La compression vidéo
rapport d'enseignement d'approfondissement

Jérôme LE NY   Benoit LABROUSSE

Le 18 décembre 2000

Table des matières

Introduction

L'amélioration des standards de comréssion vidéo est un domaine de recherche en plein essor, les enjeux technologiques étant par exemple la télévision haute définition ou la transmission en temps réel de vidéo par différents réseaux. En effet, la compression est rendue nécessaire par la quantité d'informations à transmettre : pour un film de 90 minutes, il y a 130000 images d'environ 300 kilo-octets, soit une taille totale de 39 Giga octets ...Il est donc impossible de transmettre un film images par images.

Pour contourner ce problème, on exploite la redondance spatiale et temporelle entre les images d'un film. Dans ce projet, nous avons voulu tirer parti d'un nouvel algorithme mis au point par Christophe Bernard, qui calcule rapidement le flot optique entre deux images consécutives ; nous utilisons aussi le nouveau standard de compression d'images fixes. Dans les deux cas, intervient la théorie des ondelettes.

Après avoir brièvement décrit les standards MPEG-1 et MPEG-2 de compression vidéo et exposé quelques éléments de la théorie des ondelettes, nous présentons l'architecture de notre programme. La dernière partie est consacrée à l'analyse des résultats obtenus.

Chapitre 1

La compression vidéo
État de l'art

L'étude et la conception de la compression vidéo sont récentes comme toutes les techniques liées au multimédia mais l'importance de la vidéo a fait qu'il y a eu enormément de recherches dans ce domaine. Ces travaux ont conduit àà l'établissement de standard, ce qui permet un échange aisé des données, par exemple par l'intermédiaire d'Internet. Nous allons voir quels sont les standards utilisés actuellement.

1.1 La compression d'image fixe : Le standard JPEG

1.1.1 La redondance spatiale

Dans un premier temps, nous allons examiner le standard actuel de compression des images fixes. Il existe en effet un standard, le M3JPEG, qui, pour coder une séquence vidéo, met les unes à la suite des autres les images compressées de la séquence correspondante.

Quand on regarde une image dans son ensemble, on repère immédiatement des zones où l'intensité des pixels est la même. On doit pouvoir utiliser cette redondance pour réduire la quantité d'information à transmettre pour coder une image.

1.1.2 L'utilisation de la transformée en cosinus

Considérons une image comme une fonction de $\mathbb{R}^2$ dans $\mathbb{R}$ et effectuons une transformée de Fourrier à fenêtre. Alors dans les zones où l'image est

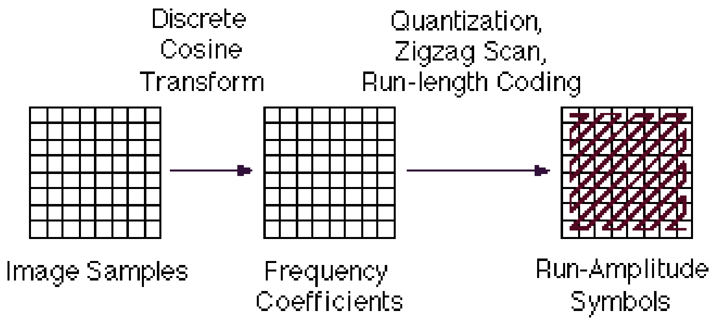

FIG. 1.1 – Le codage des coefficients par un parcours en zig-zag

régulière les hautes fréquences sont presque nulles. On va utiliser ce principe pour coder l'image : on la divise en carrés de 8x8 pixels et sur chacun de ces carrés, on effectue non plus une transformée de Fourrier à fenêtre mais une décomposition sur une base séparable de cosinus. Dans les carrés où l'image est régulière, les coefficients des cosinus de hautes fréquences seront nuls ou très proches de zéro (et donc annulés par une quantification). Il suffit alors de coder efficacement la position de ces coefficients nuls pour obtenir une compression efficace.

1.1.3 Le codage des coefficients

On a vu que la régularité de l'image correspondait à des coefficients hautes fréquence nuls. En parcourant la suite des coefficients en zig-zag (Figure 1.1), ces coefficients se suivent et se trouvent à la fin du parcours. S'ils sont nuls à partir d'un certain rang, il suffit de l'indiquer par un caractère spécial et on économise alors un certain nombre de bits. De même, on repère les élèments nuls que l'on code par un code run-length et les autres élèments par un codage entropique.

1.1.4 Qualité de compression

En adaptant la quantification à la spécificité de l'œil humain, la compression peut atteindre un facteur 10 à 15 sans dégradation visuelle notable. À plus fort taux, des distorsions plus importantes, notamment l'apparition des blocs de 8x8 pixels peuvent se produire.

1.1.5 Application à la vidéo

En affichant successivement toutes les images compressées avec le standard JPEG, on obtient un film de très bonne qualité mais dont la compression n'est pas très importante, typiquement d'un facteur 10, ce qui rend imposible le téléchargement d'un film par une ligne à faible débit. Ce type de compression est cependant très utile pour effectuer des retouches image par image de la séquence vidéo.

1.2 Le standard MPEG-1

Il s'agit certainement du standard le plus répandu. Il ne contente plus de voir une séquence vidéo comme une succession d'images indépendantes mais comme l'évolution d'une scène au cours du temps. L'objectif de ce standard est d'obtenir une qualité correcte, avec un débit de l'ordre de 1,5 Mbits/s.

1.2.1 La redondance temporelle et spatiale

Dans une séquence vidéo, lorsque l'on passe d'une image à la suivante, une grande partie n'est pas modifiée. Seules les zones où il y a un mouvement sont affectées. L'idée est alors de calculer le vecteur mouvement des points de l'image et de prédire l'image suivante grâce à ces vitesses et de ne coder que ce vecteur mouvement. En pratique le vecteur mouvement (ou flot optique) n'est pas calculé pour chaque point mais sur un groupe de 16x16 pixels. On prédit alors approximativement l'image suivante par ces vecteurs mouvements puis on corrige les erreurs introduites grâce à une image correspondant à ces erreurs.

Afin de ne pas propager les erreurs successivement introduites, le standard MPEG-1 divise la séquence à encoder en groupes de 15 images, la première est appelée I pour *Intra*, et elle est codée par l'algorithme JPEG. On prédit ensuite les images 4, 7, 10 et 13, appelées P pour *Prédites* par le calcul du flot entre 1 et 4, puis entre 4 et 7 ...Enfin, entre les images de type P ou I, il y a deux images de type B, qui sont aussi calculées par un algorithme de prédiction de mouvement mais en utilisant les deux images I ou P les plus proches afin de minimiser les erreurs de prédiction (voir Figure 1.2 et Figure 1.3). Les erreurs de prédictions résiduelle, que ce soit pour les images de type P ou les images de type B sont codées avec l'algorithme JPEG.

Le calcul du vecteur vitesse de chacun des blocs de 16x16 pixels n'est pas précisé par le standard. L'algorithme le plus utilisé est certainement celui de block matching : à partir d'un des blocs de l'image qui doit être prédite, on recherche dans l'image précédente le bloc 16x16 qui minimise l'erreur

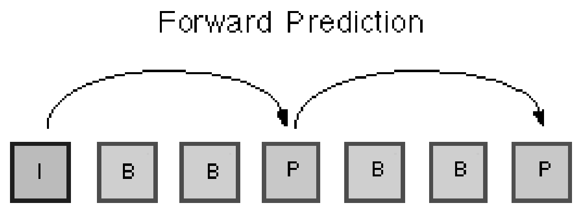

FIG. 1.2 – Les images P sont prédites à partir des images I ou P

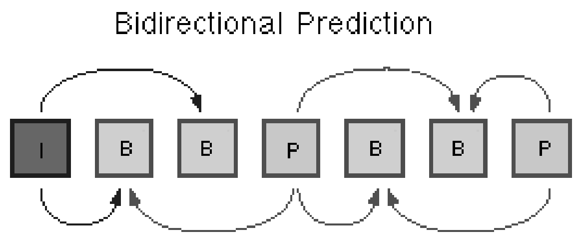

FIG. 1.3 – Les images B sont prédites à partir des images I ou P les plus proches et qui minimisent la distorsion

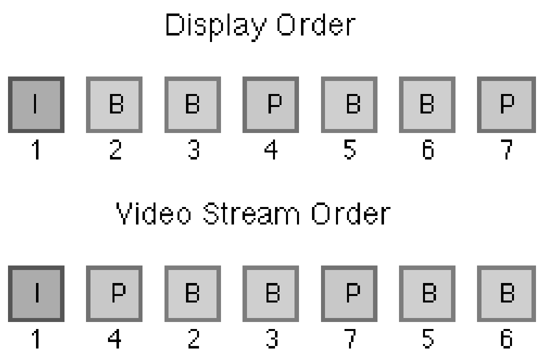

FIG. 1.4 – L'orgaisation du flux vidéo dans un fichier MPEG

quadratique. On obtient alors le vecteur déplacement du bloc correspondant en resolvant le problème :

$$\min_{|(n,m)|<M} |A_{k+1}(i+n, j+m) - A_k(i,j)|^2$$

1.2.2 La composition du flux vidéo

L'ordre d'affichage et de calcul ne sont pas les mêmes, car il faut calculer les images P avant de pouvoir afficher les images de type B. La Figure 1.4 montre la structure du fichier que l'on obtient et l'ordre dans lequel sont calculées les différentes images.

1.3 Les autres standards MPEG-2 et MPEG-4

La norme MPEG-1 n'est pas satisfaisante du fait de sa lenteur, mais aussi de sa faible définition. C'est pourquoi suite à cette norme, le comité d'experts responsable de la mise au point des standards MPEG s'est réuni. Pour une utilisation professionnelle de la vidéo numérique ou pour la télévision numérique par satellite, le standard MPEG-2 a été mis au point. Il s'agit en fait du même principe que le MPEG-1 avec des débit autorisés plus élevé, typiquement de l'ordre de 6 Mbits/s.

Le MPEG-4 est par contre totalement différent dans son approche du codage. Pour obtenir des débits d'information très faibles avec une haute qualité, le codage des images et des mouvements s'interessent au « sens » de l'image et de l'évolution temporelle. On repère des éléments ayant un sens (un personnage, un texte, ...) et on code leur déplacement. Cete approche permet d'obtenir des taux de compression tels qu'un film de qualité équivalente au DVD tient sur un CD.

Chapitre 2

L'utilisation des ondelettes

On appelle ondelette une fonction $\psi \in \mathbf{L}^2(\mathbb{R})$ de moyenne nulle, de norme quadratique égale à un et centrée au voisinage de $t = 0$.

$$\int_{-\infty}^{+\infty} \psi(t)dt = 0 \; et \; \int_{-\infty}^{+\infty} \psi(t)^2 dt = 1 \tag{2.1}$$

On définit alors une famille $\psi_{u,s}$ d'ondelettes à partir de cette ondelette mère en la translatant de u et en la dilatant d'un facteur s :

$$\psi_{u,s}(t) = \frac{1}{\sqrt{s}}\psi(\frac{t-u}{s}) \tag{2.2}$$

Tous les éléments de cette famille sont bien sûr des ondelettes.

2.1 La transformée en ondelettes

Tout comme on définit la transformée de Fourier, on définit la transformée en ondelette de $f \in \mathbf{L}^2(\mathbb{R})$ par

$$Wf(u,s) = \langle f, \psi_{u,s} \rangle = \int_{-\infty}^{+\infty} f(t)\frac{1}{\sqrt{s}}\psi^*\left(\frac{t-u}{s}\right)dt \tag{2.3}$$

La transformée en ondelette peut se réécrire sous la forme d'une convolution, ce qui permet de la considérer comme un filtrage linéaire du signal f

$$\int_{-\infty}^{+\infty} f(t)\frac{1}{\sqrt{s}}\psi^*\left(\frac{t-u}{s}\right)dt = f \star \overline{\psi}_s(u) \tag{2.4}$$

où

$$\overline{\psi}_s(u) = \frac{1}{\sqrt{s}}\psi^*\left(\frac{-t}{s}\right). \tag{2.5}$$

L'ondelette mére est centrée en zéro, de moyenne nulle, et le plus souvent a un support compact. La transformée en ondelette de f, $Wf(u,s)$ mesure la variation de f dans le voisinage de u. Lorsque s tend vers 0, $\psi_{u,s}(t)$ est non nulle que dans un voisinage proche de u donc $Wf(u,s)$ est non nulle si et seulement si f n'est pas régulière au voisinage de u. On voit alors que les coefficients de la transformée en ondelette de f tendront rapidement vers 0 si s tend vers 0 quand f est régulière.

2.2 Quelques résultats importants

2.2.1 Transformée en ondelettes sur $\mathbb{R}$

Tout comme la transformée de Fourier, la transformée en ondelette est inversible, on peut revenir à la fonction initiale à partir de sa transformée comme le prouve le théorème suivant :

Théorème 2.2.1.1. *Soit $\psi \in \mathbf{L}^2(\mathbb{R})$ une fonction réelle telle que*

$$C_\psi = \int_0^{+\infty} \frac{|\widehat{\psi}(\omega)|^2}{\omega} d\omega < +\infty \tag{2.6}$$

Alors $\forall f \in \mathbf{L}^2(\mathbb{R})$ on a

$$f(t) = \frac{1}{C_\psi} \int_0^{+\infty} \int_{-\infty}^{+\infty} Wf(u,s) \frac{1}{\sqrt{s}} \psi\left(\frac{t-u}{s}\right) du \frac{ds}{s^2} \tag{2.7}$$

On peut construire des ondelettes ψ qui génèrent des bases orthonormées de $\mathbf{L}^2(\mathbb{R})$ par translation et dilatation :

$$\{\psi_{j,n}(t) = \frac{1}{\sqrt{2^j}} \psi\left(\frac{t-2^j n}{2^j}\right)\}_{(j,n)\in\mathbb{Z}^2} \tag{2.8}$$

On peut donc imaginer stocker la transformée en ondelettes à la place du signal puis retrouver ce signal.

2.2.2 Transformée en ondelettes et images

Certains résultats sur les ondelettes sont particulirements importants dans le domaine des images. Tout d'abord, on sait construire des ondelettes ψ qui génèrent des bases orthonormées de $L^2(\mathbb{R})$ par translation et dilatation :

$$\left\{\psi_{j,n}(t) = \frac{1}{\sqrt{2^j}} \psi(\frac{t-2^j n}{2^j})\right\}_{(j,n)} \in \mathbb{Z}^2$$

La construction même de ces bases est liée aux approximations à différentes résolutions des signaux. La construcion de bases d'ondelettes en dimension 1 introduit celles de bases orthonormées d'ondelettes en dimension supérieure. On peut donc retenir que pour des signaux de dimension 2 comme les images, on sait construire des bases d'ondelettes : ceci est important notamment pour la compression d'images (cf. section suivante).

D'autre part, les images étant des signaux discrets, on définit pour la transformée en ondelettes un ondelette discrète dilatée par :

$$\psi_j[n] = \frac{1}{\sqrt{a^j}}\psi(\frac{n}{a^j})$$

où l'on prend ψ à support dans $[-K/2, K/2]$ et $a = 2^{1/v}$ tel que $2 \leq a^j \leq K^{-1}$. Cette ondelette discrète a $K\,a^j$ valeurs non nulles sur $[-1/2, 1/2]$. Tout comme avec la tranformée de Fourier, on peut définir pour des signaux finis de taille N une convolution circulaire définissant la transformée en ondelettes discrète. Pour cela on traite les signaux comme des signaux périodiques de période N, et en notant $\overline{\psi}_j[n] = \psi_j^*[-n]$:

$$Wf[n, a^j] = \sum_{m=0}^{N-1} f[m]\psi_j^*[m-n] = f \star \overline{\psi}_j[n] \tag{2.9}$$

Ces convolutions circulaires se calculent avec des FFT.

2.3 La compression par transformée en ondelettes

Le standard JPEG a été amélioré en remplaçant la base dans laquelle on décompose l'image. C'est-à-dire que l'on a remplacé la base de cosinus utilisée par des bases d'ondelettes. Ceci est intéressant aussi pour la compression vidéo, étant donné qu'un codage MPEG code une image sur 15 à l'aide d'un codage JPEG. On a donc commencé par utiliser pour la compression de chaque image marquant le début d'un groupe d'images l'algorithme de compression de Geoff Davis dans le logiciel *LastWave*, qui utilise un codage arithmétique adaptatif pour coder les coefficients obtenus par décomposition sur une base d'ondelettes. Pour des images habituelles, le codage par transformée en ondelettes produit une distorsion minimum avec la meilleure qualité visuelle parmi les codages existants.

Afin d'optimiser le codage, il faut choisir une base d'ondelettes qui produit autant de coefficients nuls que possible. On obtient une base bidimensionnelle à partir d'une base monodimensionnelle générée avec une ondelette mère ψ, choisie en fonction du nombre de moments nuls, de la taille de son support et de sa régularité. En effet, la décomposition produit des coefficients de grandes amplitudes si le support des ondelettes recouvre une discontinuité comme un contour : il faut avoir un support aussi petit que possible. Sur les zones régulières, les coefficients d'ondelettes sont petits à fine échelle si l'ondelette a suffisamment de moments nuls pour tirer parti de la régularité de l'image. Enfin, la régularité de l'ondelette est importante pour réduire la visibilité des erreurs : une erreur de quantification ajoute en effet à l'image une ondelette multipliée par l'amplitude de l'erreur de quantification. Si l'ondelette est irrégulière, il apparaît sur l'image une discontinuité.

2.4 Un calcul du flot optique

Les ondelettes ne sont pas seulement utiles pour améliorer la compression des images *Intra* dans MPEG. Christophe Bernard montre qu'elle peuvent aussi être utilisées pour calculer le flot optique décrivant les vecteurs vitesse des points d'une image. L'algorithme de Bernard a alors une complexité en seulement $O(N)$, où N est le nombre de pixels d'une image de la séquence. Il remplace dans notre programme les algorithmes habituels d'appariements par bloc.

2.4.1 Équation du flot optique

Supposons que $x(t) = (x_1(t), x_2(t))$ est la coordonnée d'un point de niveau de gris de l'image $I(t; x_1, x_2)$. Le niveau de gris $I(t; x(t))$ est constant, donc on a immédiatement :

$$\frac{\partial I(t;x)}{\partial x_1} x_1'(t) + \frac{\partial I(t;x)}{\partial x_2} x_2'(t) + \frac{\partial I(t;x)}{\partial t} = 0 \tag{2.10}$$

c'est-à-dire, en notant $v = (v_1, v_2) = (x_1', x_2')$ et $\nabla I = \left(\frac{\partial I}{\partial x_1}, \frac{\partial I}{\partial x_2}\right)$

$$v.\nabla I = -\frac{\partial I}{\partial t} \tag{2.11}$$

Cette équation du flot nous donne donc la projection du vecteur vitesse $v(x,t)$ sur $\nabla I(x,t)$ mais ne donne aucune information sur la composante orthogonale de $v(x,t)$: c'est le *problème d'ouverture*. Pour réduire le nombre de solutions (pour l'instant infini) de l'équation, la seule méthode est de faire des hypothèses supplémentaires sur la régularité de x et de v. On commence par projeter l'équation sur des ondelettes à plusieurs échelles :

$$\left\langle \frac{\partial I}{\partial x_1} v_1(x), \psi^n_{u,s} \right\rangle + \left\langle \frac{\partial I}{\partial x_2} v_2(x), \psi^n_{u,s} \right\rangle + \left\langle \frac{\partial I}{\partial t}, \psi^n_{u,s} \right\rangle = 0 \quad \forall n = 1...N \tag{2.12}$$

où les ψ^n sont N ondelettes mères différentes, dilatées et translatées :

$$\psi^n_{u,s}(x) = s^{-1}\psi^n(\frac{x_1 - u}{s}, \frac{x_2 - u}{s}).$$

On suppose que s est suffisamment petit pour que $v(x,t) \approx v(u,t)$ sur le support de $\psi^n_{u,s}$. On obtient alors, après une intégration par parties, le systéme suivant, d'inconnues $v_1(u,t)$ et $v_2(u,t)$:

$$\left\langle I, \frac{\partial \psi^n_{u,s}}{\partial x_1} \right\rangle v_1(u,t) + \left\langle I, \frac{\partial \psi^n_{u,s}}{\partial x_1} \right\rangle v_1(u,t) \approx \frac{\partial}{\partial t} \left\langle I, \psi^n_{u,s} \right\rangle \quad \forall n = 1...N \tag{2.13}$$

Aliasing temporel En pratique, une séquence vidéo est composée d'images à des intervalles de temps discrets, et on utilise donc des différences finies : $\frac{\partial I(t+1/2}{\partial t} \approx I(t+1) - I(t)$. On utilise aussi l'estimation $I(t+1/2) = \frac{I(t)+I(t+1)}{2}$. On a :

$$\frac{\partial}{\partial t} \left\langle I, \psi^n_{u,s} \right\rangle = \left\langle I(t+1) - I(t), \psi^n_{u,s} \right\rangle + \eta_a(u,t) \tag{2.14}$$

avec $| \eta_a(u,t) |= O(| v |^2 s^{-2})$

On trouve numériquement (cf. Bernard) que si la contrainte $| v |\leq 0.42 * s$ est vérifiée, les approximations sont valides avec environ 15% d'erreur. On voit donc que le choix de s résulte d'un compromis : s ne doit pas être trop grand pour que $v(x,t)$ soit à peu près constant sur le support de $\psi^n_{u,s}$ et s ne doit pas être trop petit pour que l'inégalité précédente soit vérifiée. On obtient le système discrétisé suivant :

$$\left\langle \frac{I(t+1) + I(t)}{2}, \frac{\partial \psi^n_{u,s}}{\partial x_1} \right\rangle v_1 + \left\langle \frac{I(t+1) + I(t)}{2}, \frac{\partial \psi^n_{u,s}}{\partial x_2} \right\rangle v_2 =$$

$$\left\langle I(t+1) - I(t), \psi^n_{u,s} \right\rangle \quad \forall n = 1..N \tag{2.15}$$

où v_1 et v_2 sont mesurées en u au temps $t + 1/2$. v ne peut pas être calculée si les coefficients d'ondelettes à gauche de 2.15 sont nuls. Ceci se produit lorsque l'image est constante sur le support des ondelettes.

2.4.2 système de flot optique

Le système 2.15 est surdéterminé : il y a plus d'équations que d'inconnues. On l'écrit comme $Mv = Y$, où M est une matrice complexe de taille $N \times 2$, v le vecteur de flot optique de taille 2×1 et Y un vecteur complexe de taille $N \times 1$. Si v est une solution de ce système, alors v est solution du problème aux moindres carrés :

$$\min_{v \in \mathbb{R}} ||Mv - Y||^2 \tag{2.16}$$

que l'on résout en écrivant l'annulation du gradient de la fonctionnelle $J(v) = ||Mv - Y||^2$, soit encore, en utilisant le fait que v est réelle :

$$Re(M^*M)v = Re(M^*Y) \tag{2.17}$$

Trois cas peuvent se présenter :
- la solution aux moindres carrés donne une grande erreur $||Mv - Y||$. Ceci signifie que les approximations qui ont mené au système ne sont pas valable. Soit la vitesse n'est pas approximativement constante sur le support des ondelettes, soit son amplitude est trop grande.
- Le système aux moindres carrés ne peut être résolu, si il y a un prolème d'ouverture sur le support de l'ondelette.
- La solution du problème aux moindres carrés est solution du système initial.

2.4.3 Utilisation des ondelettes. Calcul multiéchelles

En pratique, on part d'un ensemble de N ondelettes mères complexes $\psi^n(x)$ et on définit une famille discrète d'ondelettes par

$$\left\{\psi^n_{j,k} = 2^{-j}\psi^n(2^{-j}x - k)\right\}_{j \in \mathbb{Z}, k \in \mathbb{Z}^2, 1 \leq n \leq N} \tag{2.18}$$

et on calcule le système de flot optique avec ces ondelettes centrées en $2^j k$. Il faut avoir $|v| \ll 2^j$, contrainte qui est contournée par un algorithme multiéchelles qui commence par une estimation du flot optique à une échelle grossière 2^J, et la raffine progressivement avec une compensation de mouvement. Ainsi, si l'on a déjà calculé une estimation $v_{j+1,\frac{k}{2}}$ de v en $u = 2^{j+1}k/2 = 2^j k$ à l'aide d'un système d'ondelettes à l'échelle 2^{j+1}, on peut réduire l'amplitude de déplacement pour permettre le calcul à échelle plus fine : on commence par une translation de $I(t+1)$ de $-2^j\tau$, où $\tau \in \mathbb{Z}^2$ est choisi de manière à minimiser $|2^j\tau - v_{j+1,k/2}|$. On peut calculer à l'étape suivante $v_{j,k} = 2^j\tau + v^r_{j,k}$, où $v^r_{j,k}$ est solution d'un nouveau système en ondelettes.

Chapitre 3

Principe du programme

Avant d'aller plus loin sur les améliorations recherchées pour la compression, nous expliquons le principe de fonctionnement du programme Matlab. Notons que nous ne cherchons pas ici à obtenir un codeur commercial. En particulier le temps pris par la compression et la décompression d'une séquence d'images est rédhibitoire, car pas optimisé. Mais nous nous intéressons ici seulement à avoir un bon rapport compression-qualité.

3.1 Vue d'ensemble

La compression d'une séquence d'images passe par plusieurs étapes : elle est prise en charge par la fonction *compression(NomSequence,NbIm)* qui prend en arguments le nom de la séquence d'images (par exemple *taxi*) et le nombre d'images de la séquence. Cette fonction travaille en plusieurs étapes :

- On charge dans matlab deux images, disons I_1 et I_2 ; la première image d'une séquence est compressée avec le programme *encode* et enregistrée.
- On calcule le flot optique entre ces deux images, à l'aide de *OpticFlow* ; on code ce flot et on l'enregistre.
- On effectue une prédiction utilisant I_1 et le flot optique décodé, à l'aide de la fonction *warping*.
- On calcule l'image d'erreur entre la prédiction et I_2 ; cette image d'erreur est compressée avec *encode* et enregistrée.
- On revient au début du programme : I_1 devient la prédiction à laquelle on ajoute l'image d'erreur décompressée ; on charge dans I_2 l'image suivante.

En conclusion, pour une séquence d'images, nous enregistrons sur le disque la première image compressée, les flots successifs encodés ainsi que les images d'erreur succesives compressées. Nous allons maintenant revenir sur certains points importants de ce déroulement.

3.2 Compression de l'image d'erreur

On a vu que l'on utilise pour corriger la prédiction utilisant le flot optique à chaque étape une image d'erreur, qui est la différence entre I_2 et la prédiction effectuée, image que nous appellerons P. Pour que l'algorithme soit efficace, il faut que cette image d'erreur soit la plus compressée possible. En pratique, elle ne doit corriger que les problèmes classiques rencontrés par le flot optique, par exemple au niveau des contours. En faisant la différence $I_2 - P$, on obtient des valeurs entre -255 et $+255$ (on rappelle que les images sont codées ici en 256 niveaux de gris). La majorité des éléments de cette matrice vaut 0, là où les pixels n'ont pas bougé. Pour compresser cette image d'erreur E, il faut d'abord la ramener à l'intervalle $[0, 255]$; ceci est fait par $(E/2) + 128$. On obtient alors une image quasiment uniformément grise avec des valeurs autour de 128, sauf éventuellement sur les contours et les bords. Cette image étant quasiment uniforme, elle peut-être bien compressée avec le codeur en ondelettes. On discutera dans la partie suivante des taux de compression, mais typiquement on pourra compresser 50 fois l'image d'erreur.

3.3 Prédiction d'une image : le warping

Nous partons d'une image I_1, et du flot optique, qui nous donne les vecteurs vitesse des pixels de I_1. A partir de là, on veut obtenir une prédiction aussi juste que possible (ceci afin d'avoir une bonne image d'erreur) de l'image au temps suivant, à savoir I_2. Il s'agit donc de translater les pixels de I_1 de la bonne façon.

3.3.1 Warping par blocs

Le programme OpticFlow calcule le flot optique sur des carrés de pixels, d'une taille $2^{scale+1}$, où scale est un paramètre que l'on choisit. Ainsi, lorsque scale vaut 2, l'algorithme produit une moyenne des vecteurs vitesse sur des carrés de côté 8 pixels : c'est-à-dire qu'il nous donne les vecteurs vitesse pour des blocs de 8 pixels de l'image. On peut utiliser cela tel quel. Pour prédire l'image suivante, on considère que les blocs se translatent avec le vecteur vitesse donné. Sur l'image d'arrivée, on travaille avec des blocs de même taille, et il faut trouver la correspondance entre ces blocs et ceux de l'image de départ.

Warping en sens direct On part d'un bloc de l'image I_1, pour lequelle on a un vecteur déplacement, et on le translate. Mais évidemment, la valeur du déplacement n'est pas entière. Pour comprendre le problème, on raisonne en fait en niveaux de gris. Lorsqu'un point (x,y) de l'image de départ, possédant une certaine valeur γ donnant sa couleur, se déplace d'un valeur on entière (disons 1.7 pixels suivant x par exemple), il faut changer la valeur de plusieurs pixels, avec des coefficients barycentriques appropriés (dans notre exemple, on donne au pixel $x + 1$ la valeur $0.3 * \gamma$ et au pixel $x + 2$ la valeur $0.7 * \gamma$.) Dans la pratique, il faut faire un barycentrage sur 4 points au sommet d'un petit carré de côté 1 pixel.

Une difficulté apparaît immédiatement : des points différents de l'image de départ peuvent atteindre par cette méthode le même pixel. Pour avoir un résultat correct, il faudrait faire en plus une moyenne des intensités reçues des différents pixels, ce qui obligerait à garder quelque part les différents points de l'image de départ qui atteignent un même point de l'image d'arrivée. D'autre part, certains points de l'image d'arrivée ne seront peut-être pas atteints par cette méthode : ceci peut être corrigé par l'image d'erreur, mais elle devient alors plus difficile à compresser. En conclusion cette méthode dans le sens direct soulève donc des difficultés que l'on peut éviter par une transformation inverse.

Warping en sens inverse C'est cette méthode que nous avons utilisée. Ici on part d'un point l'image d'arrivée, et on cherche les points de l'image de départ qui arrivent sur ce point. Concrètement, on part d'un point (x, y) de l'image d'arrivée, pour lequel *OpticFlow* a donné un vecteur vitesse $v(x, y)$. On donne alors au point (x, y) la couleur du point $(x, y) - v(x, y)$ correspondant au point de l'image de départ. Remarquons au passage que le flot optique nous donne $v(x, y, t + 1/2)$, et on fait donc l'approximation que le pixel a un mouvement de translation uniforme entre les deux images, avec ce vecteur vitesse. Il faut de plus tenir compte des vecteurs vitesses qui nous donnent des déplacements non entiers, ceci par un barycentrage : on donne en fait au point (x,y) une couleur qui est la moyenne pondérée des couleurs des points de l'image de départ entre lesquels nous fait arriver le déplacement $-v(x, y)$.

3.3.2 Suppression des effets de blocs

Le warping par bloc considère que tous les points d'un bloc de taille $2^{scale+1}$ de l'image on la même vitesse, qui est la moyenne des vitesses des points du bloc. Ceci produit des discontinuités sur l'image prédite, car les mouvements des points sont indépendants des blocs. Encore une fois, ceci pourrait être corrigé par l'image d'erreur, mais en même temps elle est alors plus difficile à compresser. Pour atténuer ces effets de bloc, on procède avant le warping à une interpolation des vitesses : on attribue à un point de l'image la moyenne des vitesses des 4 blocs autour de ce point, avec une pondération en fonction de la distance à ces blocs. Pour cela, on utilise un nouveau système de coordonnées locales pour ce point, en considérant l'origine locale au milieu d'un des anciens blocs et un nouveau pavage de l'image par des blocs centrés en ces origines et toujours de côté $2^{scale+1}$ (cf. figure 3.1).

3.4 Codage du flot optique

Une des questions les plus importantes pour optimiser la taille du fichier final est la manière dont on va coder le flot optique. Le programme *OpticFlow* nous renvoie deux matrices contenant les valeurs (selon l'axe des x et des y) des vecteurs vitesses sur chaque bloc de taille $2^{scale+1}$. Ce sont ces deux matrices qu'il faut coder.

3.4.1 Quantification et codage entropique

La première approche est de vouloir simplement quantifier les valeurs du flot optique, puis d'effectuer un codage entropique (cf. chapitre suivant). On effectue pour une matrice une quantification uniforme, sur 256 valeurs à partir de la valeur minimale de la matrice. On écrit dans un fichier la matrice contenant le numéro de la boite de quantification de l'élément de la matrice de départ, avec le numéro 1 pour la boite commençant à partir de la valeur minimale. On a ainsi un fichier contenant des entiers non signés codés sur 8 bits.

Une fois cette quantification effectée, on procède à un codage entropique, en comprimant notre fichier d'entiers avec *gzip*. On aurait aussi pu utiliser un code de Huffman ou un code arithmétique, dans des versions adaptatives. L'essentiel est d'avoir un code sur un nombre de bits qui soit aussi proche que possible de l'entropie (cf. chapitre suivant), et il existait déjà des codeurs très performants comme *gzip*.

3.4.2 Le flot optique comme image

On observant la répartition des vecteurs mouvements sur la carte du flot optique, on remarque qu'il existe de larges zones où ces vecteurs sont nuls, et quelques zones où ces vecteurs ont une valeur non nulle. En fait, dans la structure, cela ressemble à une image, avec des zones uniformes et

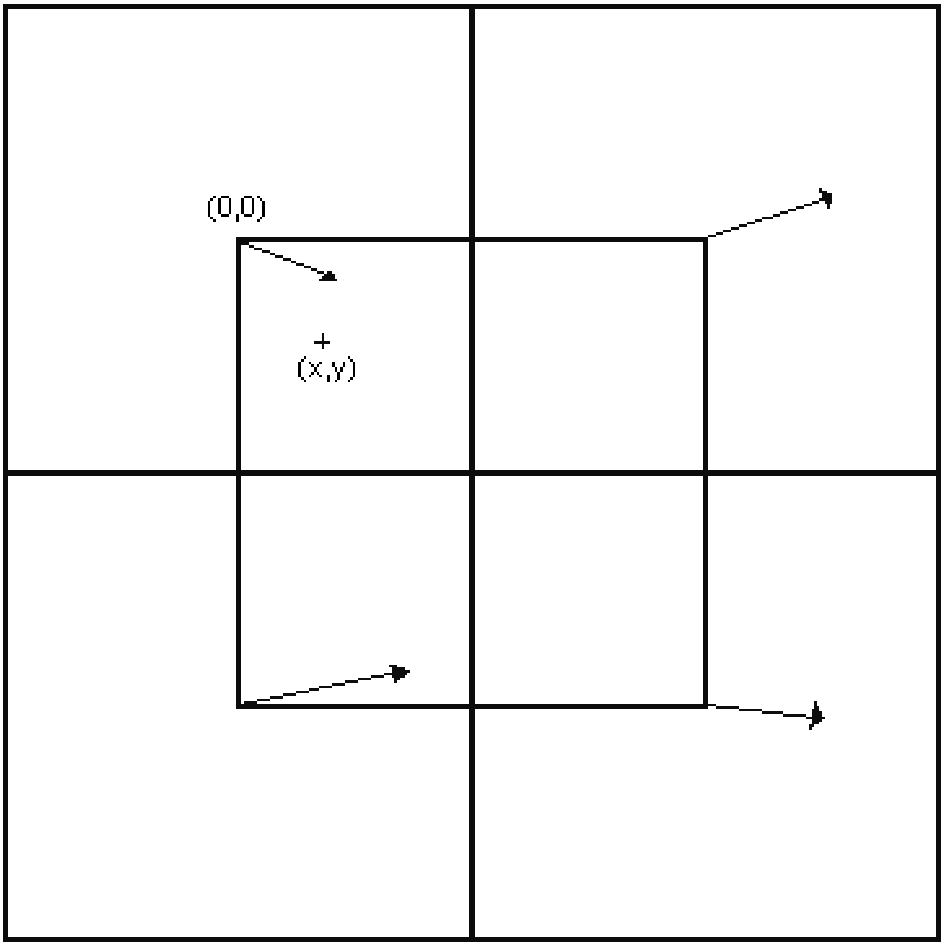

FIG. 3.1 – Vitesse en un point de l'image, moyenne pondérée de 4 vitesses

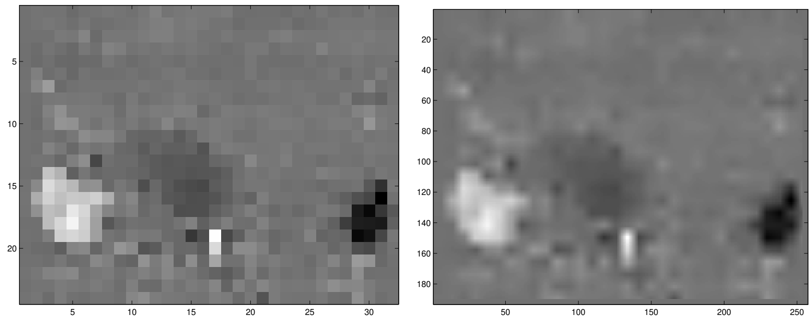

FIG. 3.2 – Résultat de l'interpolation du flot, dont les valeurs sont vues ici sous forme d'image.

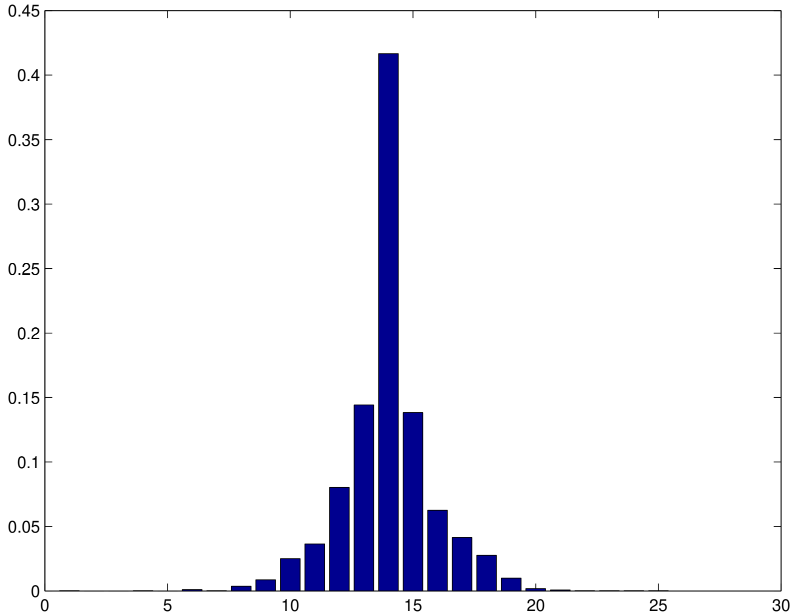

FIG. 3.3 – Répartition des valeurs du flot.

quelques discontinuités. En particulier, on a envie de tirer partie des larges zones uniformes de la carte, et pour cela l'utilisation des ondelettes semble appropriée. C'est ce qui nous motive pour utiliser le programme *LastWave* pour coder le flot en le décomposant dans une base d'ondelettes. La seule précaution à prendre avant d'utiliser la fonction *encode* est de transformer les matrices de flot en images. Là encore, on se limite à 255 valeurs, ce qui est largement suffisant. On effectue une translation et une dilatation des valeurs des matrices de flot pour se ramener entre 0 et 255 puis on tronque pour avoir des entiers ; on stocke aussi les valeurs minimum et maximum pour pouvoir faire la transformation inverse lors du décodage. L'inconvénient majeur de cette méthode est que le programme LastWave de Geoff Davis ne fonctionne pas sur des images trop réduites, ce qui nous oblige à utiliser l'échelle 1 pour le calcul du flot. On commence donc avec une quantité d'information plus élevée que dans le cas du codage entropique précédé d'une quantification où l'échelle 2 peut être utilisée avec de bons résultats. Par exemple, si l'on peut compresser 2 fois le flot quantifié calculé à l'échelle 2, il faudra alors comprimer 8 fois le flot en utilisant une transformée en ondelettes.

Chapitre 4

Allocation optimale des bits

Dans cette partie, on va tenter de minimiser la taille du fichier final (donc maximiser la compression) à distorsion fixée ou de minimiser la distorsion à compression fixée.

4.1 Calcul et estimation de la distorsion

La compression et la quantification des images d'erreurs et du flot introduisent des imperfections lors de la reconstruction de la séquence. On mesure la distorsion créée en calculant $E(|\widetilde{I} - I|^2)$ où $\widetilde{I}$ est l'image reconstruite dans la séquence. D'autres quantités comme la racine de la distorsion ou le rapport signal-bruit en crête permettent d'estimer la dégradation de la qualité de l'image.

$$MSE = E(|\widetilde{I} - I|^2) \tag{4.1}$$

$$RMSE = \sqrt{E(|\widetilde{I} - I|^2)} \tag{4.2}$$

$$PSNR = 10 \log_{10} \frac{N^2 255^2}{d(\overline{R}, f)} \tag{4.3}$$

4.1.1 Codage par transformée et distorsion

On a vu que pour coder efficacement un image, on la décompose dans une base séparable d'ondelettes de $\mathbb{R}^2$ puis on la quantifie afin d'obtenir un maximum de coefficients nuls. Décomposons une image I de N pixels dans une base orthonormale $B = \{g_m\}_{0 \leq m < N}$:

$$I = \sum_{m=0}^{N} I_B[m] g_m \textit{ avec } I_B[m] = \langle I, g_m \rangle \tag{4.4}$$

Les $I_B[m]$ sont les coefficients de I dans la base B. Le codage consiste à quantifier ces coefficients et à reconstruire I de manière approchée :

$$\widetilde{I} = \sum_{m=0}^{N} Q(I_B[m]) g_m \tag{4.5}$$

où Q est le quantificateur. Cette quantification est nécessaire car afin de pouvoir générer un code fini, chaque coefficient $I_B[m]$ ne peut prendre qu'un nombre fini de valeurs différentes. Le quantificateur scalaire Q approche chaque élément x de l'intervalle $]y_{k-1}, y_k]$ par a_k. La précision de cette quantification, et donc la valeur de la distorsion, dépend du nombre de valeurs différentes que l'on peut coder et donc du nombre de bits alloués pour coder une valeur. La distorsion peut s'écrire en fonction du nombre total R de bits utilisés :

$$d(R, I) = \|I - \widetilde{I}\|^2 = \sum_{m=0}^{N} |I_B[m] - Q(I_B[m])|^2 \tag{4.6}$$

4.1.2 Codage des coefficients

Les coefficients quantifés prennent un nombre fini de valeurs donc ils peuvent être codés par un nombre fini de bits. On peut attribuer à chacune des valeurs possibles le même poids, c'est à dire coder tous les coefficients sur le même nombre de bits ou essayer d'optimiser la repartition du nombre de bits par coefficients.

Dans le premier cas, si les coefficients prennent leurs valeurs dans un ensemble A à K éléments, chacun sera codé sur $\lceil \log_2 K \rceil$ bits. Pour l'image entière on obtient donc $N\lceil \log_2 K \rceil$. Le but du codage est de réduire ce nombre en tenant compte de la fréquence d'apparition de chacun des éléments de l'ensemble A : on utilise des mots plus courts pour coder le éléments les plus fréquents.

Modélisons un coefficient par une variable aléatoire X qui prend ses valeurs dans l'ensemble A. Notons p_k la probabilité d'apparition de la valeur x_k. Si on note l_k la longueur du mot codant la valeur x_k, on obtient le nombre de bits moyen R_X nécessaire au codage d'un symbole :

$$R_X = \sum_{k=1}^{K} P(X = x_k) l_k = \sum_{k=1}^{K} p_k l_k \tag{4.7}$$

Le but est donc de minimiser R_X

Code préfixe

Les mots n'auront pas tous la même longueur donc on doit pouvoir les séparer les uns des autres sur un autre critère que la longueur fixe. Une suite d bits doit être décodable de manière unique, et pour cela il existe une famille de code spécifique, les codes préfixes. On appelle code préfixe tout code dans lequel un mot n'est le début d'aucun autre. Par exemple, les mots :

$$\{x_1 = 1001, x_2 = 10011, x_3 = 11001\} \tag{4.8}$$

ne vérifient pas la condition préfixe et le message 100111001 peut être décodé en $x_2 x_1$ ou en $x_1 x_3$.

On peut montrer que cette condition préfixe est équivalente à la possibilité de représenter le code sous la forme d'un arbre à K feuilles et où le passage d'un noeud à son fils droit se code par l'ajout d'un 1 à la fin du mot et le passage au fils gauche par l'ajout d'un 0. La longueur du mot codé est donc donné par la profondeur de la feuille dans l'arbre.

Entropie de Shannon

Shannon définit l'entropie d'une source aléatoire X qui prend les valeurs x_k avec la probabilité p_k par

$$\mathcal{H}(X) = -\sum_{k=1}^{K} p_k \log_2 p_k \tag{4.9}$$

Alors le théorème de Shannon donne une borne inférieure du nombre de bits moyen d'un code préfixe en fonction de l'entropie de la source.

Théorème 4.1.2.1 (Shannon). *Soit X une source aléatoire dont les symboles $\{x_k\}_{1 \leq k \leq K}$ ont des probabilités d'apparition $\{p_k\}_{1 \leq k \leq K}$. Le nombre de bits moyen d'un code préfixe vérifie*

$$R_X \geq \mathcal{H}(X) \tag{4.10}$$

De plus, il existe un code préfixe tel que

$$R_X \leq \mathcal{H}(X) + 1. \tag{4.11}$$

Ce théorème montre donc qu'il suffit de calculer l'entropie d'un signal pour savoir quel taux de compression on peut atteindre. Des algorithmes tels que le codage de Huffman par blocs, le codage arithmétique ou la commande *gzip* sous Unix permettent d'atteindre des nombres de bits moyens qui tendent vers l'entropie de Shannon. Ils sont donc optimaux et sans pertes dans le sens où on retrouve exactement le signal avant compression.

4.1.3 Quantification des coefficients

Maintenant que l'on a vu que l'on pouvait minimiser le nombre de bits utilisés pour coder une suite de valeur, il faut déterminer quelles vont être ces valeurs. Pour une source aléatoire X, un quantificateur scalaire approche X par $\widetilde{X} = Q(X)$ qui prend un nombre fini de valeurs différentes. En général, un quantificateur scalaire approche tout $x \in]y_{k-1}, y-k]$ par a_k. On définit donc des intervalles de quantification sur lesquels Q est constant.

On peut distinguer deux cas particuliers dans lesquel on peut avoir une expression explicite de la distorsion en fonction des intervalles de quantifications.

Quantification haute résolution

En notant $\widetilde{X} = Q(X)$, si $p(x)$ est la densité de probabilité de X, alors la distorsion donnée par l'erreur quadratique a pour expression :

$$D = E(|X - \widetilde{X}|^2) = \int_{-\infty}^{+\infty} |x - Q(x)|^2 p(x) dx \tag{4.12}$$

Le quantificateur a une haute résolution si $p(x)$ peut être considéré comme constant sur chaque intervalle de quantification, et nul sur $]-\infty, y_0] \cup [y_k, +\infty[$. On note p_k la moyenne de $p(x)$ sur chaque intervalle de quantification de longueur Δ_k, et a_k est donc la valeur de Q sur Δ_k.

Théorème 4.1.3.1. *Alors sous cette hypothèse, l'erreur D minimum obtenue en optimisant la position des niveaux$\{a_k\}_{0 \le k \le K}$ est*

$$D = \frac{1}{12} \sum_{k=2}^{K-1} p_k \Delta_k^2 \tag{4.13}$$

Justifions rapidement cette assertion : p étant constant par morceaux et à support compact, on peut écrire :

$$D = \sum_{k=2}^{K-1} \frac{p_k}{\Delta_k} \int_{y_{k-1}}^{y_k} (x - a_k)^2 dx \tag{4.14}$$

Or cette expression est minimale en prenant $a_k = \frac{y_{k-1}+y_k}{2}$, en remplaçant on obtient la valeur correspondante.

Quantification optimale dans le cas haute résolution

Si l'on fixe l'erreur totale D, il faut optimiser le choix des intervalles de quantification afin de minimiser le nombre de bits nécesaire au codage de l'information. Le théorème de Shannon montre que l'on peut approcher ce nombre de bits par l'entropie. Il faut donc minimiser $\widetilde{\mathcal{H}}$ à D fixé.

Pour une source aléatoire X, définissons son entropie différentielle $\mathcal{H}_d(X)$:

$$\mathcal{H}_d(X) = -\int_{-\infty}^{+\infty} p(x)\log_2(x)dx \tag{4.15}$$

Alors le théorème de Gish et Pierce relie $\mathcal{H}_d(X)$ et $\widetilde{\mathcal{H}}$.

Théorème 4.1.3.2 (Gish et Pierce). *Si Q est un quantificateur à haute résolution par rapport à $p(x)$, alors*

$$\widetilde{\mathcal{H}}(X) \geq \mathcal{H}_d(X) - \frac{1}{2}\log_2(12D) \tag{4.16}$$

Cette inégalité est une égalité si et seulement si Q est un quantificateur uniforme.

Dans le cas d'égalité, on a donc

$$R_X = \widetilde{\mathcal{H}}(X) = \mathcal{H}_d(X) - \frac{1}{2}\log_2(12D) \tag{4.17}$$

et comme le quantificateur est uniforme, les Δ_k sont tous égaux à Δ donc $D = \frac{1}{12}\sum_{k=2}^{K-1} p_k\Delta^2 = \frac{\Delta^2}{12}$. On obtient alors

$$R_X = \mathcal{H}_d(X) - \log_2\Delta \tag{4.18}$$

ce qui donne une expression explicite du facteur de compression que l'on peut espérer en fonction de la distorsion.

Quantification optimale dans le cas basse résolution

Si la base B dans laquelle on décompose notre signal est bien choisie, beaucoup de coefficient vont être nuls et on trouvera un grand pic centré en 0. La pente de la densité de probabilité de la variable aléatoire est donc loin d'être négligeable et on ne peut plus considérer que cette densité est constante par morceaux. L'idée est en fait de ne coder que les coefficients tels que $Q(I_B[m]) \neq 0$ dont on code la position dans une *carte binaire* b telle que

$$b[m] = 0 \text{ } si \text{ } Q(I_B[m]) = 0 \text{ } et \text{ } b[m] = 1 \text{ } si \text{ } Q(I_B[m]) \neq 0 \tag{4.19}$$

Si S est le nombre de coefficients qui ne sont pas quantifiés à 0, les fréquences de 0 et de 1 dans la carte sont respectivement $f_0 = \frac{N-S}{N}$ et $f_1 = \frac{S}{N}$. Le nombre de bits nécessaire R_0 est donné par l'entropie de ce signal

$$R_0 = -N(p_0 \log_2 p_0 + p_1 \log_2 p_1) \tag{4.20}$$

Si on appelle R_1 le nombre de bits qu'il faut pour coder les coefficients non nuls, le nombre total de bits R est donné par

$$R = R_0 + R_1. \tag{4.21}$$

Notons $[-\frac{\Delta}{2}, \frac{\Delta}{2}]$ l'intervalle de quantification à 0. On peut alors séparer la distorsion en 2 parties, selon que $I_B[m$ soit dans $[-\frac{\Delta}{2}, \frac{\Delta}{2}]$ ou non :

$$D(R, I) = \sum_{|I_B[m]| < \Delta/2} |I_B[m]|^2 + \sum_{|I_B[m]| \geq \Delta/2} |I_B[m] - Q(I_B[m])|^2 \tag{4.22}$$

Le premier terme de cette somme représente l'erreur commise lorsque l'on approxime I par les S coefficients significatifs. Or il a été montré par DeVore, Jawerth et Lucier que cette erreur était la plus importante dans la distorsion totale.

On classe maintenant les coefficients significatifs par ordre décroissant afin d'obtenir $|I'_B[k]| \geq |I'_B[k+1]|$. Le théorème de Mallat et Falzon nous donne alors un ordre de grandeur de la distorsion en fonction de R.

Théorème 4.1.3.3 (Mallat, Falzon). *Soit Q un quantificateur uniforme. Il existe un codage adaptif tel que pour tout $s > 1/2$ et $C > 0$, si $|I'_B[k]| \sim Ck^{-s}$ alors pour $R \leq N$*

$$D(R, I) \sim C^2 R^{1-2s} \left(1 + \log_2 \frac{N}{R}\right)^{2s-1} \tag{4.23}$$

Dans les deux cas c'est donc un quantificateur uniforme qui est utilisé : dans le cadre de la quantification à haute reésolution, l'optimisation conduit au quantificateur uniforme alors dans le cadre de la quantification base résolution, c'est une hypothèse.

4.2 Répartition des bits

Dans une séquence de N images, la distorsion totale est égale à la somme de toutes les distorsions. Par exemple, lors de la compression d'une séquence de quinze images, la distorsion totale est égale à la somme de la distorsion due à la compression de la première image, et à la somme de la distorsion de chacune des images suivantes, introduite par la compression de l'image d'erreur et par la quantification du flot. On peut donc mettre la distorsion totale sous la forme :

$$D = D_0 + \sum_t D_{t,E} + \sum_t D_{t,F} \tag{4.24}$$

4.2.1 Étude de la distorsion à image fixe

Le programme externe *LastWave*, écrit par Geoff Davis, permet de compresser des images fixes par une décomposition sur une base d'ondelette suivie d'une quantification et d'un codage entropique. Le seul paramètre variable est le taux de compression, donc le nombre de bits alloués par pixel. Pour étudier l'évolution de la distorsion introduite par cette compression, on a réalisé des mesures de distorsions sur les images d'erreur de la séquence *taxi*, pour des taux de compression τ variant de 10 à 160. On obtient les résultas du tableau 4.1, où R est le nombre de bits par pixels, et *MSE, RMSE* et *PSNR* sont les mesure définies précédemment.

On a alors tracé la courbe correspondante, voir figure 4.2.1

À la vue de la courbe, on peut essayer de trouver une expresion du logarithme de la distorsion en fonction de l'inverse du taux de compression, de la forme

$$\log MSE = a\frac{1}{\tau} + b \tag{4.25}$$

Par regression linéaire, on trouve $\log MSE = -14.8\frac{1}{\tau} + 1.93$ et le resultat est assez proche de la courbe comme le montre la figure 4.2.1.

4.2.2 Étude de la distorsion due à la compression du flot

Résultats expérimentaux

Comme il a été dit dans le chapitre de description du programme, deux méthode ont été employées pour compresser le flot. La première consiste à quantifier le flot puis à effectuer un codage entropique, la deuxième à considérer le flot comme une image et à le coder par une transformée en ondelettes.

τ	R	MSE	$RMSE$	$PSNR$
10	0.8	1.61	1.27	46.1
20	0.4	3.32	1.82	43.0
30	0.27	4.01	2.00	42.1
40	0.2	4.58	2.14	41.5
50	0.16	4.97	2.23	41.2
60	0.13	5.25	2.29	40.9
70	0.11	5.57	2.36	40.7
80	0.10	5.74	2.39	40.5
90	0.089	5.85	2.42	40.5
100	0.080	5.96	2.44	40.4
110	0.073	6.05	2.46	40.3
120	0.067	6.16	2.48	40.2
130	0.062	6.22	2.49	40.2
140	0.057	6.30	2.51	40.1
150	0.053	6.36	2.52	40.1
160	0.050	6.43	2.54	40.0

TAB. 4.1 – Évolution de l'erreur en fonction du taux de compression

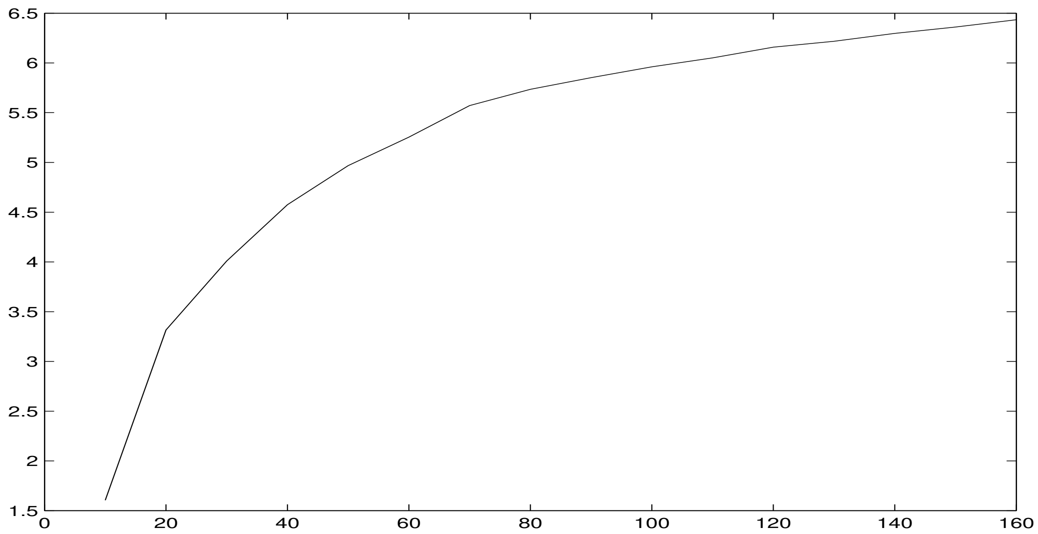

FIG. 4.1 – Évolution de la distorsion (MSE) en fonction du taux de compression

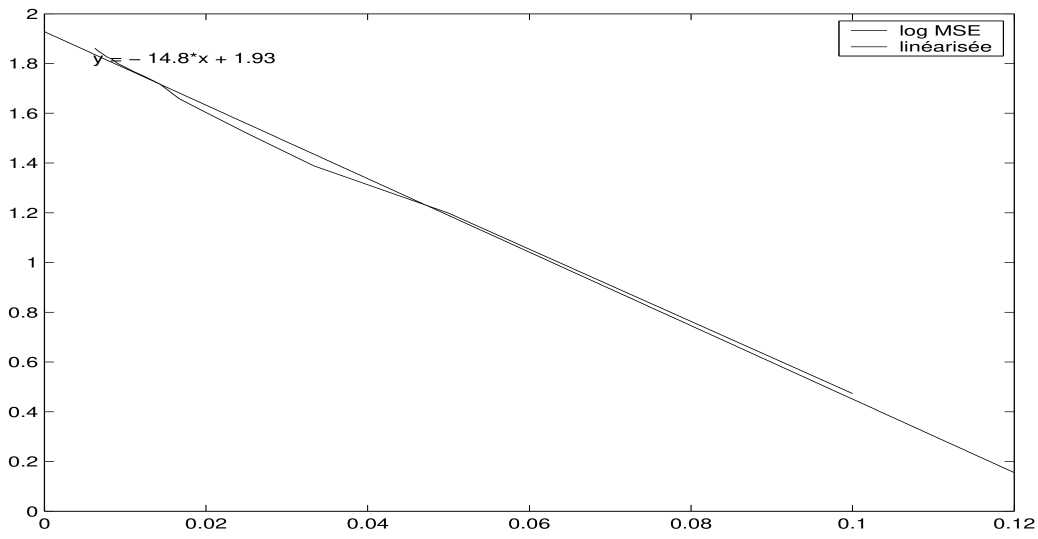

FIG. 4.2 – Linéarisaton de la MSE en fonction du taux de compression

Pour mesurer les distorsions introduites par ces deux codages, nous avons calculé la distorsion entre l'image obtenue par un warping avec le flot obtenue par *OpticFlow* et l'image obtenue par un warping avec le flot qui a subi un codage/decodage. Lorsque l'on utilise le codage par transforée en ondelettes, on fait donc varier le taux de compression et lorsque l'on fait un codage entropique, on fait varier le pas de quantification. Les résultats obtenus sont résumés dans les tableaux et graphiques suivants.

La courbe 4.3 est totalement irregulière, nous n'avons aucun espoir de trouver une expression de la distorsion en fonction du pas de quantification ou en fonction de l'entropie. Par contre, la courbe 4.5 est plus régulière, si l'on considère que le premier point n'apporte pas d'information (en effet c'est une compression d'un facteur 1, donc d'aucune utilité). En traçant la droite qui approche le mieux la courbe par régression linéaire, on a une aproximation de la valeur de la dérivée.

Par régression linéaire sur la courbe 4.6, on trouve alors une expression de la distorsion en fonction du taux de compression :

$$MSE = a\tau + b \; avec \; a = 0.5 \; et \; b = 0.17 \tag{4.26}$$

Δ	H	MSE	$RMSE$	$PSNR$
0.005	11.9	6.6	2.6	40
0.01	10.1	16	4	36
0.015	9.0	17	4.1	36
0.02	8.3	24	4.9	34
0.025	7.7	28	5.3	34
0.03	7.2	17	4.1	36
0.035	6.8	18	4.2	36
0.04	6.4	19	4.4	35
0.045	6.2	38	6.2	32
0.05	5.9	37	6.1	32
0.055	5.6	20	4.5	35
0.06	5.4	41	6.4	32
0.065	5.3	46	6.8	32
0.07	5.0	40	6.3	32
0.075	4.8	42	6.5	32
0.08	4.8	22	4.7	35
0.085	4.7	42	6.4	32
0.09	4.4	21	4.6	35
0.095	4.5	44	6.6	32
0.1	4.3	26	5.1	34

TAB. 4.2 – Évolution de la distorsion en fonction du pas de quantification Δ du flot.

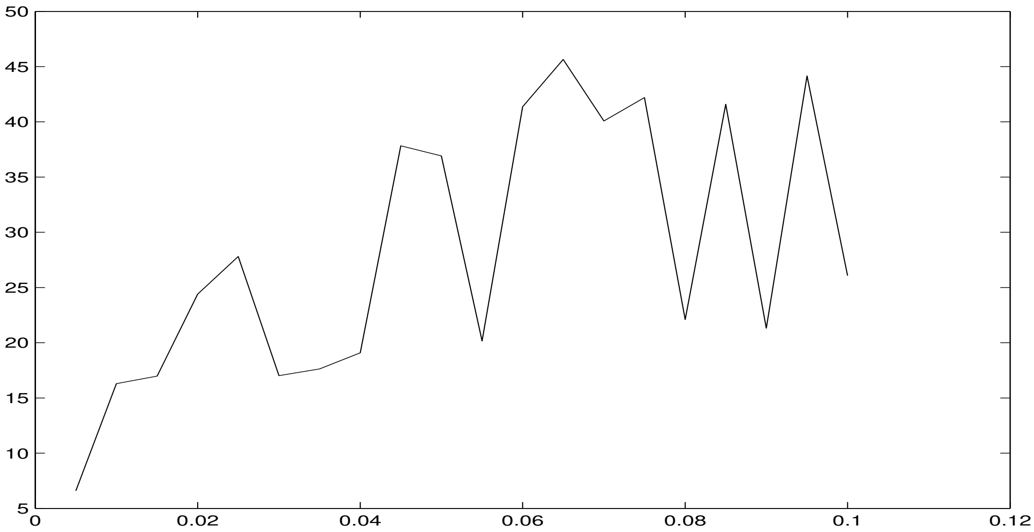

FIG. 4.3 – Évolution de la distorsion (MSE) en fonction du pas de quantification.

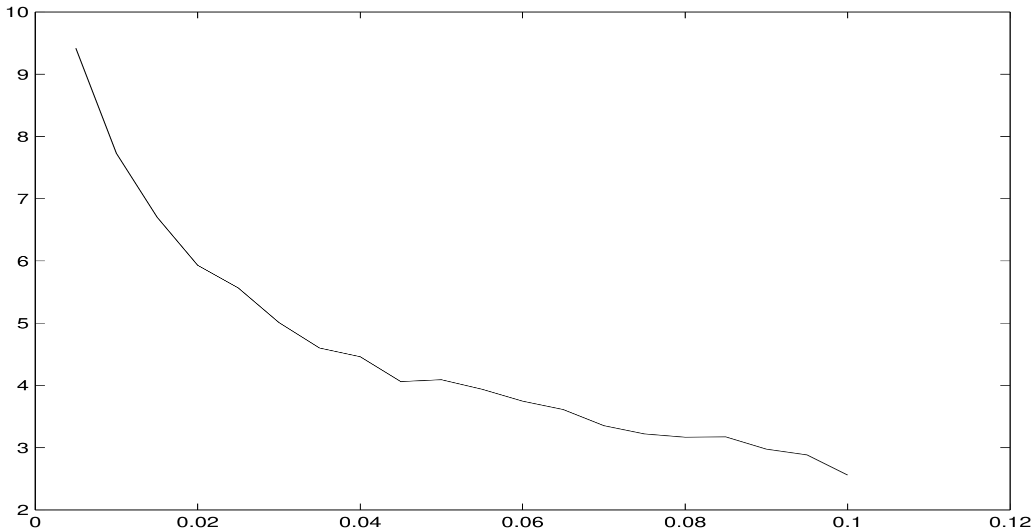

FIG. 4.4 – Entropie totale du flot selon x et du flot selon y en fonction du pas de quantification

τ	R	MSE	$RMSE$	$PSNR$
1	1	0.15	0.4	56.4
2	0.5	0.6	0.8	50.5
3	0.33	1.3	1.1	47.1
4	0.25	2.1	1.5	44.8
5	0.2	2.8	1.7	43.6
6	0.17	3.0	1.7	43.3
7	0.14	3.2	1.8	43.1
8	0.13	3.8	2.0	42.3
9	0.11	4.7	2.2	41.5
10	0.1	4.5	2.1	41.6
11	0.091	5.6	2.4	40.6
12	0.083	5.7	2.4	40.5

TAB. 4.3 – Évolution de la distorsion (MSE) en fonction du taux de compression du flot.

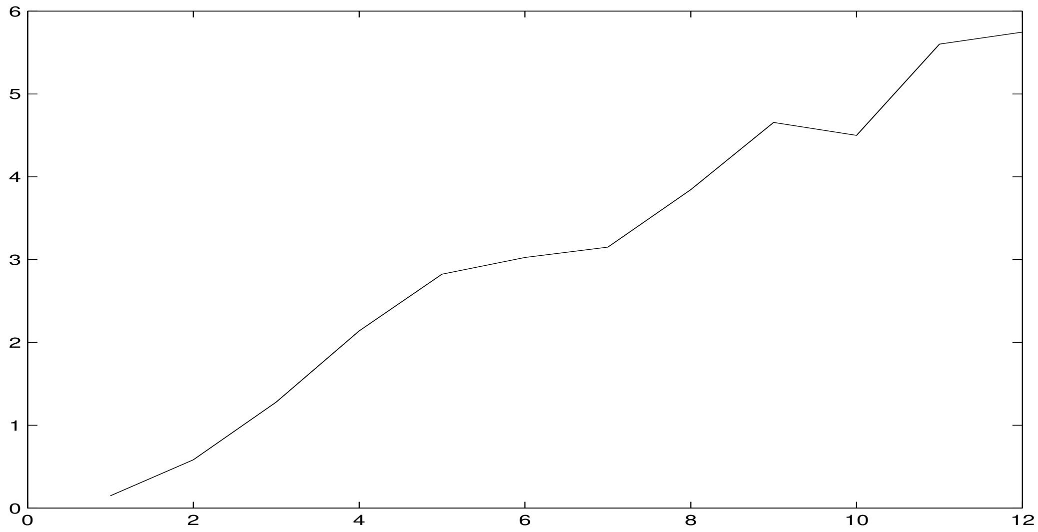

FIG. 4.5 – Évolution de la distorsion (MSE) en fonction du taux de compression du flot.

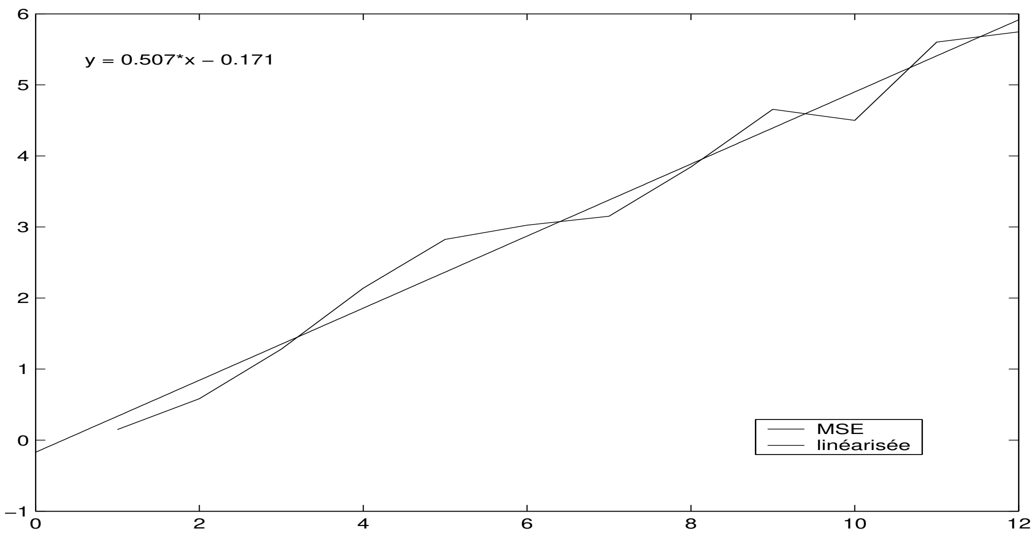

FIG. 4.6 – Linéarisation de la distorsion

Modélisation

Revenons à l'expression de la distorsion et appelons D_t la distorsion entre la t-ième image de la séquence originelle et la t-ième image de la séquence compressée. Cette distorsion est la somme de la distorsion D_E introduite par la compression de l'image d'erreur et par la compression D_F du flot, quelle que soit la méthode employée, et chacun de ces termes dépend du nombre de bits alloués au codage de l'information. Soit R_1 le nombre de bits alloués à l'image d'erreur et R_2 le nombre de bits alloués au codage du flot. On veut alors minimiser la distorsion D avec $R_1 + R_2 = R$ fixé. Utilisons la méthode de Lagrange pour minimiser :

$$D(R_1, R_2, \lambda) = D_E(R_1) + D_F(R_2) + \lambda(R_1 + R_2 - R) \tag{4.27}$$

Au minimum, on a donc

$$D'_E(R_1) = D'_F(R_2) \tag{4.28}$$

Avec les expressions précédentes, en fixant l'objectif de compression, c'est à dire $R = R_1+R_2$, on obtient alors la répartition optimale entre la compression du flot et la compression de l'image d'erreur. Prenons par exemple un objectif de $R = 1500$ octets pour coder une image d'erreur et les deux flots, c'est à dire un facteur de compression de 30 environ pour la séquence du taxi. Alors, en travaillant avec l'échelle 1 du flot, les données sont :

$$R_1 = \frac{L * l}{\tau}$$
$$R_2 = \frac{L * l}{8\tau} = R - R_1$$

où L et l sont respectivement la longueur et la largeur de l'image.

On résout alors l'équation précédente en remplaçant R_2 par $R - R_1$ et on obtient la valeur de R_1 optimale, puis de R_2. En pratique, la courbe n'est pas suffisamment bien approchée par la droite, ce qui fait que la valeur de la dérivée est difficilement calculable. Cependant, au cours des calculs effectués, on obtient un resultat cohérent lorsque la valeur obtenue pour R_2 est telle que la dérivée est bien approchée par la pente de la droite. Par exemple, pour la séquence *taxi*, lorsque l'objectif final est de $R = 2000$ octets, on trouve $R_1 = 660$ et $R_2 = 670$, ce qui correspond aux valeurs que l'on trouve par tatonnements.

Par contre, la courbe 4.4 est suffisamment régulière pour pouvoir l'utiliser : elle donne la correspondance entre le nombre de bits utilisés pour coder le flot (selon x et selon y) en fonction du pas de quantification. On peut donc fixer un objectif de compression en fixant le nombre de bits utilisés, fixer le taux de compression de l'image d'erreur donc le nombre de bits utilisés pour la coder et en déduire grâce à cette courbe et au nombre de bits restant à allouer le pas de quantification à utiliser pour respecter le taux de compression totale. Prenons un exemple : une image de le séquence *taxi* occupe $256 * 190 = 48640$ octets. S'il on veut se ramener à une occupation de 1600 octets soit un facteur de compression total de 30, alors on peut décider de limiter la compression de l'image d'erreur entre 60 et 84. Pour 66, par exemple, l'image d'erreur occupera alors 736 octets, il reste donc 864 octets pour le flot. D'apres la courbe 4.4, on prendra alors un pas de quantification de 0.035. En faisant varier le taux de compression de l'erreur et en adaptant le pas de quantification, on peut alors repèrer la combinaison qui permet de minimiser la distorsion. Les résultats pour la séquence *taxi* sont resumés dans le tableau 4.4

Dans ce cas précis, la répartition optimale consiste à allouer 705 octets à l'image d'erreur et 895 octets au codage du flot. On obtient alors une séquence où la distorsion est optimale. Cette méthode permet de trouver une répartition optimale, mais elle prend enormément de temps. Par conséquent, elle ne peut pas être utilisée, la première approche est la seule viable.

compression erreur	octets erreur	octets flot	bits par pixel(flot)	Δ	MSE pour les images		
					n^o 2	n^o 5	n^o 8
60	810	790	4.2	0.042	12.4	16.4	19
63	772	828	4.4	0.04	12.7	16.9	19.9
66	736	864	4.5	0.035	12.5	16.4	19.7
69	705	895	4.7	0.032	12.5	15.8	19.2
72	676	924	4.9	0.03	12.9	16.6	20
75	649	950	5	0.029	13	17.1	21
78	624	976	5.1	0.027	13	17.5	22.5
81	600	1000	5.3	0.025	13	17.2	22
84	579	1021	5.4	0.022	13.2	17.5	21.7

TAB. 4.4 – La séquence taxi compressée 30 fois avec différentes répartitions des bits.

Conclusion

L'objectif de cette étude était d'arriver à comprimer suffisamment un signal vidéo pour pouvoir le transmettre en temps réel ou rapidement via des reseaux à débit limité. Pour y parvenir nous avons essayé d'utiliser un certain nombre de résultats de la théorie des ondelettes, en particulier pour pouvoir exploiter les possibilités offertes par l'algorithmes de calcul du flot optique et par le programme de compression d'images fixes.

Lorsque la théorie ne permettait plus d'obtenir de nouveaux résultats, l'expérimentation et de multiples calculs ont parfois donné des pistes exploitables.

En conclusion, ce travail nous a permis d'aboutir à des taux de compression de l'ordre de 30 par rapport à la séquence initiale, ce qui nous rapproche du standard MPEG, avec une qualité visuelle bonne. Le temps nécessaire à la compression d'une séquence d'une vingtaine d'images reste toutefois un peu trop élevé. une des amèliorations envisageables serait de tirer davantage parti de la nullité presque partout du flot optique, par exemple avec un codeur par transformée en ondelettes qui accepte des images de taille plus petite.

Bibliographie

[1] Stéphane Mallat, *A Wavelet tour on signal processing*, Academic Press, 1997.

[2] Stéphane Mallat, *Une exploration des signaux en ondelettes*, extraits, Editions de l'Ecole Polytechnique, 2000.

[3] Stéphane Mallat, *Traitement du signal*, Ecole Polytechnique.

[4] Christophe Bernard, *Discrete wavelet analysis : a new framework for fast optic flow computation*

[5] Aria Nosratinia and Michael T. Orchard, *Optimal unified approach to warping and overlapped block. Motion estimation in video coding*, Proc.SPIE, 1996.

[6] Paul G. Howard and Jeffrey Scott Vitter, *Pratical implementations of arithmetic coding*, Images and texts compression, 1992.

Annexes

Annexe A

Sources du programme