Evaluation de la complexité et optimisation des algorithmes de codage vidéo par maillages

Alexandre Buisson[1] Nathalie Laurent[1] Laurent Demaret[1] Olivier Sentieys[2]

[1] France Telecom R&D - DIH / HDM, CCETT
4, Rue du Clos Courtel 35512 Cesson Sevigne France
[2] Enssat-Groupe Signal Architecture-Lasti
Technopole Anticipa BP 447 - 6 rue de Kerampont
22305 Lannion Cedex France

alexandre.buisson@rd.francetelecom.fr

Résumé

Dans cet article nous présentons un schéma de codage vidéo basé maillages. Les algorithmes basés maillages peuvent constituer une alternative aux techniques actuelles comme JPEG, JPEG2000 ou MPEG4, car ils fournissent des résultats intéressants aussi bien en codage d'images fixes, qu'en codage vidéo. Toutefois, ces algorithmes et les implémentations prototypes dont nous disposons nécessitent des temps de calcul trop importants. Afin d'accélérer les développements de ces techniques et anticiper sur leur possible émergence, il s'avère indispensable de réaliser des optimisations. Nous présentons donc un schéma de codage par maillages, ainsi qu'un certain nombre d'optimisations. Les gains obtenus et les pistes encore inexplorées laissent entrevoir la possibilité d'une implémentation quasi temps réelle d'un codeur-décodeur de vidéo basé maillages.

Mots Clef

Codage Vidéo, Maillages, Complexité, Optimisations, Pentium, Profiling, JPEG, MPEG4, JPEG2000

1 Introduction

Si les performances en terme de codage Intra et Inter des outils de codage par maillages ne sont pas encore suffisantes pour menacer les standards MPEG4 ou JPEG2000, ils fournissent tout de même des résultats prometteurs, et pourraient éventuellement être combinés avec les technologies actuelles afin d'en améliorer les taux de compression. Toutefois nos algorithmes basés maillages sont très coûteux en temps calcul, et sont incapables de répondre à des contraintes temps réelles. Or Martins [1] a montré sur une application d'interpolation temporelle de trames, qu'il est possible d'implémenter des algorithmes basés maillages de manière efficace. Il utilise principalement une triangulation de Delauney rapide, des structures de données adaptées, ainsi que des instructions MMX lors de la compensation de mouvement. Nous avons donc optimisé nos propres algorithmes pour obtenir des performances similaires. Nous présentons donc en section 2 un schéma de codage vidéo par maillages. La section 3 décrit le fonctionnement des principales briques, et présente la répartition du temps d'exécution de notre prototype. Dans les sections 4 et 5, nous détaillerons les optimisations réalisées et les performances ainsi obtenues. Enfin dans la section 6 nous verrons les futures pistes à suivre pour fournir des outils de codage et de décodage par maillages quasi temps réels.

2 Schéma de codage vidéo basé maillages

Notre schéma de codage vidéo, résumé dans la figure 1, utilise les maillages pour le codage des images Intra et Inter. La séquence d'images à coder est notée S, et sa première image I_1 est codée en Intra. Cette étape est basée sur une pyramide de maillages triangulaires réguliers. Les valeurs de colorimétrie des noeuds de chaque niveau de maillages, sont optimisées pour minimiser l'erreur globale de reconstruction avec une interpolation linéaire. Enfin, sur le dernier niveau de maillages, des permutations de diagonale sont permisent si elles apportent une amélioration de la qualité. Cette méthode dite des "Eléments Finis" décrite dans [2], s'adapte bien aux basses fréquences, mais doit être complétée par une technique permettant de corriger les textures présentant des hautes fréquences. La technique choisie, mise en oeuvre et développée dans [2], est une DCT appliquée aux triangles de la hiérarchie présentant une trop forte erreur. Ces outils sont couplés avec une méthode d'estimation de mouvement décrite dans [3]. L'estimation de mouvement hiérarchique et multirésolution associe à chaque niveau de maillages, une résolution d'image. En effet combiner des maillages grossiers avec des images très filtrées et des maillages fins avec des images présentant plus de détails permet de prendre en compte de manière robuste les mouvements

globaux de forte amplitude, et les déplacements locaux. A chaque niveau, une optimisation globale des vecteurs nodaux est réalisée par une méthode de Gauss-Newton pour minimiser la DFD (Différence d'image déplacée). Le champ de mouvement $\overrightarrow{d}$ obtenu sur le dernier niveau permet d'approximer une image I_t à partir de l'image $\widehat{I_{t-1}}$ ou $\widehat{I_{t-\alpha}}$. Les images $\widehat{I_t}$ prédites peuvent alors être améliorées par un codage d'erreur résiduelle à base de DCT ou d'ondelettes.

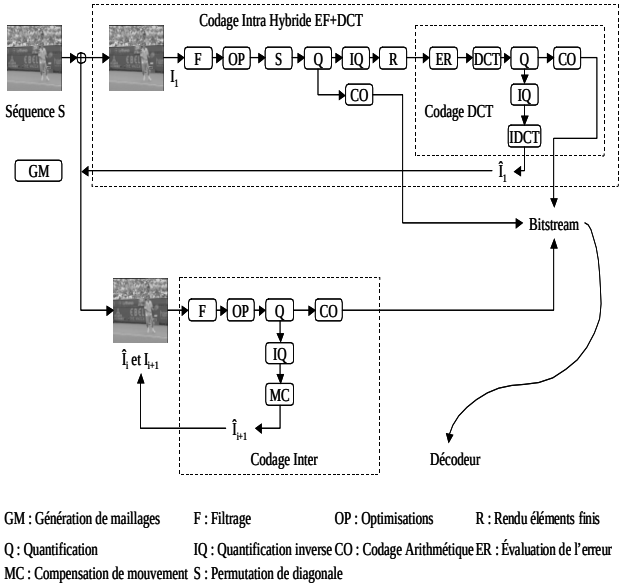


FIG. 1 – Schéma de codage vidéo basé maillage.

Les performances et la complexité de certaines briques de la figure 1 sont présentées dans la section 3.

3 Profil et complexité algorithmique

Les performances de notre codeur vidéo prototype sont étudiées sur un Pentium III 800MHz bi-processeurs, sur les 10 premières images de la séquence "susie" CIF. Ce test permet de ne pas rendre négligeable le calcul des images Intra devant celui des images Inter. Le codage est réalisé en $35.265s$, et le décodage en $6.453s$. La figure 2 montre la répartition du temps d'exécution entre les principales briques du codeur. On en déduit que les goulots d'étranglement principaux sont la génération de maillages, les filtrages, les optimisations, les rendus et les compensations.

Avant de proposer de nouvelles structures de données, et des algorithmes optimisés, nous décrivons les techniques employées dans chacun de ces goulots d'étranglements.

– L'étape de base, la création d'un maillage hiérarchique, est coûteuse en mémoire et en temps calcul. Notre prototype utilise un triangulation de Delauney implémentée selon [4] ayant une complexité en $NLogN$ grâce à une stratégie "Diviser pour mieux régner".

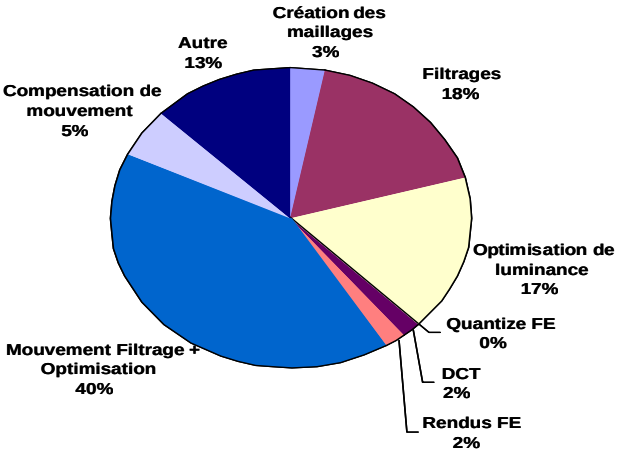


FIG. 2 – Profil du prototype de codeur vidéo basé maillages.

– Le coût par pixel du filtrage passe bas utilisé (sinus cardinal ou gaussien) pour la création de la multirésolution dans l'estimateur de mouvement, est présenté dans le tableau 1.

	multiplication	addition	accès mémoire
Filtrage	6	4	8

TAB. 1 – Filtrage : nombre d'opérations par pixel

– L'optimisation des valeurs de chaque noeud (luminance ou mouvement), implique de minimiser l'erreur de reconstruction ou la DFD. Ce problème des moindres carré est résolu par une méthode directe dans le cas de l'optimisation Intra, et de manière itérative (Gauss-Newton) dans le cas du mouvement. Il est donc nécessaire pour l'Intra de résoudre un système linéaire $Ax = b$ de dimension $N \times N$, et dans le cas du mouvement de résoudre à chaque itération un système $Ax = b$ de dimension $2N \times 2N$ (N nombre de noeud du maillage) décrit dans [5]. Le coût de l'initialisation de ce système, est présenté dans le tableau 2. La plupart des éléments du système étant nuls, une implémentation par matrice profil permet un stockage et une résolution efficace [6]. La factorisation et la résolution, conduit à une complexité globale de $np^2 + 7np + 2n$ flops et n racines carrées, ou n est la dimension de la matrice profil et p sa demi-largeur de bande.

	multiplication	division	addition
Optimisation L2	15	3	15
Estimation de mouvement	48	4	39

TAB. 2 – Remplissage de A et b : nombre d'opérations par pixel

– Afin de réaliser l'approximation d'images par la méthode des éléments finis, nous minimisons la norme L^2 pour obtenir les valeurs nodales de luminance et de mouvements optimales pour chacun des sommets du maillage. Nous utilisons alors le polynome d'interpolation de Lagrange pour en déduire les valeurs associées à chaque pixel. Cela nécessite de parcourir l'ensemble des pixels de chaque triangle, et de pondérer les valeurs nodales en fonctions des coordonnées barycentriques des pixels. Les coûts associés à ces procédures sont synthétisés dans le tableau 3.

	multiplication	division	addition
Rendu FE/triangle	4	...	15
Rendu FE Intra/pixel	9	3	9
MC/pixel	12	7	3

TAB. 3 – Coût de l'interpolation sur une composante image (Y) : nombre d'opérations par pixel

– Enfin, la technique de correction de l'erreur résiduelle du mode Intra utilisant la DCT s'avère également d'une complexité très élevée. En effet, les matrices à traiter peuvent être de taille très variable, ce qui implique l'usage d'algorithmes très généraux. Notre prototype utilise donc une méthode de calcul direct dont la complexité est en N^4.

4 Optimisations

Nous avons donc choisi d'optimiser notre prototype sur une cible Pentium III 800MHz à 2 processeurs. Effectivement si les applications du traitement d'image et de vidéo sont souvent développées et optimisées pour des plateformes spécifiques ARM ou DSP, les processeurs généralistes utilisés dans les stations de travail PC, sont des candidats potentiels pour les applications multimédia et plus particulièrement pour les applications de codage vidéo. En effet ils deviennent de plus en plus puissants, et enfin ils offrent à l'heure actuelle plusieurs niveaux de parallélismes décrit en détails dans [7].

Les optimisations présentées portent donc sur 4 briques de notre application de codage : la génération de maillages, l'interpolation d'images, notre opérateur de filtrage, et enfin la brique de codage par DCT basée triangle.

4.1 Génération de maillages hiérarchiques

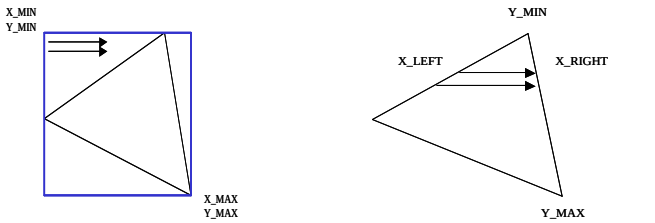
La création de la hiérarchie de maillages a été optimisée suivant 3 axes : les structures de données, la minimisation de l'espace mémoire utilisé, et le choix de la technique de triangulation. Afin de pouvoir manipuler efficacement la pyramide de maillages, nous avons conçu de nouvelles structures de données. La nouvelle implémentation se compose de 3 tableaux qui contiennent respectivement tous les noeuds T_NOEUD, tous les triangles $T\_TRIANGLE$, et tous les maillages $T\_MAILLAGE$. Chaque maillage se définit par ses nombres de points, et de triangles, mais aussi par 2 pointeurs, un sur le premier élément de T_NOEUD et un autre sur un triangle de $T\_TRIANGLE$. Ce procédé qui permet d'éviter une réplication inutile des noeuds, implique d'inclure des informations de hiérarchie dans la structure de donnée des noeuds. Afin de construire la pyramide, on réalise en premier lieu l'allocation du nombre de triangles et de points nécessaires en fonction des paramètres initiaux. Nous avons pour cela précalculé le nombre de points et de triangles, ainsi que les coordonnées (x, y) des points des grilles de base. A partir du nombre de triangles et de points de la grille initiale choisi, et du nombre de division souhaité, l'équation d'Euler nous permet de calculer itérativement le nombre de triangles et de points nécessaires à toute la pyramide.

Les positions (x, y) des points, permettent l'initialisation des noeuds de T_NOEUD appartenant au maillage initial. L'algorithme de triangulation de ce niveau de base est complètement déterministe, pour chaque noeud i, on trouve par un simple décalage basé sur le nombre de noeuds par ligne, les noeuds j et k nécessaires à la création des triangles de base. Enfin, la création du voisinage des triangles et des noeuds est nécessaire après chaque création de niveau, pour simplifier la manipulation des structures de données, et simplifier l'algorithme de subdivision. Ainsi chaque triangle connaît ses 3 voisins, et chaque point connaît les triangles auxquels il appartient ainsi que les noeuds qui l'entourent. L'algorithme de division 1 vers 4 que nous utilisons, adaptée de la méthode décrite par Loop dans [8], est constitué de 2 étapes. Dans la première, on connaît le nombre de noeuds à insérer, ainsi que la position du premier des nouveaux noeuds. On parcour donc la liste des triangle du niveau N_i, afin d'insérer des noeuds sur leurs arêtes. Un nouveau noeud de T_NOEUD est initialisé sur une arête A du triangle T seulement si le triangle voisin de T avec lequel il partage A n'a pas été divisé. Chaque triangle à la fin de cette phase contient ces noeuds sommets, ainsi que les nouveaux noeuds portés par ses arêtes. Ceci permet dans la deuxième étapes, de parcourir chaque triangle, pour créer ces 4 triangles fils et les relations hiérarchiques les reliant.

4.2 Rendus éléments finis

L'interpolation des valeurs de colorimétrie (YUV) associées à chaque pixel nécessite un certain nombre d'opérations non réductible. Toutefois nous avons pu optimiser la technique de parcours des pixels. En effet, nous avons remplacé les parcours par boite englobante 3.a par un parcours classique utilisé en 3D pour le remplissage de polygone 3.b. Ceci permet de supprimer des boucles, et de nombreux calculs de coordonnées barycentriques inutiles. De plus les triangles sont parcourus plusieurs fois dans le mode Intra (Rendus FE, Evaluation de l'erreur, DCT). Il nous a donc semblé judicieux que les triangles puissent garder les informations sur les pixels leur appartenant, même si cela implique une augmentation de l'occupation mémoire. De

ce fait le parcours de polygone n'est réalisé que si un triangle a changé d'aspect.



(a) Boite englobante. (b) Remplissage de polygone

FIG. 3 – Différentes méthodes de parcours des pixels appartenant à un triangle.

4.3 Filtrage et multirésolution d'images

Les processeurs Pentium disposent depuis le Pentium II de jeux d'instructions permettant un parallélisme de données(SIMD : Single Instruction Multiple Data). Les instructions MMX sont aujourd'hui très répandus dans les applications de traitements d'images sur PC, et l'implémentation d'un opérateur de filtrage linéaire séparable en est une application classique décrite dans [9]. Notre filtrage utilisé pour la création d'une multirésolution de luminance est une cible idéale pour le jeu d'instructions MMX. Pour l'instant le fitrage est décomposé en 2 étapes, une sur les lignes, une autre sur les colonnes. Dans chaque étape nous chargeons 3 vecteurs de 8 pixels simultanéement, et on utilise 6 multiplications, 4 additions et 2 décalages binaires pour obtenir 8 valeurs filtrées. Toutefois, la symétrisation du signal aux bords des images nécessite tout de même quelques décalages et copies de registres supplémentaires. Des optimisations sont encore possibles, en effet la vectorisation et la précision de ce filtrage pourraient être encore améliorées par l'usage d'instructions SSE dont les registres sont plus longs, et les opérations plus précises.

4.4 La DCT basée triangle

Cette brique décrite en détail dans [2] nécessite pour chaque triangle, une transformation affine de sa texture, une DCT réalisée sur une matrice symétrique, un parcours en zigzag de la partie inférieure de la matrice de coefficients DCT, une quantification, et enfin le codage des coefficients DC et AC obtenus, par un codeur arithmétique adaptatif. Les implémentations de codeur à base de DCT (JPEG et MPEG) peuvent être optimisée car la taille des blocs utilisés (8x8) est fixe. Mais, dans notre cas, les dimensions de nos matrices sont très variables (de 8x8 à 50x50). Nous avons donc testé plusieurs voies pour obtenir une solution rapide. La technique retenue utilise des tables précalculées de coefficients DCT et de positions ZigZag, pour chaque taille de matrice. Nous avons également implémenté les DCT et IDCT en MMX et SSE pour les blocs 8x8, mais elles ne seront utilisées que si les dimensions de matrices sont faibles. Pour finir, le compilateur fourni par Intel, permet de générer automatiquement des codes optimisés (MMX ou SSE) pour certaines boucles. Nous avons donc utilisé cette fonctionnalité pour améliorer les performances de notre méthode.

5 Performances

Nous présentons ici les résultats des tests de comparaison des performances de chaque brique prise indépendamment. Les tests sont réalisés sur un Pentium III 800Mhz à 2 processeurs. L'aspect multi-processeur n'ayant pas été abordé, toute la charge de l'exécution d'une tache a été placée sur un seul processeur. Notre prototype est noté "SPOT_REF" et nos briques optimisées "SPOT_DEV".

Le tableau 4 présente les résultats obtenus sur plusieurs tests. La création de maillages hiérarchiques est équivalente sur un faible nombre de niveaux (<3), mais elle est déjà 8 fois plus rapide à partir de 6 niveaux. Les performances de notre opérateur de filtrage sont environ 50 fois supérieures à celle du prototype. Quant à l'évaluation de nos procédures de rendus par éléments finis, elle montre un ratio compris entre 2 et 3, mais elle ne tient pas compte de notre optimisation décrite dans la partie 4.2, qui devrait accélérer les parcours suivants dans le cas du codage Intra.

	SPOT_REF	SPOT_DEV
Génération de maillages 5x5 6 niveaux	8.124	1.14
Filtrage CIF 4 niveaux	0,401561	0,0075736
Filtrage QCIF 4 niveaux	0,0986934	0,00198423
Interpolation d'images CIF 5x5 6 niveaux	0,562	0,25
Interpolation d'images QCIF 5x5 4 niveaux	0,094	0,031

TAB. 4 – Comparaison des performances exprimées en seconde de la génération de maillages, du filtrage et du rendu

La figure 4 montre les performances des 2 implémentations dans le cas d'une génération de maillages hiérarchiques sur 4 niveaux à partir d'une grille 5x5, le rendu éléments finis de l'image "Susie" au format CIF, et sa correction par le passage de tous les triangles du niveau 3 dans la procédure de codage par DCT basée triangle. Cette figure résume les gains obtenus sur chaque brique optimisée, et on peut également mesurer sur le tableau 5 l'effet de chaque optimisation décrite dans la partie 4.4, et en particulier le gain provenant de la vectorisation automatique de boucle.

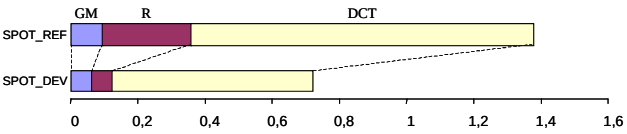


FIG. 4 – Comparaison des perfomances en secondes de la génération de maillages, du rendu éléments finis, et de la DCT avec quantification, zigzag et codage arithmétique.

	Temps (s)
Prototype	1.016
Tables	0.813
RDCT NLOGN	0.859
RDCT NLOGN + Vectorisation	0.735
Tables + Vectorisation	0.578

TAB. 5 – Evaluation des performances en seconde des différentes optimisations

6 Conclusions et Perspectives

Les gains obtenus, sur chacunes des briques présentées, par des optimisations d'algorithmes, de structures de données, et d'implémentation, montrent que le respect de contraintes temps réelles pour des applications à base de maillages est possible. Certaines optimisations comme les outils MMX et SSE développés au cours de ces travaux sont réutilisables pour d'autres applications (Filtrage MMX, et DCT/IDCT MMX et SSE). De plus, de nombreuses pistes se dégagent. Elles portent principalement sur l'optimisation de la génération des voisinages, sur la résolution des systèmes linéaires, mais aussi sur le parallèlisme multiprocesseur.

Références

[1] Fernando C.M. Martins. Real-time frame rate adaptation based on warping of edge-preserving meshes.

[2] N. Laurent P. Alliez and P. Lechat. Mesh coding : application to texture and scalable 3d scene coding. 55 :117–130, 2000.

[3] Nathalie Laurent. Hierarchical mesh-based global motion estimation, including occlusion areas dectection. Sept 2000.

[4] J.R. Shewchuk. Triangle : Engineering a 2d quality mesh generator and delauney triangulator. 1996.

[5] P. Lechat, M. Ropert, and H. Sanson. Hierarchical Mesh-based Motion Estimation Using a Differential Approach and Application to Video Coding. 4 :2081–2084, Sept. 8-11 1998.

[6] Gene H.Golub and Charles F.van Loan. *Matrix Computation Second Edition ISBN : 0-8018-3739-1*.

[7] Eric Debes et Fulvio Moschetti. Applications vidéo sur des systèmes à base de processeurs généralistes : L'exemple de l'encodeur mpeg2.

[8] Charles Teorell Loop. *Smooth Subdivision Surfaces Based On Triangles*. PhD thesis, 1987.

[9] Intel Corporation. *The complete guide to MMX Technology ISBN : 0-07-006192-0*.