours : Calculus of variations or shape gradients ?," *SIAM Applied Mathematics*, vol. 1, no. 2, pp. 2128–2145, 2003.

[12] M. Gastaud, M. Barlaud, and G. Aubert, "Combining shape prior and statistical features for active contour segmentation," *to appear in IEEE TCSVT special session on Audio and Video Analysis for Interactive Multimedia Services*, May 2004.

[13] S. Jehan-Besson, M. Gastaud, F. Precioso, M. Barlaud, G. Aubert, and E. Debreuve, "From snakes to region-based active contours defined by region-dependent parameters," *Applied Optics*, vol. 43, pp. 247–256, january 2004.

[14] S. Jehan-Besson, M. Gastaud, M. Barlaud, and G. Aubert, "Region-based active contours using geometrical and statistical features for image segmentation," in *IEEE International Conference on Image Processing*, vol. 2, (Barcelona, Spain), pp. 643–646, September 2003.

[15] M. Gastaud, M. Barlaud, and G. Aubert, "Tracking video objects using active contours and geometric priors," (IEEE 4th European Workshop on Image Analysis for Multimedia International Services), pp. 170–175, April 2003.

[16] M. Gastaud, M. Barlaud, and G. Aubert, "Tracking video objects using active contours," in *IEEE Workshop on Motion and Video Computing*, (Orlando, FL.), pp. 90–95, December 2002.

[17] M. Gastaud and M. Barlaud, "Video segmentation using active contours on a group of pictures," in *IEEE International Conference on Image Processing*, vol. 2, (Rochester, N.Y.), pp. 81–84, September 2002.

[18] M. B. A. Herbulot, S. Jehan-Besson and G. Aubert, "Shape gradient for multimodal image segmentation using joint intensity distributions," in *Proceedings of 5th International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS)*, (Lisbon, Portugal), April 2004.

[19] M. B. A. Herbulot, S. Jehan-Besson and G. Aubert, "Shape gradient for image segmentation using information theory," in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 3, (Montreal, Canada), pp. 21–24, May 2004.

[20] A. Herbulot, S. Jehan-Besson, M. Barlaud, and G. Aubert, "Shape gradient for image segmentation using mutual information," in *International Conference on Image Processing*, (Singapore), October 2004.

[21] A. Gouze, *Schéma Lifting Quinconce pour la Compression d'Images.* PhD thesis, Université de Nice Sophia Antipolis, France, 2002.

[22] N. Božinović, J. Konrad, T. A. M. Antonini, and M. Barlaud, "Motion-compensated lifted wavelet video coding : toward optimal motion/transform configuration," in *Proc. EUSIPCO*, (Vienna, Austria), September 2004.

[23] T. André, M. Cagnazzo, M. Antonini, M. Barlaud, N. Božinović, and J. Konrad, "(N,0) motion-compensated lifting-based wavelet transform," in *Proc. IEEE Intern. Conf. on Acoustics, Speech and Signal Processing*, vol. 3, (Montreal, Canada), pp. 121–124, May 2004.

[24] M. Cagnazzo, T. André, M. Antonini, and M. Barlaud, "A smoothly scalable and fully jpeg2000-compatible video coder," in *Proc. IEEE International Workshop on Multimedia Signal Processing*, (Siena, Italy), September 2004.

[25] M. Cagnazzo, T. André, M. Antonini, and M. Barlaud, "A model-based motion compensated video coder with jpeg2000 compatibility," in *Proc. IEEE International Workshop on Multimedia Signal Processing*, (Singapore), October 2004.

[26] T. André, M. Antonini, and M. Barlaud, "Codage vidéo par transformée en ondelettes 3d au fil de l'eau et compensée en mouvement," in *Proc. Congrès International IEEE de Signaux, Circuits et Systèmes*, (Monastir, Tunisia), pp. 57–60, March 2004.

[27] F. Payan and M. Antonini, "3D mesh wavelet coding using efficient model-based bit allocation," in *Proceedings of IEEE International Symposium on 3D Data Processing Visualization and Transmission (3DPVT)*, pp. 391–394, june 2002.

[28] F. Payan and M. Antonini, "Multiresolution 3D mesh compression," in *Proceedings of IEEE International Conference in Image Processing (ICIP)*, september 2002.

[29] F. Payan and M. Antonini, "Weighted bit allocation for multiresolution 3D mesh geometry compression," in *Proceedings of SPIE Visual Communications and Image Processing (VCIP) Conference*, july 2003.

[30] F. Payan and M. Antonini, "3D multiresolution context-based coding for geometry compression," in *Proceedings of IEEE International Conference in Image Processing (ICIP)*, september 2003.

[31] F. Payan and M. Antonini, "Mean square error for biorthogonal m-channel wavelet coder," *Transactions in Image Processing*, 2003, Submitted.

[32] F. Payan and M. Antonini, "Model-based geometry coding of 3D multiresolution surface meshes," *Transactions in Computer-Aided Graphics Design*, 2004. Submitted to Special session on 3D geometry coding.

[33] F. Payan and M. Antonini, "Model-based bit allocation for normal mesh compression," in *Proceedings of IEEE international workshop on MultiMedia Signal Processing*, (Siena, Italy), september 2004.

[34] G. L. J. Rissanen, "Arithmetic coding," *IEEE Trans. On Comunication*, June 1981.

[35] R. M. N. I. H. Witten and J. G. Cleary, "Arithmetic coding for data compression," *Communication of the ACM*, 1987.

[36] Ohm, "Three dimensional subband coding with motion compensation," *IEEE Trans. On Image Processing*, 1994.

[37] A. H. Y. I. T. Koga, K. Linuma and T. Ishiguro, "Motion compensated interframe coding for video conferencing," *Proc. Nat. Telecommun. Conf.*, 1981.

[38] B. Z. R. Li and M. Liou, "A new three-step search algorithm for block motion estimation," *IEEE Trans. On Circuits ans Systems for Video Technology*, vol. 4, August 1994.

[39] S. Zhu and K. Ma, "A new diamond search algorithm for fast block-matching motion estimation," *IEEE Trans. On Image Processing*, vol. 9, February 2000.

[40] M. B. B.N. Benmoussat and A. Belbachir, "Fast diamond search algorithm for block based motion estimation," *IEEE Trans. On Image Processing*, 2003.

[41] X. L. S. Zhu and L. Chau, "Hexagon-based search pattern for fast block motion estimation," *IEEE Trans. On Circuits ans Systems for Video Technology*, vol. 12, May 2002.

[42] S. O. Y. Liu, "Complexity comparaison of fast block-matching motion estimation algorithms," *IEEE Trans. On Circuits ans Systems for Video Technology*, 2004.

[43] G. Boudaud and A. Garnier, "Segmentation d'objets en mouvement dans les séquences à caméra mobile suivant le principe de compensation," 2001.