UNIVERSITÉ
DE REIMS
CHAMPAGNE-ARDENNE

Compression multi-vues des flux auto-stéréoscopiques

THÈSE

présentée et soutenue publiquement le

pour l'obtention du

Doctorat de l'Université de Reims Champagne-Ardenne (URCA)

(Spécialité Informatique)

par

Benjamin BATTIN

Composition du jury

Rapporteurs :	M^me^ Luce MORIN	Professeur, INSA de Rennes
	M. Frédéric DUFAUX	Directeur de Recherche, Télécom ParisTech
Examinateurs :	M. Christophe RENAUD	Professeur, Université du Littoral Côte d'Opale
	M. Yannick REMION	Professeur, URCA
Invité :	M. Didier DEBONS	OPEX Media
Directeurs :	M. Laurent LUCAS	Professeur, URCA
	M. Philippe VAUTROT	Maître de Conférences, URCA

CReSTIC-SIC – EA 3804

Remerciements

Je tiens dans un premier temps à adresser mes remerciements à madame Luce Morin ainsi que monsieur Frédéric Dufaux pour l'intérêt qu'ils ont porté à ces travaux en acceptant de les rapporter. Je remercie également monsieur Christophe Renaud pour avoir accepté de participer au jury.

Cette thèse a été cofinancée par l'agence nationale de la recherche (ANR) et la région Champagne-Ardenne. Je souhaite remercier Didier Debons, Michel Frichet et Florence Debons pour leur implication dans le projet Cam-Relief. Merci également à Laurent Lucas et Yannick Rémion de m'avoir donné la possibilité d'effectuer cette thèse. Celle-ci m'a permis d'élargir grandement le champ de mes connaissances mais aussi de m'apporter plus de rigueur dans mon travail.

Je remercie mes encadrants, Laurent Lucas et Philippe Vautrot pour leur suivi ainsi que leurs conseils durant ces trois années. Ceux-ci ont toujours été disponibles pour m'apporter leur aide quand j'en avais besoin et ce, malgré des emplois du temps chargés.

Merci à tous mes collègues de l'équipe du CReSTIC avec lesquels j'ai toujours eu de bonnes relations. Merci à mes amis de voyage, Jessica, Aassif et Cédric avec lesquels je suis parti deux années de suite à San Francisco. Cette expérience a été très enrichissante pour moi et m'a permis de goûter (voir d'abuser) de la gastronomie locale. Je remercie également Stéphanie, Gilles, Céline et Doudou pour le temps consacré à la relecture de mes articles et à l'amélioration de mon anglais écrit.

Merci à tous mes anciens collègues ex-3DTVs notamment Romain, Romuald, Vincent, Pierre-Adrien, Robert avec lesquels j'ai partagé de bons moments notamment lors de nos soirées sur Reims. Merci aussi à Napnap avec qui j'ai effectué la majeur partie de mes études universitaires et ai passé des soirées rémoises que je qualifierais d'épiques.

Enfin je souhaite remercier ma famille et notamment mes parents qui ont toujours cru en moi (même dans des moments où cela n'a pas dû être évident) et m'ont toujours poussé à continuer mes études afin de pouvoir exercer un métier qui me passionne. Si j'en suis arrivé jusqu'ici aujourd'hui, c'est grâce à vous.

Table des matières

Table des figures

Liste des algorithmes

Liste des tableaux

Introduction

Vidéo et TVHD 3D

L'extension de contenus visuels à la troisième dimension, comme la capture d'une scène dynamique en 3D en générant une double image de celle-ci sur un site distant en temps réel, a longtemps été considérée comme des faits relevant de la science fiction. C'est aujourd'hui une réalité collectivement désignée sous le terme de télévision en trois dimensions (3DTV ou TVHD 3D). Ce nouveau type d'images dites en 3D relief permet de créer l'illusion d'un environnement réel en son absence. Au plan technologique, leur fabrication cible toute la chaîne de production d'images qui couvre aussi bien les moyens d'acquisition que les moyens de compression/transmission et de représentation. Depuis le succès du film "Avatar", l'acronyme "3D" est devenu l'argument marketing de tous les grands industriels de l'audiovisuel mais sans pouvoir affranchir le téléspectateur du port de lunettes. Pour voir en relief sans lunettes, de nombreux verrous scientifiques restent encore à dépasser comme par exemple, celui de la prise de vue temps réel d'images réelles suivant des normes qualitatives comparables aux productions d'images relief de synthèse.

Les recherches menées sur ce nouveau type d'images s'appuient sur différents domaines scientifiques allant du traitement du signal et des images au rendu interactif 3D en passant par les mathématiques appliquées et l'informatique. Les multiples verrous technologiques, qui pourront être levés à l'aide des différents domaines précédemment cités, sont répartis sur quatre axes majeurs : la captation de ces signaux 3D, la représentation et le codage de ceux-ci, leur diffusion ainsi que leur restitution sur des dispositifs adaptés.

Compression vidéo multi-vues : vers de nouveaux paradigmes

De nos jours, la compression est une brique fondamentale dans le domaine de la diffusion de médias numériques (images, vidéos). Le poids mémoire de ces médias est intrinsèquement lié aux capacités du dispositif de restitution. Or, les progrès technologiques constants en terme d'écran permettent d'atteindre des résolutions d'images de plus en plus hautes. On peut d'ores et déjà trouver, par exemple, dans le commerce des écrans QFHD (Quad Full HD) proposant un affichage à une résolution de 3840×2160 pixels tandis que la norme UHD (Ultra HD, permettant un affichage à une résolution de 7680×4320 pixels, voir Sugawara (2008), Shishikui *et al.* (2009)) commence à faire son apparition. Parallèlement à cela, on constate aussi que la fréquence d'affichage des écrans ne cesse d'augmenter, ainsi, la majorité des écrans sont dotés d'une fréquence d'affichage à 120Hz.

Afin de mieux mettre en évidence le caractère indispensable de la compression pour la diffusion de médias numériques, prenons un exemple simple : considérons une séquence vidéo Full HD (de résolution 1920×1080 pixels) à 30 images/seconde, d'une durée de 1h30 et dont chaque pixel est codé sur 24 bits. Le poids mémoire total pour une telle séquence est donc de : (1920 × 1080 × 30 × 5400 × 24) / 8 = 938,56 Go soit un débit 1,39 Gbits/s. Ce volume/débit de données est multiplié par 4 (respectivement par 8) si la résolution passe en QFHD (respectivement UHD). Sans cette étape de compression, le stockage ou la transmission de tels médias serait purement impossible, et ce, même avec l'arrivée sur le marché de supports de stockage tels que le Blu-Ray, ou dans le cas de la transmission, de fournisseurs d'accès proposant des offres fibre optique garantissant des débits allant jusqu'à 10 Gbits/s.

Ces différentes observations se vérifient d'autant plus dans le cas de médias stéréoscopiques comme auto-stéréoscopiques. En effet, une séquence vidéo standard est constituée d'un flux vidéo

unique alors que si l'on considère une séquence vidéo stéréoscopique (resp. auto-stéréoscopique), celle ci est constituée de 2 (resp. N où $(N \geq 2)$) flux vidéos distincts. Dans ces deux cas, on constate généralement que le débit nécessaire pour transmettre ces séquences est lié de manière linéaire au nombre de vues en entrée. Or comme pour les écrans 2D standards, les écrans 3D auto-stéréoscopiques permettent l'affichage d'un nombre de vues de plus en plus élevé (généralement entre 5 et 9 vues voir plus) augmentant ainsi le débit nécessaire pour la transmission de ces séquences à N vues.

Toutefois, un des points important concernant les séquences vidéos multi-vues (stéréoscopiques et auto-stéréoscopiques) est que chaque frame à un instant t du flux multi-vues correspond à N $(N \geq 2)$ acquisitions simultanées d'une même scène par N capteurs disposés selon une géométrie de capture spécifique. Il existe donc de fortes redondances entre chaque vue. Ces redondances doivent être prises en compte par le système de compression afin de pouvoir "briser" la corrélation débit/nombre de vues soulignée dans le paragraphe précédent.

Contexte de la thèse

Ce travail de recherche a été effectué au sein du groupe SIC du CReSTIC dans le cadre du projet ANR CamRelief (2008-2010) en collaboration étroite avec la société 3DTV Solutions [1] reprise depuis par OPEXMedia [2]. Ce projet ciblait à la fois la spécification et le développement de matériels et logiciels dédiés à la chaîne de production d'images 3D reliefs comme le montre la figure 1.

Les verrous technologiques concernés par ce programme correspondent principalement à trois études du projet global. Elles ont porté sur la conception :

(1) d'un système de contrôle commande embarqué pour l'asservissement visuel visant à « suivre » un objet mobile en adaptant en temps réel la géométrie du dispositif ;

(2) de nouveaux systèmes de compression spécifiquement dédiés au flux multi stéréo et au flux auto-stéréoscopique ;

(3) de modules logiciels pour la reconstruction du relief et le mixage réel/virtuel, dans le but d'enrichir le contenu 3D relief diffusé sur écrans auto-stéréoscopiques.

Le but de cette thèse s'inscrit pleinement dans la brique "transmission" de la chaîne de restitution décrite par la figure 1, l'objectif étant de proposer des systèmes de codage répondant à différents besoins. Ces besoins, tous spécifiques à un usage particulier, sont les suivants :

- permettre un encodage en temps réel des flux vidéos multi-vues issus de nos périphériques de capture ;
- développer un algorithme de compression sans perte dédié au stockage des séquences multi-vues destinées à la post-production ;
- développer un algorithme de compression avec pertes (non nécessairement temps réel) qui

1. http ://www.3dtvsolutions.com
2. http ://www.opexmedia.com

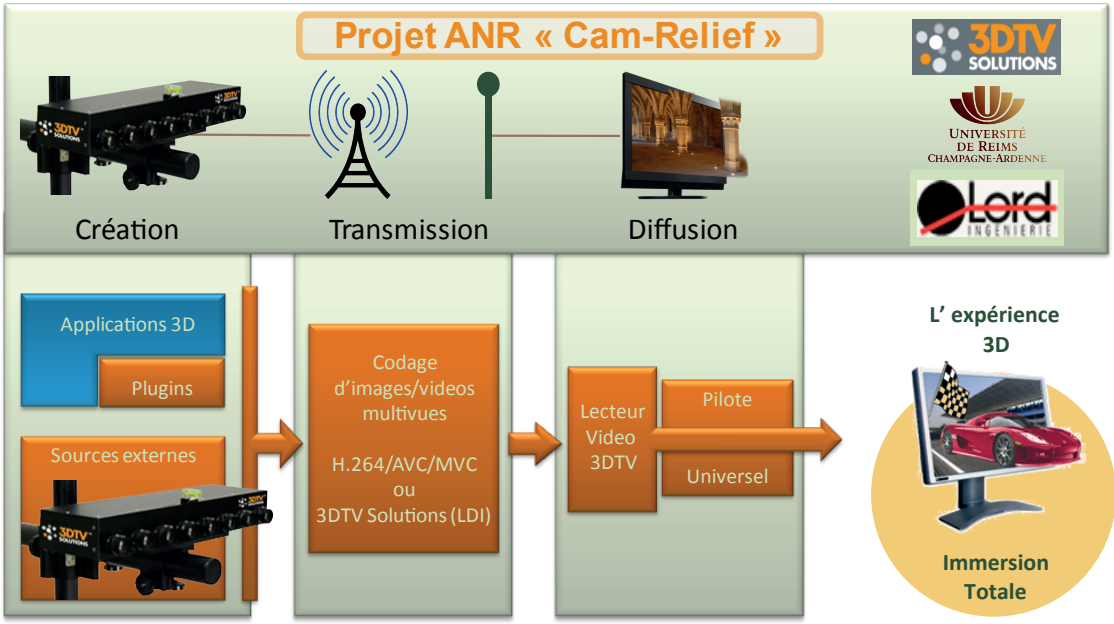


FIGURE 1 – Le projet "Cam-Relief"

constituerait une alternative aux standards de compression multi-vues actuels (H.264/MVC).

Dans ce manuscrit, nous détaillerons les différents modules logiciels développés et dédiés à la compression de flux vidéo multi-vues à des fin de stockage et de transmission à savoir :

- MICA (Multiview Image Compression Algorithm) : un algorithme à faible complexité destiné à la compression en temps réel des images et séquences multi-vues issues de nos différents dispositifs de capture.
- Multiview-LS : un algorithme de compression sans pertes permettant de réduire le volume de données occupé par des séquences multi-vues qui seront ensuite retouchées à l'aide de logiciels de post-traitement. L'absence de distorsion devra permettre une application plus aisée des différents effets sur celles-ci.
- Enfin, deux algorithmes de compression utilisant un algorithme novateur d'estimation des disparités combiné à l'approche LDI (Layered-Depth Image). Le premier est basé sur la DCT-3D alors que le deuxième repose sur la DWT et l'algorithme SPIHT. Ces deux algorithmes constituent une alternative aux techniques actuelles de codage (H.264/MVC) et permettent différentes applications possibles telles que la synthèse de points de vue intermédiaires ou la génération de meshes.

Plan du manuscrit

Ce manuscrit s'articule en trois parties. La première est une présentation générale des techniques relatives à la stéréo- ainsi qu'à l'auto-stéréoscopie. La seconde partie comporte un état de l'art des techniques de compression utilisées dans le cas d'images standards et présente nos contributions, en terme de modules de codage, basés sur ces techniques. Dans la troisième partie, les différentes méthodes spécifiques au codage de médias multi-vues ainsi que nos différentes réalisations sont décrites.

Partie I :

- Le chapitre I.1 décrit le principe de la vision binoculaire essentielle à la vision stéréoscopique et trace un bref historique des premières inventions associées à celle-ci.

- Le chapitre I.2 présente les différentes techniques de restitution stéréoscopique et auto-stéréoscopique disponibles à ce jour.

- Le chapitre I.3 est dédié à la présentation des géométries de capture associées à l'auto-stéréoscopie et présente les dispositifs d'acquisition relief développés dans le cadre du projet Cam-Relief .

Partie II :

- Le chapitre II.1 expose un état de l'art des méthodes de compression dédiés aux flux 2D standards. La description de ces méthodes, généralement utilisées par les techniques de compression adaptées au multi-vues, permettra une compréhension plus aisée pour le lecteur.

- Le chapitre II.2 présente notre première contribution : MICA (Multiview Image Compression Algorithm), un algorithme de compression avec pertes orienté temps réel adaptant son schéma de compression afin de préserver les zones soumises à l'effet parallaxe lors de la restitution d'importantes distorsions. L'algorithme est ensuite comparé au format JPEG de manière objective et subjective.

- Le chapitre II.3 décrit notre deuxième réalisation : Multiview-LS, notre module de codage multi-vues sans perte basé sur l'algorithme JPEG-LS et dédié au stockage de séquences multi-vues déstinées à la post-production. Nous confrontons ensuite les résultats obtenus par notre algorithme face à d'autres algorithmes de compression sans perte de l'état de l'art.

Partie III :

- Le chapitre III.1 présente un état de l'art des différentes approches en terme de codage multi-vues (stéréoscopiques et auto-stéréoscopiques). On détaillera les différents formats 3D disponibles ainsi que les différentes méthodes de codage associées à ceux-ci.

- Le chapitre III.2 détaille l'approche LDI (Layered-Depth Image) mise au point de manière spécifique à notre géométrie de capture. Celle-ci comprend une phase d'estimation des disparités (destinées à générer les cartes de profondeur) ainsi que la génération du LDI à proprement parler.

- Le chapitre III.3 est dédié à notre module de codage basé sur l'approche LDI précédemment décrite. Nous présenterons deux algorithmes destinés à la compression de l'information chromatique du LDI : le premier est basé sur la DCT 3D, le deuxième, quant à lui, propose une amélioration du précédent via l'utilisation de la SA-DWT et de l'algorithme SPIHT. Nous parlerons aussi des différents algorithmes possibles pour la compression de l'informa-

tion de disparité contenue dans le LDI en prenant en compte les contraintes spécifiques à notre approche. Enfin nous comparerons les résultats obtenus par nos deux approches au standard H.264/AVC.

Une conclusion et des perspectives pour de futurs travaux sont finalement présentées. Notons que les jeux de données utilisés dans ce manuscrit sont issus du site "Middlebury Stereo Vision Page"[3]. Les différentes images constituant ces jeux de données sont présentées dans l'annexe A de ce document.

3. http ://vision.middlebury.edu/stereo/

Première partie

Présentation générale de la vision multiscopique

Chapitre I.1

Perception du relief et histoire de la vision 3D

Dans ce premier chapitre, nous allons tout d'abord présenter les différents facteurs physiologiques gouvernant la perception du relief chez l'être humain. Une bonne compréhension de ces multiples mécanismes est nécessaire afin de bien comprendre les différents concepts associés à la vision multiscopique. Puis nous dresserons un bref historique relatif à l'apparition de la notion de "stéréopsie" au cours de l'histoire et des premiers systèmes de restitution associés.

I.1.1 La vision humaine

Le cerveau humain nous permet, dans la vie de tous les jours, de percevoir le relief ainsi que d'apprécier les distances. Ce mécanisme très complexe induit le fonctionnement de notre système de vision ainsi que de multiples zones de notre cerveau via plusieurs facteurs de perception. Ces différents facteurs peuvent être séparés en deux familles distinctes : les facteurs de perception monoculaires et binoculaires. Pour les premiers de ces indices, seul l'image perçue par un œil est prise en compte. On compte notamment parmi ceux-ci :

- L'accommodation : la contraction (resp. la dilatation) du cristallin lors de l'observation d'un objet proche (resp. éloigné) informe le cerveau de la distance relative de l'objet.
- La taille relative des objets : lorsque deux objets semblables sont de tailles différentes, l'observateur perçoit l'objet plus petit comme étant plus éloigné.
- Les occlusions : lorsqu'une partie d'un objet est occulté par un autre objet, l'observateur comprend que l'objet occulté est situé plus loin.

Il existe d'autres facteurs de perception monoculaire tels que la perspective, les jeux d'ombres et de lumière ainsi que le parallaxe de mouvement. Ces différents facteurs permettent une première appréhension du relief.

A l'inverse, les facteurs de perception binoculaires proviennent, eux, du travail simultané des deux yeux et permettent l'appréciation des distances. Il existe deux facteurs de perception binoculaire (qui sont étroitement liés) :

- La convergence : elle résulte de l'information transmise au cerveau lors de l'effort musculaire pour faire converger nos deux yeux vers un même objet. Cette information, comme pour l'accommodation, permet de renseigner le cerveau quant à la distance de cet objet.
- La parallaxe interoculaire : tout corps humain (normalement constitué) possède deux yeux espacés en moyenne de 65 mm. Du fait de la position différente de chaque œil, les différents objets de la scène ne sont pas toujours situés au même endroit dans les images formées sur chaque rétine. Ce décalage possible entre la position d'un même objet sur les deux images est appelé parallaxe interoculaire. Lorsque la parallaxe interoculaire d'un objet est positive (resp. négative), alors celui-ci est situé derrière (resp. devant) le plan de collimation (aussi appelé plan "écran" et correspondant à une parallaxe nulle) des deux yeux et fournit à l'observateur une impression de profondeur (resp. jaillissement).

L'ensemble des facteurs de perception binoculaires associés à l'activité du cerveau humain est désigné par le terme "stéréopsie" (du grec *stereo* "solide" et *opsie* "vision"). C'est sur cette notion, et plus précisément sur la parallaxe interoculaire, que sont basés tous les systèmes de restitution stéréoscopiques et auto-stéréoscopiques. Sachant qu'il est important de noter que 5 à 10% de la population présente une déficience de la "stéréopsie" pouvant être causée par plusieurs facteurs tel que le strabisme. Les personnes concernées peuvent donc uniquement appréhender les distances et le relief par le biais de facteurs de perception monoculaire présentés ci-dessus et en conséquence, ne percevront pas plus de relief sur les différents dispositifs de restitution 3D que sur un écran standard.

I.1.2 Un peu d'histoire...

On retrouve des traces de notion de vision binoculaire très loin dans l'histoire. Dès le 3$^{\text{ième}}$ siècle avant J.C., Euclide (grand mathématicien grec), définit la vision binoculaire : "Voir le relief, c'est percevoir au moyen de chaque œil l'impression simultanée de deux images dissemblables du même sujet". De même Léonard de Vinci, en 1484, précise les principes régissant la vision binoculaire.

Au XVIeme siècle, G. B. Della Porta et J. Chimenti (Fig. I.1.1) réalisent des dessins d'une même scène sous des angles différents. Même si on ne peut pas considérer ces œuvres comme de la stéréoscopie, en effet nous n'avons aucune preuve que ces dessins étaient destinés pour cet usage, la restitution en relief reste cependant possible.

Il a fallu attendre jusqu'en 1832 pour voir apparaître les premières techniques de stéréovision. A cette époque, l'anglais Charles Wheatstone (1802 - 1875) étudie les différents moyens permettant de percevoir le relief. En 1838, il définit la notion même de "stéréopsie" et réalise ainsi son premier stéréoscope constitué de deux miroirs réfléchissant les deux images placées aux extrémités (Fig. I.1.2).

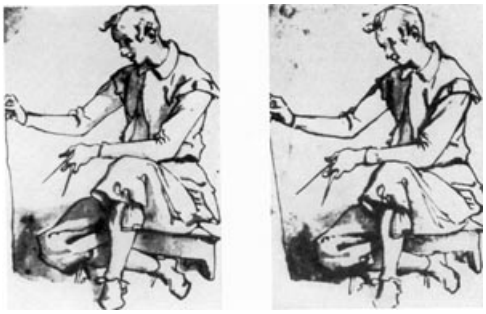


FIGURE I.1.1 – Les deux dessins de J. Chimenti

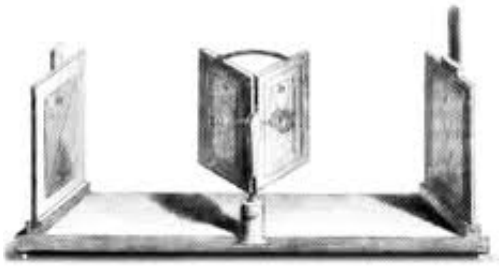


FIGURE I.1.2 – Le stéréoscope de Wheatstone

Peu de temps après, Sir David Brewster (1781 - 1868) a l'idée de remplacer les deux miroirs du stéréoscope de Wheatstone par un couple de lentilles prismatiques (ou non selon le type d'image à visionner) et crée ainsi un stéréoscope portable (Fig. I.1.3(a)) moins encombrant et plus simple à utiliser. Cette idée est reprise par le physicien américain Oliver Wendell Holmes (1809 - 1894) en 1850 pour la réalisation de son stéréoscope à main (Fig. I.1.3(b)).

(a) Le stéréoscope de Brewster (b) Le stéréoscope de Holmes

FIGURE I.1.3 – Les différents types de stéréoscopes

A partir de cette période, les techniques de vision stéréoscopique n'auront de cesse d'évoluer et de connaître un nouvel essor grâce notamment à l'invention du cinéma en 1895 par les frères Lumière, des films photographiques couleur en 1935 avec l'Agfacolor ainsi que de la photographie numérique en 1981. Nous pensons pouvoir dire sans nous tromper que le 21ème siècle sera sans doute l'ère de la 3D, grâce à la production de films tels que Avatar, à la démocratisation des écrans 3D utilisant des lunettes actives et l'apparition, dans les prochaines années à venir, des

premiers modèles d'écrans auto-stéréoscopiques pour le grand public. Pour un historique plus détaillé de la vision stéréoscopique, le lecteur peut se référer à Michel (2012).

Chapitre I.2

Systèmes de restitution relief

Il existe de nombreuses technologies permettant l'affichage de contenus 3D. Celles-ci, présentées dans la figure I.2.1, peuvent être séparées en deux branches distinctes : les systèmes de diffusion "réels", qui permettent l'affichage dans des volumes 3D et les systèmes de diffusion "illusoires", qui eux, exploitent les principes régissant la "stéréopsie" afin de tromper le cerveau humain et fournir à l'utilisateur une perception 3D à partir d'images 2D.

Ce dernier type de dispositifs présente des avantages notamment en terme de facilité de génération des médias reliefs mais aussi vis à vis du coût de production des systèmes de restitution associés. Le projet Cam-Relief, présenté dans l'introduction générale de ce document, étant mené dans un cadre industriel, celui-ci privilégie les techniques de restitution relief accessibles au grand public : les systèmes binoculaires à lunettes et les systèmes auto-stéréoscopique (correspondant à la partie "Diffusion" de la figure 1). La suite de ce chapitre sera donc dédiée à la présentation des différents dispositifs de restitution associés à ces deux type de technologies.

I.2.1 Systèmes de restitution stéréoscopiques

De nombreux systèmes de restitution stéréoscopiques permettent à l'utilisateur de visionner des scènes en 3D. Cette perception du relief (toujours basée sur le principe de la vision binoculaire) est rendu possible via l'utilisation de lunettes spécifiques aux procédés de différenciation des deux images constituant le couple stéréoscopique (détaillés ci-après).

I.2.1.1 Les anaglyphes

Ce procédé a été introduit par W. Rollman en 1853, puis repris par Louis Ducos du Hauron qui finit de le mettre au point en 1891. En 1935, l'ingénieur français Louis Lumière, réalise

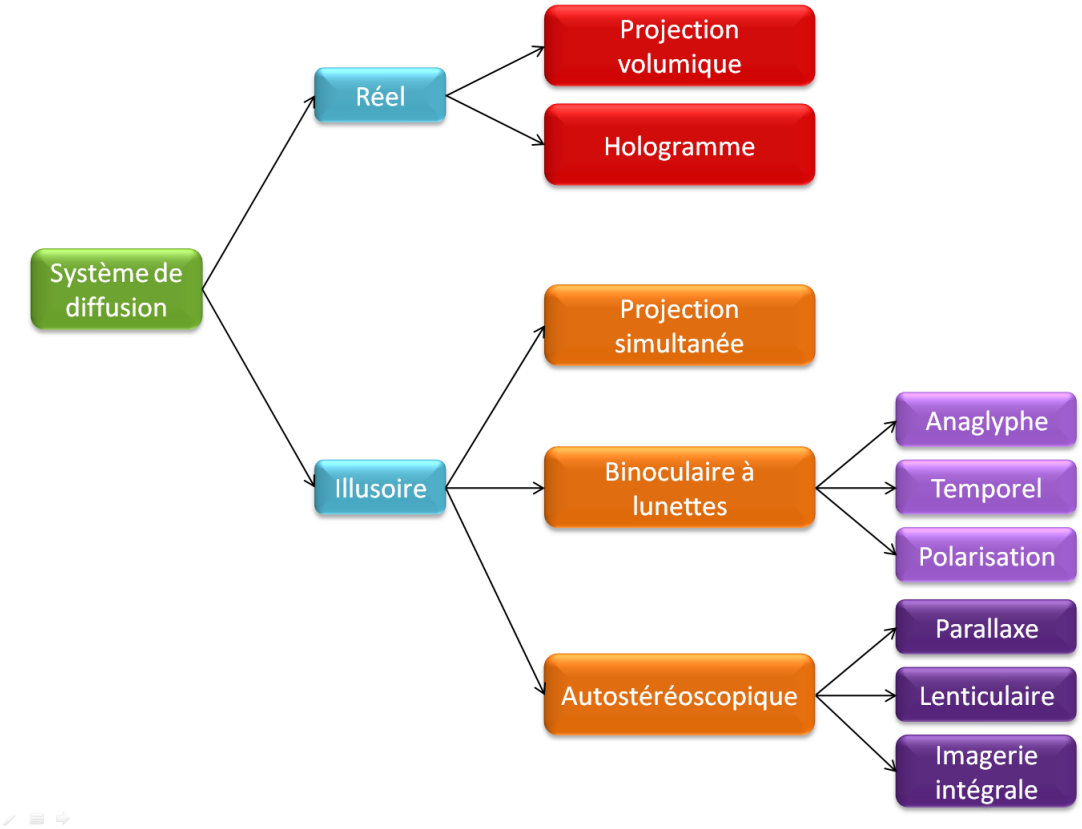


FIGURE I.2.1 – Les différents systèmes de restitution relief.

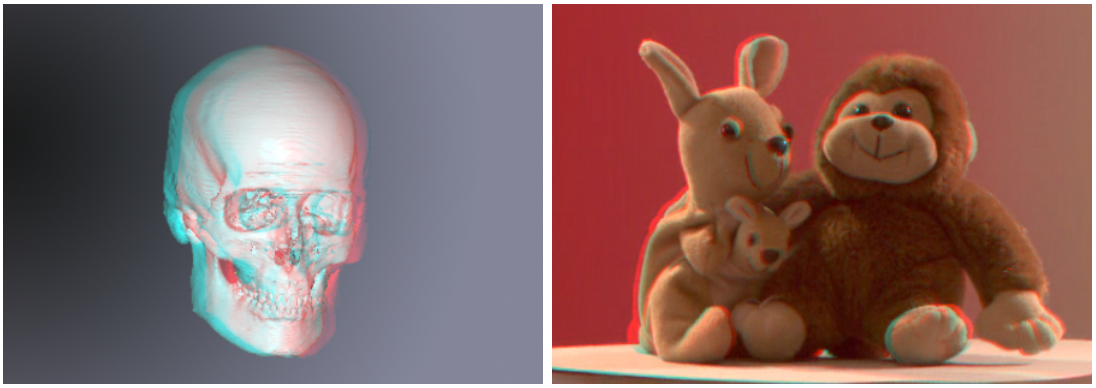
un *remake* en anaglyphes du célèbre court-métrage "L'arrivée d'un train en gare de La Ciotat" produit 40 ans plus tôt par son frère et lui.

Un anaglyphe est la superposition des deux vues constituant le couple stéréoscopique. Une de ces deux vues est projetée en utilisant une couleur déterminée, l'autre vue (dite homologue) est quant à elle projetée en utilisant la couleur complémentaire dans l'espace RGB de la précédente. Généralement on utilise le couple de couleurs suivant : rouge pour l'œil gauche et cyan (vert + bleu) pour l'œil droit. Les différentes zones de ces deux vues sont décalées suivant leur éloignement par rapport au plan physique de l'image (effet parallaxe), permettant ainsi à l'utilisateur d'apprécier les distances de celles-ci.

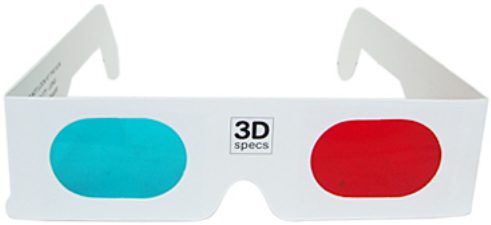
La restitution 3D à partir d'un anaglyphe s'effectue par synthèse soustractive des couleurs (d'où l'emploi de couleurs complémentaires) via l'utilisation de lunettes à filtres colorimétriques. L'œil gauche, sur lequel est placé le filtre rouge, ne perçoit donc que l'image cyan et inversement pour l'œil droit. Le décalage est ensuite interprété par le cerveau afin de former l'image 3D.

Sur la figure I.2.2, on peut voir une paire de lunettes à filtres rouge-cyan ainsi que deux anaglyphes utilisant ce couple colorimétrique.

Les anaglyphes étant relativement simples à produire et les lunettes peu coûteuses, ce procédé est surtout utilisé dans l'industrie du loisir ou de la publicité. Toutefois, l'utilisation de l'espace



(a) Deux anaglyphes rouge-cyan



(b) Une paire de lunettes rouge-cyan

FIGURE I.2.2 – Deux anaglyphes rouge-cyan et sa paire de lunettes associé

colorimétrique pour restituer le relief dénature la couleur originale des différents objets constituant la scène (surtout si ceux-ci ont la même couleur que l'un des deux filtres) et ce procédé ne peut donc pas être utilisé pour des images ou de la vidéo couleur.

I.2.1.2 Les lunettes polarisantes

Ce principe de restitution relief utilise la propriété d'orientation de la lumière : la polarisation. Comme on peut le voir sur la figure I.2.3, il est possible de donner une certaine orientation à une onde lumineuse (la polarité). Lorsque cette onde traverse un filtre polarisant, soit la polarité du filtre est identique à celle de l'onde et dans ce cas elle peut se propager, soit, lorsque la polarité du filtre est différente, l'onde ne peut pas passer à travers celui-ci.

Les systèmes de restitution utilisant cette technologie, généralement les cinémas, utilisent un couple de projecteurs munis de filtres de polarités différentes projetant simultanément les deux images du couple stéréoscopique, tandis que les spectateurs sont munis de paires de lunettes équipées de verres à polarités différentes (Fig. I.2.4). Chaque œil va donc percevoir l'image de même polarité que son verre et permettre à l'utilisateur de percevoir le relief.

Le problème majeur de cette approche est le coût élevé du système de projection (écran argentique et les deux projecteurs munis de filtres) et c'est pourquoi il est le plus souvent utilisé dans des salles de cinémas ou dans les parcs d'attraction.

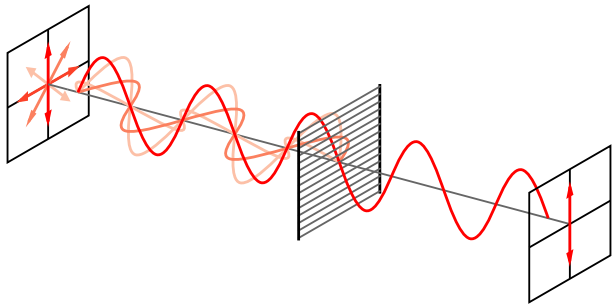


FIGURE I.2.3 – Polarisation d'une onde lumineuse selon plusieurs directions et propagation de l'onde polarisée verticalement à travers le filtre associé.

FIGURE I.2.4 – Une paire de lunettes polarisées

Depuis plusieurs années, l'utilisation d'un seul projecteur a été rendu possible grâce au système Real D. Celui-ci utilise un unique projecteur qui envoie alternativement l'image gauche et l'image droite couplé à un alternateur de polarité synchronisé avec le projecteur et qui permet d'associer chaque image envoyée avec sa polarité correspondante. Le principal avantage de ce système est l'utilisation de la polarisation circulaire qui permet au spectateur d'incliner la tête sans voir apparaître d'effet fantôme permettant ainsi une visualisation moins contraignante ; toutefois, comme pour les lunettes actives, l'alternance temporelle de chaque image diminue la luminosité.

Il existe aussi des écrans à diffusion polarisée qui diffusent simultanément les deux images du couple stéréoscopique et permettent une polarisation (verticale ou horizontale suivant le modèle) des deux images. Ces moniteurs sont équipés d'un filtre appliqué sur l'écran qui polarise une ligne (resp. colonne) de pixels. Les deux images étant projetées simultanément, aucun phénomène de clignotement n'est ressenti par le spectateur et la luminosité est conservée. Toutefois, la polarisation s'effectuant une ligne (ou colonne) sur deux, les images doivent être entrelacées divisant ainsi leur résolution par deux sur une des dimensions.

I.2.1.3 Les lunettes actives

Cette technique utilise des lunettes alternantes à cristaux liquides (aussi appelées *shutter glasses* en anglais) synchronisées avec un système de projection (moniteur, télévision ou projecteur) haute fréquence et permet une répartition temporelle des deux images constituant le couple stéréoscopique. Chaque image (gauche et droite) est projetée alternativement à une fréquence

donnée (par exemple 60 Hz) et les cristaux liquides constituant chaque verre de la paire de lunettes passent alternativement d'un état transparent à un état opaque de manière synchrone au système de projection, permettant ainsi à l'œil gauche et à l'œil droit de voir uniquement l'image qui leur est destinée (ie. une image sur deux).

Ce type de restitution existe déjà depuis de nombreuses années mais était limité par la fréquence relativement faible des écrans cathodiques de l'époque (notamment pour le standard PAL à 50 Hz) ce qui rendait l'expérience relativement désagréable. En 1999, nVidia sort une solution basée sur ses cartes TnT/TnT2 couplées avec des lunettes ELSA Revelator (Fig. I.2.5(a)) pouvant supporter des fréquences allant de 50 à 140 Hz, mais encore une fois, la fréquence supportée par les moniteurs CRT de l'époque ne permettait pas une visualisation confortable. Depuis 2007, nVidia commercialise un kit 3D-Vision (Fig. I.2.5(b)) permettant une restitution sur des écrans 120 Hz (60 Hz pour chaque œil) à l'aide de lunettes synchronisées par capteur infrarouge.

(a) ELSA Revelator (b) nVidia 3DVision

FIGURE I.2.5 – Différents types de lunettes actives

Ce type de lunettes, qui doit être parfaitement synchronisé avec le dispositif de projection, embarque de l'électronique et par la même occasion une alimentation ce qui peut le rendre lourd et relativement coûteux à l'achat (autour de 100 euros). Il nécessite un système de projection à haute fréquence afin d'éviter tout phénomène de clignotement. Toutefois, cette technique permet une restitution du relief sans altération de l'espace colorimétrique (comme c'était le cas avec les anaglyphes) et utilise les images d'entrée à pleine résolution assurant ainsi une qualité optimale. Celle-ci a d'ores et déjà été retenue pour la norme Blu-ray 3D et permet au grand public de disposer d'une solution de restitution 3D de qualité chez eux.

I.2.2 Systèmes de restitution auto-stéréoscopiques

Toutes les techniques de stéréoscopie détaillées dans la section précédente nécessitent le port de lunettes spécifiques (anaglyphes, alternantes ou polarisées). Il est toutefois possible de percevoir le relief directement à l'écran sans avoir recours à des lunettes grâce à l'intégration directement au niveau de l'écran d'un dispositif de séparation des vues. Ce type de procédé, appelé

auto-stéréoscopie, permet ainsi l'affichage de médias N vues (typiquement 2, 5, 8 ou 9). L'utilisateur pourra alors visionner *N-1* couples stéréoscopiques différents suivant sa position relative à l'écran. En effet, lorsque celui-ci se déplace latéralement, l'angle formé par les champs de vision de chaque œil et l'écran varie et le dispositif de séparation des vues permet le passage à un autre couple stéréoscopique. Pour cela, les N vues en entrée sont entrelacées : chaque vue est multipliée par un filtre (spécifique au constructeur de l'écran) qui va permettre d'occulter certaines composantes de pixels de celle-ci. Ensuite toutes ces vues filtrées vont être tout simplement additionnées avant projection sur l'écran auto-stéréoscopique. La figure I.2.6 illustre le processus d'entrelacement sur trois vues. Pour plus de détails concernant les écrans auto-stéréoscopiques (à barrière de parallaxe ou à réseau lenticulaire) et leur fonctionnement, le lecteur pourra se référer à Dodgson (2005), Matusik et Pfister (2004), Halle (2005) et Hirsch et Lanman (2010).

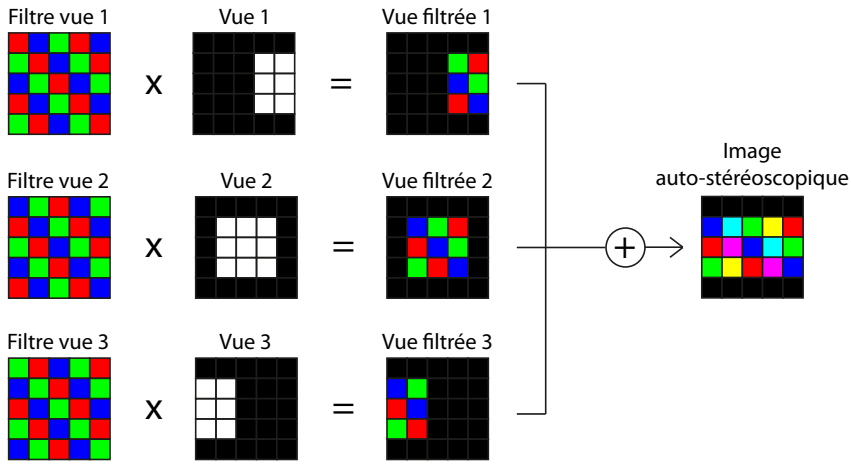


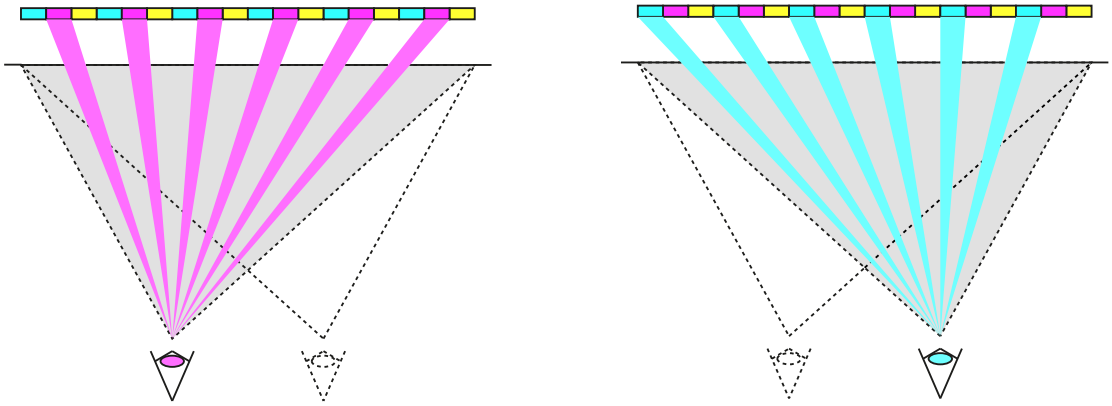
FIGURE I.2.6 – Principe d'entrelacement d'une image auto-stéréoscopique

Dans les deux parties suivantes, nous allons illustrer les deux types d'écrans auto-stéréoscopiques actuellement sur le marché : les écrans à barrière de parallaxe et les écrans à réseau lenticulaire.

I.2.2.1 Ecrans auto-stéréoscopiques à barrière de parallaxe

Ce dispositif, introduit par Berthier (1896) puis repris par Ives (1902), est un filtre composé de zones verticales opaques et de zones verticales transparentes distribuées alternativement. Ce filtre est placé directement devant la dalle et permet d'occulter certaines zones de l'écran (vues) tout en laissant d'autres visibles. Les parties visibles de l'écran sont déterminées par l'angle d'incidence. Lorsque le spectateur se déplace devant un écran auto-stéréoscopique à barrière de parallaxe, l'angle d'incidence de son champ de vision change lui permettant ainsi de percevoir ce qui n'était pas visible précédemment (voir Fig. I.2.7).

Ce type d'écran souffre tout de même d'un inconvénient majeur : chaque œil ne perçoit que $1/N$ du nombre de pixels total, cela ayant pour effet de grandement réduire la luminosité de l'image perçue (et ce même en réglant la luminosité de la dalle au maximum) ainsi que sa résolution. Ce phénomène devient encore plus gênant lors de la projection d'images 2D standards. De plus, plus le nombre de points de vue est élevé plus la largeur des bandes opaques du filtre



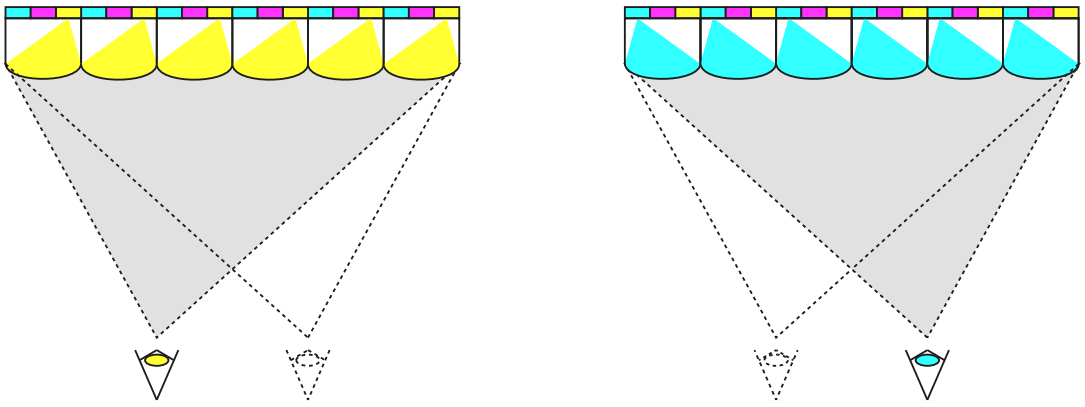
(a) Vision de l'œil gauche (b) Vision de l'œil droit

FIGURE I.2.7 – Vision auto-stéréoscopique sur écran à barrière de parallaxe

doit être grande ce qui réduit encore la luminosité.

I.2.2.2 Ecrans auto-stéréoscopiques à réseau lenticulaire

Le principe du réseau lenticulaire a été introduit par le physicien français Gabriel Lippman en 1908, puis repris par Walter Hess en 1915. Il est basé sur l'utilisation de micro-lentilles sphériques directement apposées sur l'écran. Ces micro-lentilles vont dévier les rayons lumineux de la dalle et ainsi adresser un couple stéréoscopique à l'observateur suivant sa position. Contrairement aux écrans à barrière de parallaxe qui sont basés sur le masquage de certaines zones de la dalle, les écrans à réseau lenticulaire conservent au maximum la luminosité de l'image et du coup, permettent l'utilisation d'un plus grand nombre de vues. La figure I.2.8 schématise le fonctionnement général des écrans à réseau lenticulaire.



(a) Vision de l'oeil gauche (b) Vision de l'oeil droit

FIGURE I.2.8 – Vision auto-stéréoscopique sur écran à réseau lenticulaire

Ce principe est également utilisé dans le cadre des technologies dites par “imagerie intégrale” (Lanman *et al.* (2010), Holografika, Kim *et al.* (2010), Wetzstein *et al.* (2011)) qui constituent actuellement l'une des perspectives les plus prometteuses en matière d'affichage 3D autostéréoscopique.

Chapitre I.3

Systèmes d'acquisition relief

Nous avons vu dans la section précédente les différents systèmes de restitution stéréoscopique et auto-stéréoscopique. Afin d'exploiter ces différents systèmes, il faut être en mesure de produire des contenus adaptées à ceux-ci. On distingue généralement deux types de médias : les images ou séquences en images de synthèse produites à l'aide d'outils de type 3DS Max (ou autres) couplés à un système de caméras virtuelles afin de produire les N vues ($N \geq 2$), et les images (ou séquences) issues de scènes réelles. Pour ces dernières, il existe déjà des dispositifs d'acquisition stéréoscopiques destinés à l'usage professionnel (pour le cinéma par exemple) ou orientés grand public tel que l'appareil photo Fuji Real 3D. Dans le cas de l'auto-stéréoscopie, le nombre de points de vue est généralement compris entre 2 et 9.

Un des aspects du projet Cam-Relief consiste à développer (en collaboration avec l'entreprise 3DTVSolutions) des systèmes d'acquisition auto-stéréoscopiques afin de pouvoir générer différents médias associés à cette technologie. Dans cette section, nous allons tout d'abord décrire les géométries de capture possibles associées à l'auto-stéréoscopie, puis nous présenterons les différents systèmes d'acquisition élaborés dans le contexte du projet Cam-Relief.

I.3.1 Les différentes géométries de capture

Dans cette section, nous présentons les différentes géométries de capture généralement utilisées lors de l'acquisition de médias auto-stéréoscopiques. Pour une étude plus approfondie de ces géométries, le lecteur peut se référer aux travaux de Prévoteau *et al.* (2010) et de Yamanoue (2006).

I.3.1.1 Géométrie convergente

Ce type de géométrie utilise N caméras dont les axes optiques convergent tous vers un unique point appelé "centre de convergence" et dont la distance entre les centres optiques et le centre de convergence est appelée "distance de convergence" (désignés respectivement par $C_{convergence}$ et $D_{convergence}$). La figure I.3.1 illustre la géométrie de capture convergente pour un système d'acquisition contenant 3 vues : la distance entre chaque centre optique (*dico*) est constante, les vecteurs verticaux (*upvector*) sont parallèles entre eux et sont tous perpendiculaires au plan contenant les centres optiques et le centre de convergence. Les centres optiques peuvent être alignés soit sur une même droite soit sur un arc de cercle.

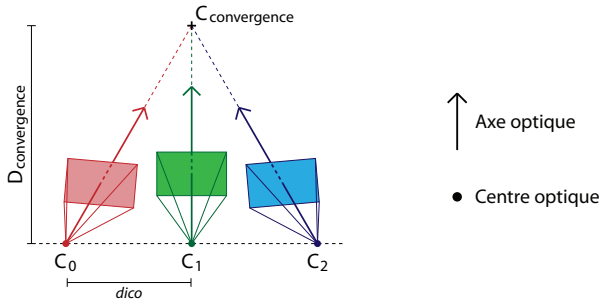


FIGURE I.3.1 – Géométrie convergente

I.3.1.2 Géométrie parallèle

Contrairement à la géométrie convergente, la géométrie parallèle utilise des capteurs dont les centres optiques sont toujours alignés sur une même droite (appelée "droite des centres optiques"). Les axes optiques des divers capteurs étant parallèles entre eux, la distance de convergence est donc, dans ce cas, infinie. La figure I.3.2 illustre ce type de géométrie.

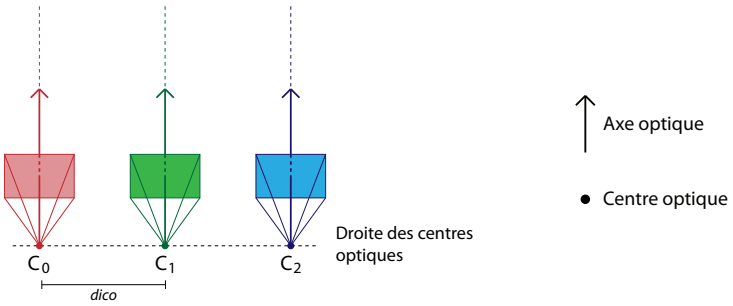


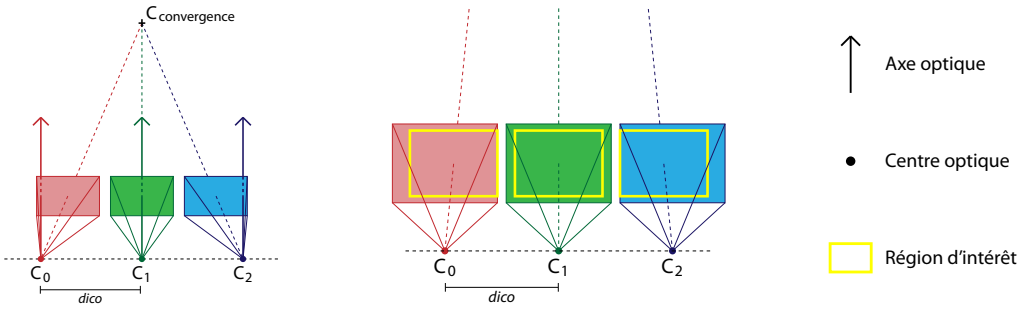
FIGURE I.3.2 – Géométrie parallèle

L'exploitation des séquences acquises selon une géométrie parallèle ne peuvent pas être directement exploitées pour affichage sur un écran auto-stéréoscopique, le centre de convergence étant situé à l'infini, l'impression de relief par jaillissement est impossible (voir Yamanoue (2006)). Pour pallier à ce problème, une rectification des images est possible afin de se ramener à une géométrie parallèle décentrée (voir la figure I.3.3(b) de la section suivante) en découpant chaque

image pour n'en garder qu'une région d'intérêt (dépendante à la fois de la position du capteur mais aussi de la position du centre de convergence).

I.3.1.3 Géométrie parallèle décentrée

Comme pour la géométrie parallèle, les axes optiques des différents capteurs sont parallèles entre eux. Toutefois, le centre de convergence n'est pas situé à l'infini car les capteurs utilisés sont particuliers : leur pyramide de vision est décentrée, ainsi, la droite reliant le centre optique du capteur au centre de convergence passe par le centre de l'image produite. La figure I.3.3(a) illustre la géométrie de capture parallèle décentrée.



(a) Géométrie parallèle décentrée

(b) Principe de rectification des images par ROI

FIGURE I.3.3 – Géométrie parallèle décentrée et principe de rectification par région d'intérêt

I.3.2 Rail photo

Le premier système d'acquisition développé dans le cadre du projet Cam-Relief est le rail photo (figure I.3.4). Celui-ci est constitué d'un appareil photo disposé sur un rail muni d'un moteur et se déplaçant latéralement afin de produire N vues.

Ce dispositif repose sur la géométrie de capture parallèle, la distance entre chaque prise de vue est constante et l'axe optique de l'appareil est perpendiculaire à l'axe du rail. Afin de permettre un décentrement logiciel lors de la production de l'image auto-stéréoscopique, on utilise un capteur haute définition de telle sorte à ce que les régions d'intérêt une fois extraites gardent une bonne résolution.

FIGURE I.3.4 – Le rail photo

La contrainte majeure vis à vis du rail photo est que les différentes vues sont capturées à des instants différents, ce dispositif est donc destiné à l'acquisition de scènes statiques.

I.3.3 Caméras multi-vues

Le projet Cam-Relief a aussi donné lieu au développement de deux prototypes de caméras multi-vues 8 vues : l' Octo-cam et la caméra Banc-titre (voir respectivement les figures I.3.5(a) et I.3.5(b)).

(a) L'octocam (b) La caméra Banc-titre

FIGURE I.3.5 – Nos deux prototypes de caméras : (a) l'Octocam, (b) la Banc-titre

Ces deux caméras reposent sur une géométrie de capture décentrée grâce à l'utilisation de capteurs optiques décalés, les images en sortie de ces caméras pouvant ensuite être directement entrelacées pour les afficher sur un écran auto-stéréoscopique. Ces capteurs sont aussi équipés de filtres de Bayer (BE et Company (1975)) qui permettent la capture d'une seule composante couleur (R, G ou B) par pixel. Ce dispositif qui permet de réduire le flux de pixels provenant des caméras doit aussi être pris en compte pour la conception de nos différents algorithmes de compression dédiés à celles-ci.

Les deux prototypes qui ont des distances inter-centres optiques différentes (et donc des distances de convergence différentes) sont destinées à des usages différents :

- La Banc-titre, destinée à capter un petite scène (env. 0.027 m^3), utilise des capteurs dont les centres optiques sont séparés d'une distance d'environ 35 mm (la distance de convergence résultante étant de 1.64 m).
- L'Octocam permet la capture d'une scène plus grande (env. 1 m^3) et est destinée à des applications de type visio-conférence. La distance entre chaque centre optique est d'environ 59 mm (la distance de convergence résultante étant de 4.2 m).

Conclusion

Cette première partie a permis une présentation générale de la vision stéréoscopique (et auto-stéréoscopique) ainsi que des diverses techniques associées à celles-ci.

Dans le premier chapitre, nous avons présenté de manière succincte le principe de la vision binoculaire, pierre angulaire de la vision stéréoscopique, et exposé les principales dates associées à l'apparition des premiers dispositifs exploitant la vision binoculaire.

Nous avons ensuite décrit les différents systèmes de restitution relief. On peut généralement séparer ceux-ci en deux familles distinctes : Les systèmes de restitution stéréoscopiques qui nécessitent le port de lunettes spécifiques à la méthode de restitution (anaglyphe, lunettes actives ou passives) afin de pouvoir apprécier le relief et les systèmes de restitution auto-stéréoscopiques qui permettent l'affichage simultané d'un certain nombre de couples auto-stéréoscopiques. Le dispositif de restitution étant directement intégré à l'écran, l'utilisateur n'est pas contraint par le port de lunettes.

Le dernier chapitre était consacré aux systèmes d'acquisition relief. Nous avons tout d'abord détaillé les différentes géométries de capture possibles associées à l'auto-stéréoscopie puis nous avons présenté les différents systèmes d'acquisition développés dans le cadre du projet Cam-Relief (en collaboration avec la société 3DTVSolutions).

Dans la prochaine partie, nous allons aborder le cœur de cette étude doctorale en commençant tout d'abord par développer l'adaptation multi-vues des méthodes de compression monoscopique. Nous dresserons tout d'abord un état de l'art relatif à ces méthodes, puis nous présenterons nos diverses contributions résultant de l'adaptation de celles-ci au cas multi-vues.

Deuxième partie

Adaptation multi-vues des méthodes de compression monoscopique

Chapitre II.1

État de l'art

Le premier chapitre de cette partie dresse un état de l'art des méthodes les plus utilisées pour la compression d'images 2D. Bien que cette thèse porte sur la compression d'images et de vidéos auto-stéréoscopiques, on constate généralement que les approches adoptées pour ce type de médias utilisent des schémas dérivés de méthodes de compression destinées aux images 2D. Il est donc important de présenter ces différents algorithmes afin de fournir une meilleure compréhension au lecteur. Nous parlerons dans un premier temps des méthodes de compression sans perte, puis avec pertes, pour finir sur les principaux concepts associés à la compression vidéo.

II.1.1 Méthodes sans perte

Cette section présente les différentes techniques de compression d'images sans perte les plus classiques. La compression sans perte reconstruit l'information à l'identique de la source et ne génère ainsi aucune distorsion du signal d'entrée. Nous avons choisi de détailler ces méthodes car celles-ci devront être prises en considération dans la section II.2.2.3 pour la compression des vues de référence de l'algorithme MICA ainsi que pour la compression des cartes de disparités dans la section III.3.2.

II.1.1.1 Méthodes associées au codage entropique

Les méthodes présentées ici sont relativement rudimentaires, mais restent toutefois fortement utilisées par les algorithmes de compression d'images avec ou sans pertes afin de produire un train de bits en fin de traitement (Huffman ou codage arithmétique par exemple).

II.1.1.1.1 RLE : Run-Length Encoding

Le codage RLE (pour Run-Length Encoding) est basé sur la répétition de symboles. Les images contiennent généralement des zones uniformes dans lesquelles les pixels ont la même valeur. Le codage RLE vise donc à exploiter ces répétitions en générant des symboles du type (*longueur*, *valeur*) et ainsi réduire le volume des données.

Examinons le comportement de l'algorithme sur l'exemple II.1.1.

156	148	148	164
148	164	156	156
156	156	156	164
156	156	156	156

Tableau II.1.1 – Exemple de signal issu de l'image Lena

Le poids mémoire de cette séquence de pixels est de 16 octets. La représentation de cette partie d'information après application de l'algorithme devient :

(1, 156), (2, 148), (1, 164), (1, 148), (1, 164), (5, 156), (1, 164), (4, 156)

La séquence de pixels pèse maintenant 16 octets soit aucun gain en terme de compression. En effet, le gain obtenu par le codage des répétitions va rapidement être limité par l'encodage des pixels non répétés. Dans le pire des cas (aucune répétition au sein de l'image), la taille du fichier compressé sera le double du fichier original. Une représentation plus adéquate de cette séquence de pixels devra être :

156, *2*, 148, 164, 148, 164 ,*5*, 156, 164, *4*, 156

De cette façon le code binaire généré pèse 11 octets soit un gain de 7 octets. Le problème est maintenant de pouvoir différencier les symboles de longueur (en italique) de ceux représentant directement la valeur du pixel. Pour cela, diverses solutions sont possibles (par exemple l'ajout d'un bit supplémentaire par coefficient) et sont laissées au soin de l'utilisateur. Cet algorithme de compression, qui est relativement performant lorsque que le signal d'entrée présente beaucoup de répétitions, est utilisé dans de nombreux formats d'images tels que le format BMP, ou alors couplé avec un codage de Huffman pour le codage entropique des coefficients AC du format JPEG. Cet algorithme sera notamment repris en section II.2.2.2 pour le mode run-length de notre algorithme MICA.

II.1.1.1.2 Le codage de Huffman

Le codage de Huffman (1952) est basé sur les statistiques et sur le codage de Shannon-Fano (Fano (1961)). Cet algorithme va encoder les différents symboles constituant l'alphabet de départ

en utilisant des codes binaires à longueur variable. La longueur de chacun de ces codes va donc être déterminée d'après les statistiques globales de l'alphabet et est inversement proportionnelle à la probabilité d'apparition d'un symbole donné (cette probabilité est calculée à partir du nombre d'occurrences de ce symbole et de la longueur totale du message à coder).

L'algorithme se déroule en quatre phases :
1) Génération des statistiques : le signal d'entrée est entièrement parcouru et les probabilités d'occurrence de chaque symbole constituant l'alphabet sont calculées.
2) Construction de l'arbre de Huffman : A partir de chaque symbole, on crée un nœud auquel on affecte comme poids la probabilité d'occurrences du symbole. Puis on regroupe les 2 nœuds dont le poids est le plus faible pour ne constituer qu'un seul nouveau nœud dont le poids est égal à la somme des poids de ses deux fils. On réitère le processus jusqu'à n'obtenir qu'un seul nœud qui constitue alors l'arbre de Huffman.
3) Génération des codes de l'arbre : Pour chaque feuille (symbole de l'alphabet), on génère le code binaire à longueur variable associé, en remontant l'arbre et en concaténant par la gauche le bit 0 (resp. 1) pour un lien "fils gauche" (resp. "fils droit") au code binaire.
4) Génération du train de bits : On remplace chaque symbole contenu dans le fichier à compresser par le code binaire associé à celui-ci.

Ci-dessous, on peut trouver les résultats de l'exécution de chaque phase de l'algorithme à partir de l'exemple II.1.1. Le tableau II.1.2 présente les probabilités associées aux différents symboles de l'alphabet, la figure II.1.1 illustre l'arbre généré lors de la phase 2 et le tableau II.1.3 répertorie les différents codes binaires associés à chaque symbole de l'alphabet.

Symbole	Probabilité
148	0.1875 (3/16)
156	0.625 (10/16)
164	0.1875 (3/16)

Tableau II.1.2 – Statistiques des symboles de l'exemple II.1.1

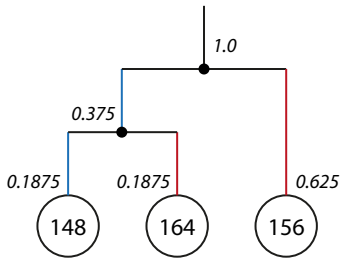


Figure II.1.1 – Arbre de Huffman généré à partir de l'exemple II.1.1

Symbole	Code de Huffman
148	00
156	1
164	01

Tableau II.1.3 – Codes de Huffman générés à partir de l'exemple II.1.1

Le codage de Huffman est considéré comme presque optimal vis-à-vis de l'entropie du signal

d'entrée (nombre de bits minimum par symbole pour représenter celui-ci). Considérons un signal S contenant des symboles provenant d'un alphabet Ω comptant n symboles dont les probabilités d'occurrences dans S sont données par $p_1, p_2, ..., p_n$. L'entropie de Shannon pour ce signal, notée $H(S)$, est alors définie par l'équation II.1.1.

$$H(S) = -\sum_{i=1}^{n} p_i \times \log_2(p_i) \tag{II.1.1}$$

On peut alors démontrer que si on considère L la longueur moyenne du code de Huffman pour le signal S, alors elle vérifie l'équation II.1.2.

$$H(S) \leq L < H(S) + 1 \tag{II.1.2}$$

Celle-ci nous montre bien que le codage de Huffman tend à se rapprocher au maximum de l'entropie de la source à 1 bit/symbole près. Cela est dû majoritairement au fait que la mesure d'entropie n'est pas une mesure entière, or la longueur minimum du code associé à un symbole est de 1 bit. Cela devient d'autant plus gênant lorsque, par exemple, l'entropie du signal d'entrée est de 0.1 bit/symbole alors que le codage de Huffman ne pourra pas descendre en dessous de 1.

Dans le cas de l'exemple de la table II.1.1, l'entropie est égale à :

$$-\left(\frac{10}{16} \times \log_2\left(\frac{10}{16}\right) + \frac{3}{16} \times \log_2\left(\frac{3}{16}\right) + \frac{3}{16} \times \log_2\left(\frac{3}{16}\right)\right) \approx 1.33 \text{ bits/symbole}$$

Avec notre codage de Huffman, on obtient $10\times1+3\times2+3\times2 = 22$ bits soit 1.375 bits/symbole ce qui vérifie bien l'inégalité II.1.2.

De plus, le codage de Huffman de base ne prend pas en compte le cas où les probabilités d'occurrence des symboles évoluent au cours du temps. Une version adaptative de celui-ci a été initialement développée par Faller (1973) et Gallager (1978) (puis améliorée par Knuth (1985) pour aboutir à l'algorithme FGK (Faller-Gallager-Knuth)) : à l'état initial, l'algorithme considère chaque symbole de l'alphabet source comme équiprobable puis, au fur et mesure du déroulement de l'algorithme, met à jour dynamiquement les probabilités d'apparition (ainsi que l'arbre de Huffman associé). Cette méthode permet ainsi de s'affranchir de la première passe de récupération des statistiques et de pouvoir coder en temps réel une source d'information.

Le codage de Huffman est encore utilisé de nos jours par le standard JPEG, notamment pour le codage entropique des coefficients issus de la DCT (voir section II.1.2.1.1). Celui-ci sera également évoqué dans la section II.2.2.2, dans laquelle nous utiliserons une approche relativement similaire pour la génération des codes binaires de notre algorithme MICA, et dans la section III.3.1.1.5 où nous adapterons les tables de Huffman afin de pouvoir prendre en charge les coefficients issus de la DCT-3D de notre algorithme de compression.

II.1.1.1.3 Le codage arithmétique

Le concept de codage arithmétique a été évoqué par Peter Elias au début des années 60, formalisé en 1963 par Abramson (1963) et repris par Witten *et al.* (1987), Moffat *et al.* (1998). L'idée principale du codage arithmétique est de pouvoir franchir la limite du 1 bit/symbole minimum et ainsi se rapprocher de manière quasi-idéale de l'entropie d'un signal à compresser. Pour cela, le codeur arithmétique ne va pas utiliser les statistiques des symboles de l'alphabet de départ, mais plutôt générer une information globale (contenant l'intégralité du message à coder) et encoder cette information.

L'algorithme repose sur le calcul de la fonction de répartition des probabilités d'apparition de chaque symbole de l'alphabet. La version de base de l'algorithme requiert donc, comme celui d'Huffman, une première passe pour calculer les statistiques. On part de l'intervalle $[0, 1[$ et on le subdivise conformément à la fonction de répartition de chaque symbole, puis on définit le sous-intervalle associé à ce symbole comme intervalle courant. On réitère cette opération tant que le message à coder contient de l'information. Une fois tous les symboles lus, il suffira d'encoder l'étiquette du dernier intervalle courant (c'est à dire son point milieu). Imaginons tout d'abord que l'on travaille sur un alphabet $\Omega = \{a_1, a_2, ..., a_n\}$ de probabilités d'apparitions $p_1, p_2, ..., p_n$ et que l'on veuille coder un message de longueur 1 contenant le symbole a_i. La fonction de répartition pour le symbole a_i, notée $F(a_i)$ est définie par :

$$F(a_i) = \begin{cases} 0.0 & \text{si } i < 1 \\ \sum_{j=1}^{i} p_j & \text{si } 1 \leq i < n \\ 1.0 & \text{si } n \leq i \end{cases} \tag{II.1.3}$$

On part donc de l'intervalle $I_0 = [L_0, U_0[$ ($[0, 1[$ à l'initialisation), qu'il faut maintenant découper selon les probabilités cumulées de chaque symbole. Dans ce cas, on va donc découper I_0 en n sous-intervalles SI_i définis par leur étiquette ϵ_i conformément à la formule II.1.4.

$$\forall i \in [1, n], SI_i = [F(a_{i-1}), F(a_i)[, \epsilon_i = F(a_{i-1}) + \frac{1}{2} \times p_i \tag{II.1.4}$$

Le message à coder en binaire sera donc ϵ_i. Dans le cas d'un message plus long (de longueur m), il faudra de la même manière partir de I_0 et à chaque lecture d'un symbole a_i, subdiviser celui-ci selon les probabilités cumulées et considérer le sous-intervalle SI_i associé au k$^{\text{ième}}$ symbole lu (a_i) comme intervalle courant ($I_k = SI_i$) et ainsi réitérer l'opération pour chaque nouveau symbole. Il faut donc à chaque lecture de symbole calculer les bornes inférieures et supérieures du nouvel intervalle courant à partir de celles du précédent ainsi que des statistiques d'apparition du symbole venant d'être lu. A la lecture du k$^{\text{ième}}$ symbole (a_i) de la chaine à coder, on a :

$$\begin{cases} L_k = L_{k-1} + (U_{k-1} - L_{k-1}) \times F(a_{i-1}) \\ U_k = L_{k-1} + (U_{k-1} - L_{k-1}) \times F(a_i) \end{cases} \tag{II.1.5}$$

Le but de l'algorithme va être à chaque lecture de symbole de considérer l'intervalle associé à ce symbole comme intervalle courant et de le re-subdiviser proportionnellement aux probabilités cumulées. Lorsque tous les symboles sont lus, il suffit alors de coder l'étiquette du dernier intervalle courant. Prenons un exemple simple : considérons la première ligne de l'exemple de la table II.1.1, le signal à coder sera donc :

156 148 148 164 148

TABLEAU II.1.4 – Exemple de signal 5x1 issu de l'image Lena

Dans ce cas, l'alphabet $\Omega = \{a_1, a_2, a_3\} = \{148, 156, 164\}$ et les statistiques d'apparition correspondantes sont répertoriées dans le tableau II.1.5.

Symbole	Probabilités	F()
148	0.6 (3/5)	0.6
156	0.2 (1/5)	0.8
164	0.2 (1/5)	1.0

TABLEAU II.1.5 – Statistiques des symboles de l'exemple II.1.4

On définit $I_0 = [L_0, U_0[$ comme intervalle de départ. La table II.1.6 ainsi que la figure II.1.2 illustre l'évolution des valeurs de L et U à chaque lecture de symbole.

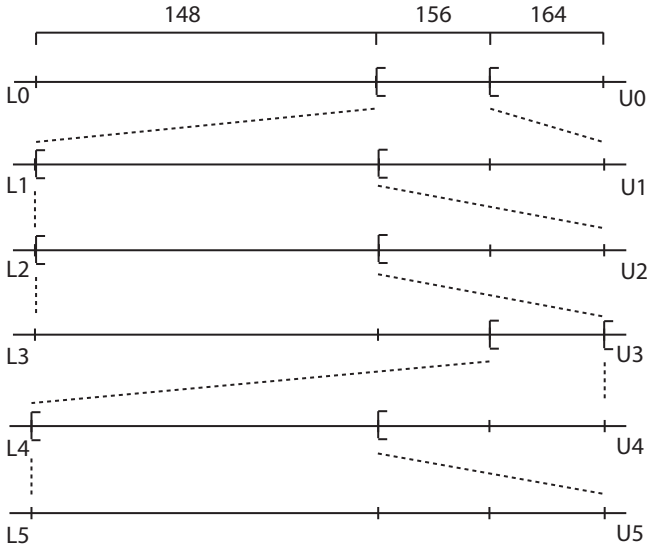


FIGURE II.1.2 – Subdivisions successives de l'interval I

L'information à coder est donc l'étiquette du dernier intervalle courant soit 0.5546875. L'implémentation détaillée ci-dessus n'est pas réalisable en pratique. Il est facile de se rendre compte

Etape	Symbole lu	L	U	Etiquette associée
Initialisation	∅	0.0	1.0	0.5
1	156	0.6	0.8	0.7
2	148	0.6	0.72	0.66
3	148	0.6	0.672	0.636
4	164	0.6576	0.672	0.6648
5	148	0.6576	0.66624	0.66192

TABLEAU II.1.6 – Evolution des bornes de l'intervalle I dans le temps.

que pour un message relativement long (texte, image, ...) l'étiquette correspondant au dernier intervalle généré par l'algorithme aura un nombre très grand de chiffres derrière la virgule. En pratique, les ordinateurs ne disposent pas d'une précision aussi importante et il faut donc modifier l'algorithme afin de pallier à ce problème. Pour cela, on va plutôt travailler sur des nombres entiers pour représenter les bornes inférieures et supérieures de l'intervalle courant (généralement représentées sur 16 bits). Le problème va donc être de représenter l'intervalle $[0, 1[$ par des entiers. Pour cela, on va choisir de représenter uniquement la partie flottante du nombre à virgule. Pour tout intervalle $[0.l, 0.u[$ inclus dans $[0.0, 1.0[$, L va être représenté par le nombre l et U par $u - 1$ (-1 pour simuler l'intervalle ouvert). La seule modification de l'algorithme est qu'à chaque fois que le premier digit de L et U est commun, on envoie alors celui-ci au codeur, on décale vers la gauche les chiffres de L et U puis on complète avec un 0 à droite pour L et un 9 pour U. Ces modifications sont aussi répercutées sur les intervalles réels où le premier chiffre après la virgule est tout simplement supprimé. L'exécution de l'algorithme sur l'exemple précédent est présentée dans le tableau II.1.7. On retrouve bien le message 66192 correspondant à l'étiquette 0.66192 calculée sur l'exemple précédent.

Etape	Symbole lu	L (réel)	U (réel)	L (entier)	U (entier)	nombre à coder
Initialisation	∅	0.0	1.0	00000	99999	∅
1	156	0.6	0.8	60000	79999	∅
2	148	0.6	0.72	60000	71999	∅
3	148	0.6	0.672	60000	67199	6
4	164	0.6576	0.672	57600	71999	∅
5	148	0.6576	0.66624	57600	66239	6192

TABLEAU II.1.7 – Evolution des bornes de l'intervalle I dans le temps et correspondance avec les intervalles entiers.

Nous reparlerons du codage arithmétique, notamment dans la section II.1.3.3.4 consacrée à l'algorithme CABAC du standard H.264/AVC. Le codage arithmétique sera aussi évoqué en section III.3.1.2, où celui-ci sera utilisé pour l'étape de codage entropique de notre algorithme de compression d'images multi-vues basé sur les ondelettes.

Les différents algorithmes présentés ci-dessus sont dédiés au codage entropique d'une source, ils sont utilisés dans des schémas de compression en fin de traitement mais ne constituent pas une méthode de compression à proprement parler. Pour la compression sans perte d'images 2D, on distingue généralement deux familles distinctes d'algorithmes : ceux basés sur les dictionnaires et ceux basés sur la modélisation de contexte. Ces deux types d'approches font l'objet des deux prochaines sections.

II.1.1.2 Méthodes basées sur les dictionnaires

En 1977, Ziv et Lempel (1977) présentent une nouvelle méthode de compression sans perte : l'algorithme LZ77 (aussi appelée LZ1). Celui-ci introduit l'utilisation d'une fenêtre glissante (*sliding window*) composée de deux buffers (voir Fig. II.1.3).

FIGURE II.1.3 – La fenêtre glissante de l'algorithme LZ77

Le premier buffer, à gauche sur la figure II.1.3 ("*look-ahead buffer*"), va servir de dictionnaire et permettre de retenir une séquence de symboles (de taille finie) précédemment encodée. La partie droite de la fenêtre ("*search-for buffer*"), contient quant à elle la séquence de symboles à encoder. La taille du buffer dictionnaire est généralement très grande (afin de retenir le maximum de symboles) par rapport à celle du buffer de recherche. Le but de l'algorithme va consister à déplacer cette fenêtre le long du signal d'entrée. Pour chaque nouvelle séquence de symboles dans le buffer de recherche, on regarde si celle-ci a déjà été précédemment encodée en parcourant le buffer dictionnaire de droite à gauche. Le système de compression va alors générer des symboles du type (*offset*, *length*, *nextToCode*) appelés "token", qui signifient : "recopie *length* symboles depuis la position *offset* (prise depuis la fin du buffer dictionnaire) et ajoute le symbole *nextToCode*". La fenêtre est ensuite décalée de *length* + 1 symboles vers la droite. Les différents champs *offset*, *length* et *nextToCode* de chaque token sont ensuite codés en binaire.

La figure II.1.4 illustre le fonctionnement de l'algorithme sur un exemple simple. Le premier symbole à être lu est 'a', le tampon dictionnaire étant vide, le token (0,0,a) est donc généré et la fenêtre se décale d'un cran vers la droite. Le deuxième symbole est encore un 'a' suivi d'un 'b', or un 'a' est déjà dans le tampon dictionnaire. Dans ce cas on génère le token (1,1,'b') pour signifier au décodeur qu'il faudra écrire la chaîne de symboles située à 1 case de la tête de lecture sur une longueur de 1 suivi d'un 'b'. La fenêtre est ensuite décalée de 2 cases. Le symbole 'b' est lu, la chaîne 'b' est déjà présente dans le dictionnaire. On lit alors un autre caractère, ici 'b' or 'bb' n'est pas dans le dictionnaire. On génère donc le token (1,1,'b') et la fenêtre est décalée. Ensuite un 'a' est lu, or on dispose déjà d'un 'a' dans le tampon dictionnaire. On lit alors des symboles jusqu'à ce que le symbole lu diffère du symbole associé dans la chaîne du buffer dictionnaire. Dans notre cas, la séquence 'aab' est commune mais pas la séquence 'aaba'. On envoie le token (5,3,a) et ainsi de suite.

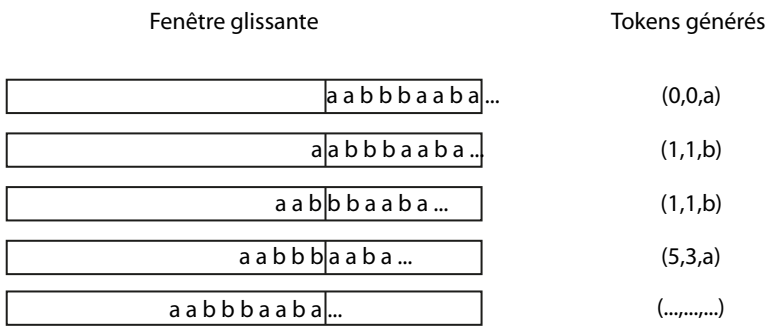


FIGURE II.1.4 – Trace d'exécution de l'algorithme LZ77

Il se peut que dans certains cas, on rencontre deux fois la même chaîne dans le buffer dictionnaire créant ainsi une ambiguïté sur la décision concernant la séquence à associer. Dans ce cas, l'algorithme LZ77 privilégie la première séquence trouvée (de la droite vers la gauche). Ce type de codage est tout particulièrement efficace lorsque le signal d'entrée contient beaucoup de répétitions.

De nombreuses variantes et améliorations ont été apportées à cet algorithme. Les plus répandues sont :

- LZ78 Ziv et Lempel (1978) qui utilise non plus une fenêtre glissante mais un dictionnaire à part entière. Soit n le nombre de symboles différents de l'alphabet constituant le signal, le dictionnaire est représenté sous la forme d'un arbre n-aire où chaque nœud (étiqueté par une adresse) correspond à une séquence précise de symboles (ou à un symbole unique). L'intérêt majeur de cet algorithme est de construire des tokens contenant uniquement deux champs *address* et *nextToCode*, évitant ainsi d'avoir à transmettre le champ *length* du LZ77. En effet, en utilisant le champ *address* il est possible de reconstituer la séquence de symboles en redescendant dans l'arbre et il suffit alors de concaténer le dernier symbole *nextToCode* pour obtenir la séquence originale. De plus, la structure d'arbre améliore largement le temps de recherche des séquences de symboles.

- LZW Welch (1984) qui permet de se passer du champ *nextToCode* des tokens de l'algorithme LZ78. Le dictionnaire est maintenant initialisé avec les symboles constituant l'alphabet de base, puis chaque symbole du signal d'entrée est lu et accumulé tant que la chaîne I est présente dans le dictionnaire. Lorsque la chaîne I concaténée avec le dernier caractère lu x n'appartient pas au dictionnaire, on envoie alors le token contenant l'adresse du nœud correspondant à I, on ajoute la chaîne Ix au dictionnaire et on recommence le processus en prenant comme chaîne de départ x. Généralement, on utilise des champs *address* 16 bits permettant ainsi une taille de dictionnaire de 64 Ko.

- Deflate Katz (1996) qui combine l'algorithme LZ77 (fenêtre glissante) avec le codage de Huffman. Cet algorithme génère trois types d'informations : des littéraux (symbole unique non présent dans le dictionnaire), des longueurs (correspondant au champ *length* du LZ77) ainsi que des distances (qui correspondent au champ *offset* de LZ77). Encoder les littéraux séparément va permettre à l'algorithme de s'affranchir du troisième champ *nextToCode* du LZ77 classique. De plus, des tables de Huffman (codes à longueur variable) sont utilisées pour encoder les informations de longueur et de distance. Enfin, la recherche des séquences de symboles dans le buffer dictionnaire de la fenêtre glissante est effectuée en utilisant une table de hachage rendant le processus plus efficace. Cet algorithme est encore actuellement utilisé dans de nombreuses applications telles que le protocole HTTP, le format de fichier PNG ou bien même le format PDF d'Adobe.

Deux formats d'images bien connus utilisent la compression basée sur les dictionnaires : le format GIF et le format PNG. Ces deux formats, qui seront utilisés pour les tests réalisés en section III.3.2 pour la compression sans perte des cartes de disparité, sont succinctement décrits ci-dessous.

II.1.1.2.1 Le format GIF (Graphics Interchange Format)

Le format GIF (de son vrai nom GIF 87a) a été développé en 1987, puis repris en 1989 pour sa version 89a, par la firme CompuServe afin de permettre le téléchargement d'images via internet (voir Blackstock (1987)). GIF est un format de compression 8 bits, il utilise une palette à 256 niveaux et permet ainsi de compresser les images allant du monochrome (1bit/pixel) aux images niveaux de gris ou palettisées (8bits/pixel). Ce format utilise un dictionnaire dynamique qui s'initialise à $2^n - 1$ entrées (où n désigne le nombre de plans de bits de l'image à compresser) et peut voir sa taille doubler lorsqu'il est plein. Lorsque sa taille atteint 4096 entrées, l'encodeur décide de garder le dictionnaire en l'état ou d'en démarrer un nouveau. Chaque ligne est scannée une à une et les différents symboles sont encodés selon l'algorithme LZW. Du fait du parcours de l'image, le format GIF peut ainsi déceler des redondances au sein d'une même ligne mais pas entre deux lignes. Ainsi il n'exploite pas les redondances 2D au sein d'une image et produit de faibles taux de compression. Le format PNG décrit ci-dessous permet de remédier à ce défaut.

II.1.1.2.2 Le format PNG (Portable Network Graphics)

Le format PNG a été mis au point en 1995 par le groupe de développement PNG [1]. Ce format est destiné à remplacer le format GIF qui posait des problèmes de royalties dus à l'algorithme LZW breveté utilisé, mais aussi afin de proposer un nouvel algorithme permettant d'obtenir de meilleurs résultats en terme de compression. Parmi les différentes caractéristiques du format PNG, on compte : la compression d'images couleurs où chaque composante peut comprendre jusqu'à 16 plans de bits, la gestion de la transparence via un canal alpha, la compression sans perte basée sur l'algorithme Deflate (Katz (1996)), disponible sur http ://www.ietf.org/rfc/rfc1951) couplée à un système de prédiction et la gestion de l'entrelacement (pour un affichage progressif).

La compression se fait en deux étapes distinctes : une étape de filtrage ("delta filtering") puis l'étape de compression à proprement parler grâce à l'algorithme Deflate. L'étape de filtrage, qui s'effectue composante par composante et ligne par ligne, permet de prédire la valeur du pixel courant $P_{i,j}$ selon cinq méthodes utilisant le voisinage antérieur de $P_{i,j}$. La valeur prédite est ensuite soustraite de $P_{i,j}$ pour donner $D_{i,j}$. La méthode de prédiction sélectionnée est celle qui minimise la somme des différences $D_{i,j}$ pour toute la ligne en cours de codage. La ligne transformée est ensuite envoyée au codeur Deflate pour compression et l'information contenant le type de prédiction est aussi fournie afin de permettre la décompression. Pour plus de précisions sur le format PNG, le lecteur peut se référer à la spécification disponible sur http ://www.libpng.org/pub/png/spec/iso/.

II.1.1.3 Méthodes basées sur les contextes

Cette section présente les différentes méthodes de compression d'images basées sur la modélisation de contextes. Ces méthodes se basent sur l'observation suivante : un pixel quelconque

1. http ://www.libpng.org/pub/png/

d'une image a de fortes chances d'avoir une valeur proche de celle des pixels voisins. Ces approches sont donc fortement adaptatives, elles exploitent les propriétés locales de l'image plutôt que l'ensemble du signal. Comme la plupart des méthodes adaptatives, le codeur et le décodeur fonctionnent de manière identique. L'image est parcourue (parcours raster-scan) et pour chaque symbole à coder, on considère son voisinage antérieur (symboles ayant déjà été encodés et appelés ici template) afin de déterminer un contexte. Chaque contexte possède son propre modèle statistique et permet ainsi de générer des codes binaires de longueur optimale. A noter que dans le cas de CALIC (voir section II.1.1.3.2), on considère aussi le voisinage postérieur grâce à un algorithme multi-passes.

Comme on peut le voir sur la figure II.1.5 illustrant le fonctionnement général de ces méthodes, elles ont la particularité de séparer complètement la partie modélisation de la partie codage permettant ainsi d'utiliser différents algorithmes de codage entropique, qu'ils soient statiques, adaptatifs, arithmétiques, etc ...
L'encodage (et le décodage) d'un pixel se fait généralement de la façon suivante :

1) Tout d'abord, on génère une valeur de prédiction I' pour le pixel courant à l'aide du template associé à celui-ci et d'une fonction déterministe (spécifique à l'algorithme utilisé). Cette partie constitue la partie statique du système de prédiction.
2) Le modélisateur de contextes va ensuite associer un contexte au template courant. Chaque contexte possède ses propres statistiques d'apparition et va ainsi permettre de générer une valeur de correction à appliquer à I' (partie adaptative du prédicteur).
3) L'erreur de prédiction ϵ est ensuite calculée puis renvoyée au modélisateur de contextes. Celle-ci va servir à mettre à jour les différentes tables statistiques spécifiques au contexte courant, puis est ensuite transmise au codeur entropique, éventuellement accompagnée de paramètres de codage (si celui-ci est dynamique).

L'intérêt majeur de ces méthodes est qu'elles conservent toutes les statistiques associées aux différents contextes et permettent ainsi via la procédure de correction adaptative, de minimiser les erreurs de prédiction en s'ajustant au mieux au modèle statistique. Elles offrent généralement les meilleures performances de compression pour la majorité des images. Dans la suite de cette section nous détaillerons brièvement deux méthodes basées sur la modélisation de contexte : FELICS et CALIC. Le format JPEG-LS, lui aussi basé sur la modélisation de contexte sera, présenté dans la section II.3.1 où nous décrirons notre adaptation de ce dernier au cas multi-vues.

II.1.1.3.1 FELICS : Fast Efficient and Lossless Image Compression System

L'algorithme FELICS, développé par Howard et Vitter (1993), a été conçu pour la compression d'images niveaux de gris et utilise deux pixels voisins du pixel courant pour encoder celui-ci à l'aide de codes à longueurs variables. Soit le pixel courant X, A le pixel à gauche de X et B le pixel au dessus de X, on définit alors l'intervalle $[L, H]$ où $L = min(A, B)$ et $H = max(A, B)$ ainsi que le contexte $\triangle$ tel que $\triangle = H - L$. Pour chaque contexte $\triangle$, l'intensité lumineuse de X suit généralement la distribution décrite par la figure II.1.6.

Afin de suivre au mieux la loi ρ, on va alors définir trois zones :

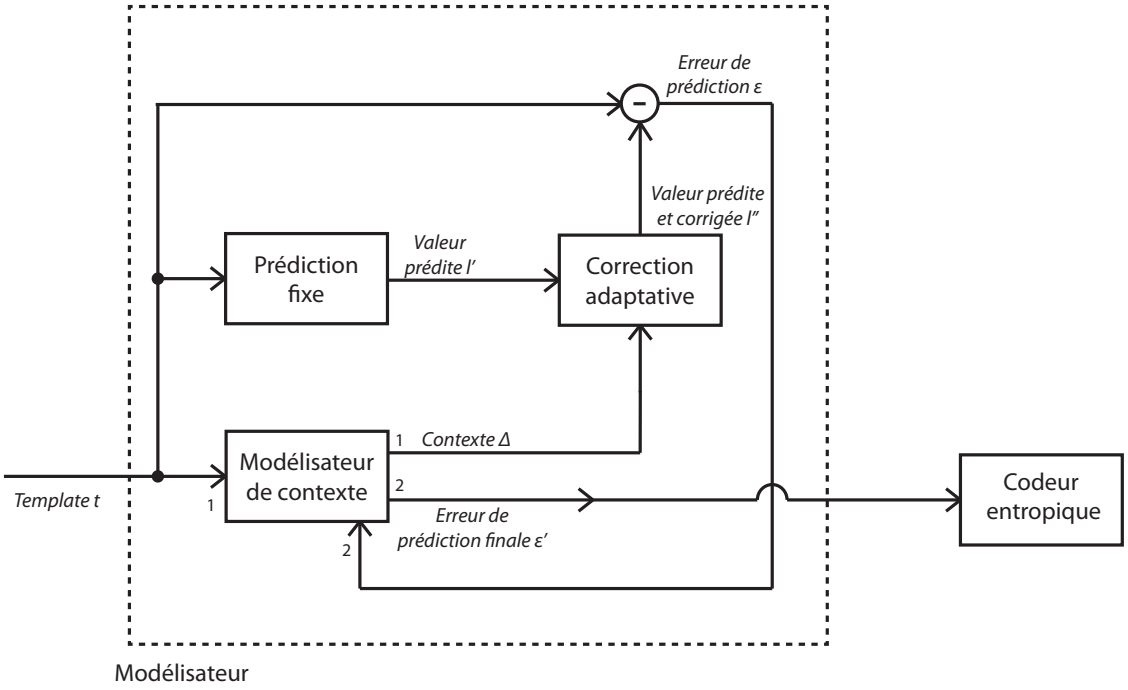


FIGURE II.1.5 – Schéma de fonctionnement général des méthodes basées sur la modélisation de contextes

- la zone 1 définie par l'intervalle $[0, L[$
- la zone 2 définie par l'intervalle $[L, H]$
- la zone 3 définie par l'intervalle $[H, 2^n - 1]$ où $2^n - 1$ est la valeur maximale possible du pixel.

La longueur du code binaire pour coder X va donc dépendre de la zone où se situe celui-ci : s'il appartient à la zone 1 ou 3, le code binaire correspondant devra être plus long que si X appartenait à la zone 2. De plus, plus on s'éloigne de L (resp. H) si X est compris dans la zone 1 (resp. 3) plus le code devra être long. Si X est compris dans $[L, H]$, le code binaire devra être légèrement plus long (ou dans certains cas de même longueur) lorsque X est situé près des bornes (cette région étant relativement uniforme). Cette idée sera reprise lors de l'élaboration de l'algorithme MICA (II.2.2.2).

Si X est dans la zone 2, on doit alors encoder la différence $X - L$ appartenant à l'intervalle $[0, \triangle]$ qui contient $\triangle + 1$ valeurs différentes. Dans le cas où $\triangle + 1$ est une puissance de deux, alors la valeur $X - L$ est tout simplement encodée en binaire en utilisant $\log_2(\triangle + 1)$ bits. Dans le cas contraire, on choisit alors de produire des codes binaires de longueur $\lfloor \log_2(\triangle + 1) \rfloor$ pour la zone centrale et de longueur $\lceil \log_2(\triangle + 1) \rceil$ près des bornes (où $\lfloor . \rfloor$ représente l'arrondi par défaut et $\lceil . \rceil$ représente l'arrondi par excès). Pour cela, on calcule les deux nombres a et b permettant de connaître la répartition de ces deux types de codes en appliquant la formule II.1.6.

$$\begin{aligned} a &= 2^{k+1} - (\triangle + 1) \\ b &= 2(\triangle + 1 - 2^k) \end{aligned} \quad \text{où } k = \lfloor \log_2(\triangle + 1) \rfloor \tag{II.1.6}$$

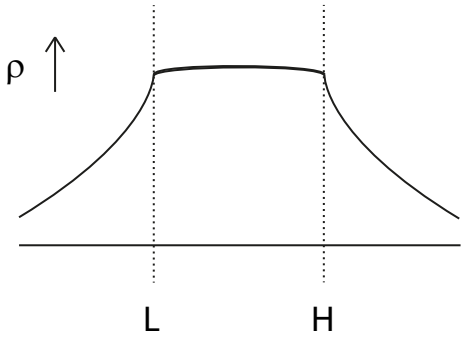


FIGURE II.1.6 – Loi de probabilité ρ de la luminosité du pixel X

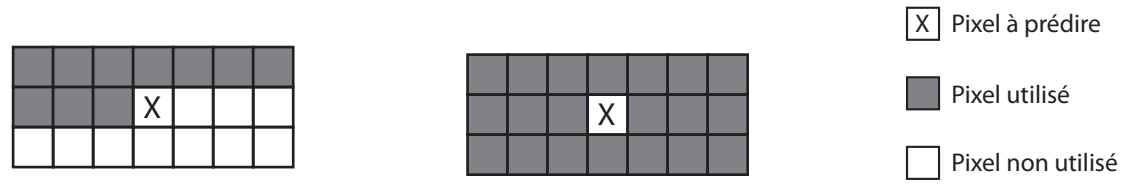
Les $\triangle + 1$ valeurs de l'intervalle $[0, \triangle]$ vont donc être codées en utilisant a codes binaires de longueur k et b codes de longueur $k + 1$ (on peut d'ailleurs remarquer que $a + b = \triangle + 1$). On commence alors à remplir l'intervalle avec des codes à k bits en commençant au milieu et en s'éloignant petit à petit. Une fois que les a codes ont été attribués, il suffit alors de continuer avec les b codes restants. Il suffit enfin d'envoyer le code binaire associé à la valeur $X - L$.

Si X se situe dans la zone 1 (resp. 3), on va alors coder la différence $L - X$ (resp. $X - H$) de telle sorte que plus la différence est élevée plus le code résultant devra être long. L'algorithme FELICS utilise pour cela le codage de Rice-Golomb (Golomb (1966), Rice (1979)). Le paramètre k_m (spécifique au codage de Rice-Golomb) est déterminé de la façon suivante : pour chaque contexte $\triangle$ et pour chaque valeur de k_m (généralement comprise entre 1 et 4 mais des valeurs plus élevées restent possible), on cumule la longueur binaire totale des codes si l'on avait encodé toutes les différences de ce contexte à l'aide du paramètre k_m. Il suffit alors de choisir le paramètre qui minimise cette quantité.

II.1.1.3.2 CALIC : Context-based Adaptative Lossless Image Compression

L'algorithme CALIC, développé par Wu (1995, 1996) est une méthode de compression basée sur la modélisation de contextes, qui se démarque des autres approches. La majorité des méthodes basées sur les contextes utilisent un encodeur et un décodeur suivant exactement la même procédure algorithmique. Ceci est possible car ce type d'algorithme utilise des pixels déjà traités pour la phase de modélisation de contexte du pixel courant. Ainsi l'encodeur et le décodeur ont accès aux mêmes informations lors du traitement d'un pixel i et on parle dans ce cas d'une modélisation de contexte asymétrique (ou à 180°, voir Fig. II.1.7(a)). Dans l'idéal, afin de modéliser au mieux le contexte du pixel courant, il faudrait utiliser tout le voisinage à 360° de ce pixel comme l'illustre la Figure II.1.7(b). Dans la réalité, ceci n'est pas possible car le décodeur n'a pas accès à l'information contenue dans les pixels suivants. Il lui est ainsi impossible de modéliser le contexte pour le pixel courant.

L'idée majeure de l'algorithme CALIC va donc être d'exploiter ce voisinage 360° tout en permettant au décodeur de disposer d'une information sur la valeur des pixels à venir. Pour cela, l'algorithme va se dérouler en trois phases indépendantes où chacune va permettre d'encoder une



(a) Contexte asymétrique (b) Contexte symétrique

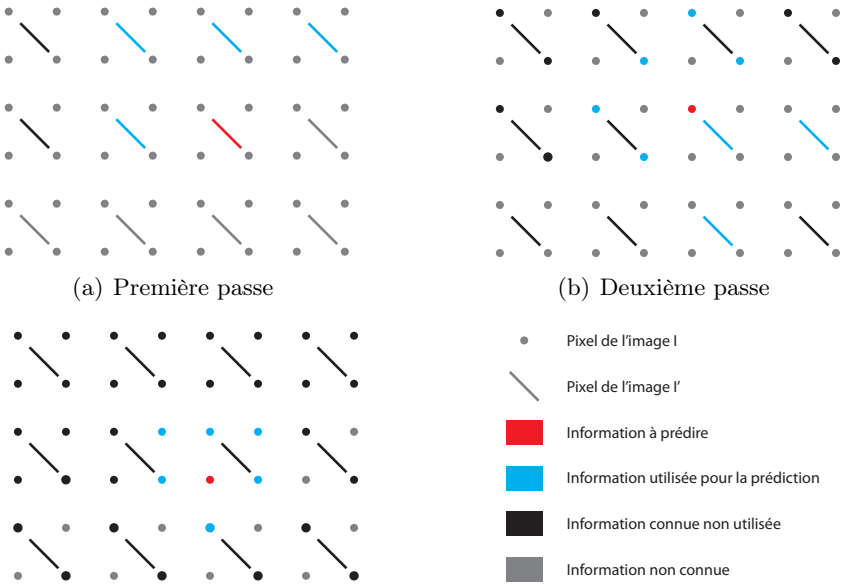
FIGURE II.1.7 – Modélisation de contexte asymétrique (180°) et symétrique (360°)

partie de l'information. Les étapes de modélisation de contextes et de codage étant communes au trois phases, nous détaillerons uniquement pour chacune d'elles la partie prédiction.

– L'objectif de la première phase va être de sous-échantillonner l'image sur des couples de pixels diagonaux afin de disposer, lors de la seconde phase, d'un template à 360° autour du pixel à prédire. Pour cela, on considère l'image I (de taille $H \times W$ pixels) et on génère à partir de celle-ci une nouvelle image I' de taille $(H/2) \times (W/2)$ en appliquant la formule II.1.7.

$$I'[i,j] = (I[2i,2j] + I[2i+1,2j+1])/2 \text{ pour } 0 \leq i < H/2, 0 \leq j < W/2 \qquad \text{(II.1.7)}$$

Les pixels constituant la nouvelle image I' sont ensuite parcourus dans l'ordre raster-scan et chacun est prédit en utilisant un template basé sur quatre voisins conformément à la figure II.1.8(a) et à l'équation II.1.8.



(a) Première passe (b) Deuxième passe

(c) Troisième passe

FIGURE II.1.8 – Les trois passes de l'algorithme CALIC.

$$x_{pred} = \frac{1}{2}I'[i-1,j] + \frac{1}{2}I'[i,j-1] + \frac{1}{4}I'[i-1,j+1] - \frac{1}{4}I'[i-1,j-1] \qquad \text{(II.1.8)}$$

– La deuxième passe va maintenant encoder les pixels de l'image I ayant servis à générer l'image I'. De manière identique, l'image I est parcourue en raster-scan et chaque pixel impliqué dans l'équation II.1.7 est prédit à partir d'un template de 8 voisins conformément à la figure II.1.8(b). Ces 8 voisins sont constitués de pixels précédemment encodés lors de la seconde passe, mais grâce à la première passe qui est d'ores et déjà terminée, on a aussi accès aux valeurs moyennes de l'image I'. Cela va nous permettre de réaliser une prédiction à 360° autour du pixel courant. La valeur de prédiction pour ce pixel est calculée grâce à l'équation II.1.9.

$$\begin{aligned} x_{pred} &= 0.9I'[i,j] + \frac{1}{6}(I[2i+1,2j-1] + I[2i-1,2j-1] + I[2i-1,2j+1]) \\ &\quad -0.05(I[2i,2j-2] + I[2i-2,2j]) - 0.15(I'[i,j+1] + I'[i+1,j]) \end{aligned} \quad \text{(II.1.9)}$$

– Enfin, la troisième phase va permettre de coder les pixels restants. Pour cette phase on dispose maintenant d'un voisinage direct (valeur de pixels et non de moyennes) à 360° pour la prédiction comme l'illustre la figure II.1.8(c). La valeur prédite est calculée à partir de ses 6 voisins directs, en utilisant l'équation suivante (II.1.10) :

$$x_{pred} = \frac{3}{8}(I[i,j-1]+I[i-1,j]+I[i,j+1]+I[i+1,j]) - \frac{1}{4}(I[i-1,j-1]+I[i-1,j+1]) \quad \text{(II.1.10)}$$

Pour chaque passe et pour chaque pixel, une fois la valeur de prédiction calculée, il est possible de déterminer le contexte courant. CALIC modélise le contexte à partir de deux paramètres d et t. Le premier paramètre d est déduit en calculant une quantité Δ appelée "error strength" à partir du template x_1, ..., x_K (où x_1, ..., x_K désignent l'ensemble des pixels utilisés pour la prédiction) et défini par :

$$\Delta = \sum_{k=1}^{K} w_k |x_k - x_{pred}|$$

où les coefficients w_k sont constants et fixés par le standard. Δ est ensuite quantifiée afin d'obtenir d de sorte à minimiser l'entropie globale des différentes classes d'erreurs. De la même façon, les valeurs de quantification sont déterminées au préalable. De manière parallèle, à partir du template courant, on détermine un contexte de texture t défini par un mot binaire de K bits afin d'obtenir une meilleure modélisation et d'améliorer les performances en terme de ratio de compression. Ce mot binaire est généré de la façon suivante :

$$t_k = \begin{cases} 0 \text{ si } x_k \geq x_{pred} \\ 1 \text{ sinon} \end{cases} \quad \text{(II.1.11)}$$

Une fois les paramètres d et t définis, on peut alors déterminer le contexte courant et ainsi calculer la valeur de correction à appliquer à x_{pred}. Une fois cette valeur de correction appliquée, il est possible de calculer l'erreur de prédiction ϵ ainsi que les différents paramètres de codage et de passer ceux-ci au codeur. Comme précisé dans la section II.1.1.3, le codage entropique étant indépendant de la partie modélisation de contexte, on ne détaillera pas la partie codage qui reste à l'appréciation de l'utilisateur.

II.1.2 Méthodes avec pertes

Nous abordons maintenant dans cette section les principales méthodes de compression avec pertes généralement utilisées (le signal reconstruit contient des distorsions par rapport au signal original). Afin que ces distorsions soient le moins visibles possibles, les algorithmes de compression d'images avec pertes doivent prendre en compte les différentes propriétés du système de vision humain et exploiter à leur avantage ses faiblesses. Concernant le système de vision humain, on accepte de manière générale les deux assertions suivantes :

(1) L'œil humain est plus sensible à la luminance qu'à la chrominance.
(2) L'œil humain est moins sensible aux détails (hautes fréquences) d'une image qu'à son allure générale.

Une image couleur standard est quant à elle définie par une double redondance :

(3) Une redondance colorimétrique entre chaque composante.
(4) Une redondance spatiale 2D pour des pixels appartenant à une même zone de l'image.

Ces deux types de redondances devront être exploités au maximum afin de dé-corréler au mieux l'information et proposer des ratios de compression les plus élevés possibles. Pour cela, on utilise des algorithmes basés sur les transformations vers le domaine fréquentiel. Tous ces algorithmes suivent un schéma de fonctionnement plus ou moins commun décrit par la figure II.1.9.

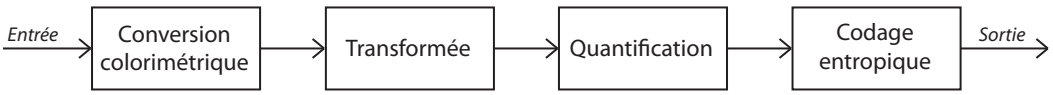


FIGURE II.1.9 – Pipeline de fonctionnement des méthodes basées sur les transformations.

Chacune de ces différentes étapes joue un rôle précis dans le processus de compression :

– L'étape de conversion colorimétrique va tout d'abord permettre de réduire la corrélation entre chaque composante couleur de l'image (la propriété (3) est ainsi exploitée). De plus, si le système colorimétrique est un modèle basé luminance/chrominance, il est possible de réaliser un sous-échantillonnage de la chrominance conformément à l'observation (1) afin de réduire de manière significative le volume de données à compresser sans pour autant altérer visuellement le signal. D'autres méthodes plus récentes (Gershikov *et al.* (2007), Santhi et Banu (2011)), consistent à corréler au maximum l'information entre chaque composante couleur, afin de n'en garder qu'une et d'approximer les deux autres par transformation affine sur la première.

– La transformée va permettre d'étudier l'image dans le domaine fréquentiel. Ainsi, on va pouvoir séparer les basses fréquences (allure générale de l'image) des hautes (détails de l'image) afin d'exploiter l'observation (2) durant la phase de quantification. Deux approches sont possibles concernant la transformée, soit elle est appliquée localement en divisant l'image en blocs de $n \times n$ pixels comme c'est le cas pour la DCT (Discrete Cosine Transform) utilisée dans l'algorithme JPEG, soit elle est appliquée de manière globale comme pour la

DWT (Discrete Wavelet Transform) de l'algorithme JPEG 2000.

– La quantification est l'étape cruciale des méthodes de compression avec pertes. C'est durant celle-ci que l'on choisit de négliger une partie de l'information afin de réduire l'entropie, entrainant ainsi en une distorsion du signal d'entrée. Dans le domaine fréquentiel, cela va nous permettre de privilégier les basses fréquences sur les hautes et ainsi d'exploiter la propriété (2).

– Enfin, la phase de codage entropique va permettre de coder les coefficients quantifiés issus de la précédente étape afin de produire un flux binaire. On utilise pour cela des algorithmes de compression sans perte (détaillés dans la première section du chapitre) tels que le codage arithmétique ou bien le codage de Huffman.

Dans les deux sections suivantes, nous allons décrire plus en détail le processus de transformation que ce soit pour la DCT ou bien la DWT. Pour chacune de ces deux transformations, on détaillera ensuite les méthodes de compression avec pertes issues de celles-ci.

II.1.2.1 Méthodes basées DCT

La DCT (Discrete Cosine Transform), introduite par Ahmed *et al.* (1974), est une des transformations les plus utilisées aussi bien pour la compression que pour d'autres applications. Elle permet de transformer un signal d'entrée (pixels, son, ...) constitué de n valeurs en une série de n coefficients représentant les différentes fréquences composant le signal : un coefficient, appelé coefficient DC, correspondant à la fréquence nulle et caractéristique de l'allure globale du signal et $n-1$ coefficients appelés coefficients AC représentant les hautes fréquences (détails de l'image). La DCT 1D sur un signal d'entrée f de taille N est donnée par la formule II.1.12 :

$$F(k) = \sqrt{\frac{2}{N}} C_k \sum_{n=0}^{N-1} f(n) \cos\left[\frac{(2n+1)k\pi}{2N}\right] \tag{II.1.12}$$

où

$$C_k = \begin{cases} \frac{1}{\sqrt{2}}, & \text{si } k = 0, \\ 1, & \text{sinon} \end{cases}$$

D'après l'équation précédente, $F(0)$ va donc correspondre au coefficient DC, alors que les $F(k)$ $(1 \leq k < N)$ constitueront les différents coefficients AC. Il est possible à partir de ces N coefficients, de retrouver le signal f d'origine en appliquant la transformation inverse définie par la formule II.1.13.

$$f(n) = \sqrt{\frac{2}{N}} \sum_{k=0}^{N-1} C_k F(k) \cos\left[\frac{(2n+1)k\pi}{2N}\right] \tag{II.1.13}$$

Dans notre cas, nous travaillons sur des images et on doit exploiter la corrélation sur des blocs 2D de l'image. Pour cela, on applique la DCT dans sa version bi-dimensionnelle dont la formule est donnée par l'équation II.1.14 (la transformée inverse est donnée par l'équation II.1.15). Une des propriétés majeures de la DCT est que celle-ci est séparable. On peut donc calculer la DCT 2D en appliquant tout d'abord la DCT 1D sur les lignes du bloc à transformer puis la ré-appliquer ensuite sur les colonnes.

$$F(k,l) = \frac{2}{\sqrt{NM}} C_k C_l \sum_{m=0}^{N-1} \sum_{n=0}^{M-1} f(m,n) \cos\left[\frac{(2n+1)l\pi}{2M}\right] \cos\left[\frac{(2m+1)k\pi}{2N}\right] \quad \text{(II.1.14)}$$

$$f(m,n) = \frac{2}{\sqrt{NM}} \sum_{k=0}^{N-1} \sum_{l=0}^{M-1} C_k C_l F(k,l) \cos\left[\frac{(2n+1)l\pi}{2M}\right] \cos\left[\frac{(2m+1)k\pi}{2N}\right] \quad \text{(II.1.15)}$$

L'avantage majeur de la DCT, et c'est pourquoi elle est utilisée dans de nombreux algorithmes de compression, est qu'elle va donner une représentation fréquentielle du signal à l'aide de coefficients réels (contrairement à la transformée de Fourier qui utilise des coefficients complexes) conformément à la figure II.1.10. Cette transformée permet à l'utilisateur de privilégier certaines fréquences (généralement les basses) via l'utilisation de tables de quantification adaptées.

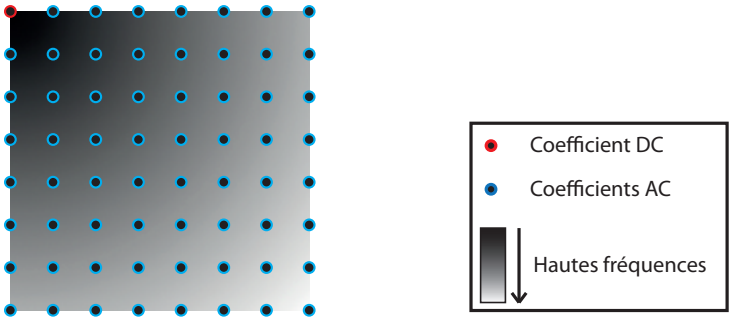


FIGURE II.1.10 – Répartition fréquentielle de la DCT sur un bloc 8*8.

La DCT est notamment utilisée par le format JPEG (voir ci-dessous) ainsi que par la majorité des codecs vidéo, comme par exemple le codec H.264/AVC (section II.1.3.3). Le format d'images JPEG, très connu et largement utilisé (notamment pour le stockage d'images sur internet) constitue un standard pour la compression d'images 2D et c'est pourquoi la section suivante est dédiée à la description des différentes étapes de cet algorithme. Celui-ci sera également mentionné en section III.3.1 où il sera adapté dans le cadre de notre schéma de compression basé DCT-3D.

II.1.2.1.1 Le standard JPEG

En 1991, le comité JPEG [2] (Joint Photographic Experts Group), un groupe d'experts chargé de standardiser le problème de la compression d'images fixes définit son nouveau standard : le

2. http ://www.jpeg.org/

format JPEG (Zhang (1990), Wallace (1991)). Celui-ci fera d'ailleurs l'objet d'une norme ISO (a) l'année suivante. Cet algorithme permet la compression d'images niveaux de gris ou RGB en proposant deux modes possibles :

- Un mode de compression sans perte basé sur la prédiction du pixel courant à partir de son voisinage (Lossless JPEG).
- Un mode de compression avec pertes basé, lui, sur la DCT et permettant de choisir la qualité de l'image reconstruite via l'utilisation d'un paramètre dédié.

JPEG est aujourd'hui, toujours considéré comme un standard en matière de compression d'images fixes (même comparé à JPEG 2000) et son utilisation reste omniprésente notamment pour le stockage des images de sites internet ainsi que dans les algorithmes de compression vidéo de la famille MPEG. Toutefois, le mode Lossless JPEG reste marginal, de nombreux algorithmes basés sur la prédiction et la modélisation de contextes (type CALIC ou JPEG-LS) obtiennent de meilleurs résultats. Pour cela, nous détaillerons dans ce qui suit, uniquement le deuxième mode de compression.

L'algorithme JPEG fonctionne conformément au pipeline défini par la figure II.1.9, se composant d'une transformation colorimétrique accompagnée d'un sous-échantillonnage éventuel des composantes de chrominance, suivie d'une DCT bi-dimensionnelle, puis de l'étape de quantification. Enfin, les différents coefficients quantifiés sont codés à l'aide d'un algorithme de codage entropique sans perte. Ces différentes étapes sont détaillées dans les parties suivantes.

a) Transformation colorimétrique vers l'espace de couleur YCbCr et sous-échantillonnage

La première des opérations à effectuer sur l'image à compresser va donc être de transformer celle-ci vers l'espace de couleurs YCbCr. Pour cela, on applique la formule II.1.16 sur les composantes R, G et B de chaque pixel de l'image.

$$\begin{cases} Y &= 0.299 \times R + 0.587 \times G + 0.114 \times B \\ Cb &= -0.1687 \times R - 0.3313 \times G + 0.5 \times B + 128 \\ Cr &= 0.5 \times R - 0.4187 \times G - 0.0813 \times B + 128 \end{cases} \tag{II.1.16}$$

Une fois cette transformation effectuée pour chaque pixel, on dispose d'une composante de luminance (Y) et de deux composantes de chrominance (Cb pour la chrominance bleue, et Cr pour la chrominance rouge). Le choix de cet espace de couleur n'est pas anodin, il permet tout d'abord de réaliser une dé-corrélation de l'information sur les composantes Cb et Cr vis-à-vis de la composante Y. De plus, conformément à l'assertion (1) de la section II.1.2, l'œil étant moins sensible aux changements de chrominance, on va pouvoir sous-échantillonner les deux composantes Cb et Cr pour réduire de façon conséquente le volume d'information à compresser sans pour autant altérer la qualité globale perçue de l'image. Pour cela, JPEG propose différents modes de sous-échantillonnage tels que 4:4:4, 4:2:2, 4:1:1 ou le plus répandu 4:2:0. La figure II.1.11 décrit les différents modes de sous-échantillonnage.

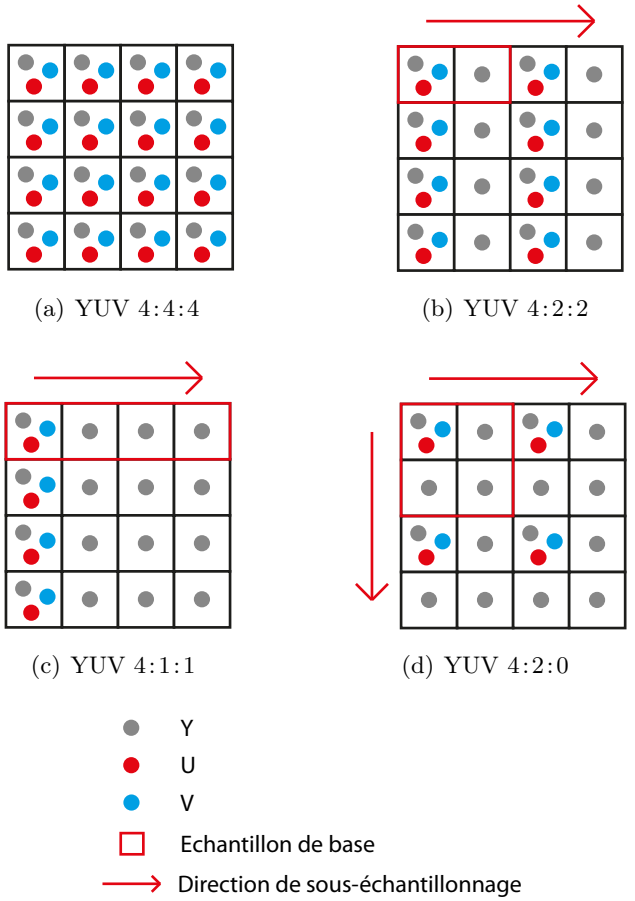


FIGURE II.1.11 – Schémas de codage couleur YUV les plus utilisés.

b) Application de la DCT bi-dimensionnelle et quantification

Une fois la conversion colorimétrique (et sous-échantillonnage éventuel) effectuée, JPEG découpe l'image en blocs de taille 8×8 pixels et applique la DCT bi-dimensionnelle sur chacun d'eux conformément à l'équation II.1.14. Une fois le passage dans le domaine fréquentiel effectué, on dispose de blocs répartissant les différentes fréquences de manière spatiale (comme indiqué dans la figure II.1.10). D'après l'assertion (2) de la section II.1.2, on peut maintenant procéder à l'étape de quantification qui va permettre d'ignorer les hautes-fréquences (détails) afin de réduire l'entropie au niveau désiré et ainsi atteindre des ratios de compression plus importants. De manière générale, on calcule la valeur quantifiée F' à partir du coefficient DCT F et du facteur de quantification Q en utilisant l'équation II.1.22.

$$F' = \lfloor \frac{F}{Q} \rfloor \tag{II.1.17}$$

Afin de déduire le facteur de quantification Q, on utilise deux tables de quantification de base (une pour la composante de luminance, l'autre pour les deux composantes de chrominance)

ainsi que le facteur de qualité q défini par l'utilisateur (q pouvant être compris entre 1 et 100, 100 étant la qualité d'image maximale). Le standard JPEG nous fournit les deux tables de base suivantes (définies après différents tests basés PSNR (Peek Signal to Noise Ratio) sur des ensembles d'images de natures diverses) :

16	11	10	16	24	40	51	61
12	12	14	19	26	58	50	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

(a)

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	29	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

(b)

TABLEAU II.1.8 – Tables de quantification de base définies par le standard JPEG pour : (a) la luminance, (b) la chrominance

A partir de ces deux tables de base, on va pouvoir calculer les deux tables de quantification spécifiques au facteur de qualité q. Pour cela, on utilise l'équation II.1.18.

$$Q(i,j) = \begin{cases} ((5000/q) \times B(i,j) + 50)/100, & \text{si } q < 50, \\ ((200 - 2 \times q) \times B(i,j) + 50)/100, & \text{sinon} \end{cases} \tag{II.1.18}$$

où $B(i,j)$ représente la table de quantification de base associée (luminance ou chrominance). On peut remarquer que plus on monte dans les hautes fréquences (coin inférieur droit des blocs 8×8), plus le facteur de quantification est élevé permettant ainsi d'omettre les hautes fréquences et de réduire de manière significative l'entropie globale du bloc pour la phase de codage entropique. De plus, pour un facteur de qualité de 50, les deux tables restent inchangées. Il suffit maintenant pour chaque bloc 8×8, d'appliquer l'équation II.1.17 en utilisant la table ainsi générée.

c) Parcours Zig-Zag du bloc et codage entropique

On doit maintenant parcourir chaque bloc 8×8 afin de coder les 64 coefficients le constituant. D'après les tables de quantification précédentes, il est facile de voir que plus l'on va aller dans les hautes fréquences (coin inférieur droit du bloc), plus on a de chance de trouver des valeurs nulles. Afin de minimiser l'entropie globale après le parcours (et ainsi atteindre des ratios de compression plus importants), on va devoir constituer des suites de valeurs nulles les plus longues possibles. Pour cela, la meilleure solution consiste à utiliser un parcours zig-zag (voir figure II.1.12) : celui-ci parcourt les différentes fréquences en partant du coefficient DC (à l'emplacement (0,0)) jusqu'aux fréquences les plus hautes orthogonalement à la diagonale, maximisant ainsi le nombre de valeurs nulles consécutives en fin de parcours.

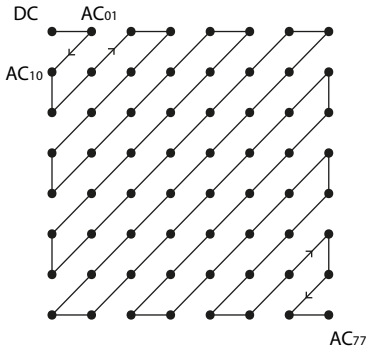


FIGURE II.1.12 – Le parcours zig-zag utilisé par le format JPEG

Une fois la procédure de parcours terminée, on dispose d'une suite de 64 coefficients (1 coefficient DC et 63 coefficients AC). Il faut maintenant effectuer le codage entropique de ceux-ci afin de produire le train de bits constituant les données image du fichier compressé. Ce codage ne va pas se faire de manière identique pour le coefficient DC et pour les 63 coefficients AC. Détaillons tout d'abord la procédure d'encodage du coefficient DC. D'après la formule de la DCT bi-dimensionelle, ce coefficient est compris dans l'intervalle $[-1024, 1024]$. Afin de réduire l'amplitude de celui-ci et ainsi utiliser moins de bits pour coder ce nombre, on va tout d'abord soustraire à cette valeur la valeur du coefficient DC du bloc précédemment encodé. Dans la majorité des cas, cela permet de réduire l'amplitude de celui-ci car deux blocs voisins ont de fortes probabilités d'avoir un coefficient DC relativement proches. L'intervalle des valeurs à coder passe donc de $[-1024, 1024]$ à $[-2048, 2048]$ (une valeur absolue de 2048 est le pire des cas).

Pour coder cette valeur, on va utiliser des codes hexadécimaux allant de la valeur (00) à la valeur $(0B)$. Pour l'encodage du coefficient DC, le premier digit ne sert pas, le deuxième digit quant à lui, va définir un intervalle de valeurs possibles ainsi que le nombre de bits supplémentaires nécessaires à ajouter pour coder celui-ci (voir table II.1.9). Ces intervalles sont gérés de manière inclusive. Si l'on note I_x l'intervalle de valeurs correspondant au code $(0x)$, celui-ci couvre l'ensemble des valeurs suivantes $I_x = [-2^x - 1, -2^{x-1}] \bigcup [2^{x-1}, 2^x - 1]$. Prenons un exemple concret : considérons le code (02) qui d'après la formule permet de coder les valeurs de l'intervalle $I_2 = [-3, -2] \bigcup [2, 3]$, on constate bien qu'il contient 4 valeurs différentes et donc que l'on a besoin de 2 bits supplémentaires afin de pouvoir reconstituer l'information durant la phase de décodage.

Le codage des coefficients AC se fait de manière un peu différente. On va toujours utiliser des codes hexadécimaux $(zeroRun, size) = (xy)$ à ceci près que le premier digit va maintenant servir à symboliser une suite de x zéros. Le deuxième digit, quant à lui, de manière semblable au paragraphe précédent, détermine l'amplitude du coefficient AC suivant les x valeurs nulles et permet de coder celui-ci à l'aide de bits additionnels (comme pour le coefficient DC). Ces codes hexadécimaux vont de la valeur 00 signifiant la fin du bloc (si la fin de celui-ci n'est pas encore atteinte, on complète le bloc avec des zéros) à la valeur (FA) (car les coefficients AC ont des valeurs appartenant à l'intervalle $[-1023, 1023]$). A noter que le code $(F0)$ symbolise une suite de 16 zéros qui peut être répétée dans le cas, par exemple, d'une suite de n zéros ($n > 16$) suivie d'un coefficient AC non nul.

Les codes hexadécimaux (utilisés pour les coefficients DC et AC) sont codés soit en utilisant les tables de Huffman statiques fournies par le standard (4 tables : DC luminance, DC chrominance, AC luminance et AC chrominance, qui, comme pour les tables de quantification de base, ont été générées à partir de tests sur différents types d'images) soit en construisant celles-ci après une première passe de récupération des statistiques de l'image. Celles-ci sont disponibles dans l'annexe K de ISO (a).

Code hexadécimal	Nombre de bits supplémentaires	Intervalle géré
00	0	0
01	1	$[-1, 1]$
02	2	$[-3, -2] \bigcup [2, 3]$
03	3	$[-7, -4] \bigcup [4, 7]$
04	4	$[-15, -8] \bigcup [8, 15]$
05	5	$[-31, -16] \bigcup [16, 31]$
06	6	$[-63, -32] \bigcup [32, 63]$
07	7	$[-127, -64] \bigcup [64, 127]$
08	8	$[-255, -128] \bigcup [128, 255]$
09	9	$[-511, -256] \bigcup [256, 511]$
0A	10	$[-1023, -512] \bigcup [512, 1023]$
0B	11	$[-2047, -1024] \bigcup [1024, 2047]$

Tableau II.1.9 – Table de codage des coefficients DC

II.1.2.2 Méthodes basées DWT

La transformée en ondelettes discrète (DWT) a été introduite pour la première fois par Croiser *et al.* (1976) et est dédiée à la décomposition d'un signal discret. La même année, Crochiere *et al.* (1976) présentent des travaux relativement similaires pour le codage de signaux sonores et nomment leur approche "codage en sous-bandes". Ce codage en sous-bandes constitue la base de toutes les approches de compression d'images basées sur la DWT. Ce codage en sous-bandes va nous permettre de décomposer le signal d'entrée, comme son nom l'indique, en différentes sous-bandes fréquentielles afin de pouvoir coder celles-ci de manière différente. Cette décomposition va pouvoir être possible grâce à l'utilisation des ondelettes via la DWT.

Dans le cas discret, on associe généralement l'application de la DWT à une opération de filtrage entre le signal d'entrée $x(N)$ et deux filtres : un filtre passe-bas demi-bande qui va nous permettre de récupérer les fréquences faibles associées à l'allure approximative du signal, et un filtre passe-haut (demi-bande lui aussi) permettant d'obtenir les hautes fréquences qui sont, elles, associées aux détails du signal. De manière pratique, le filtrage d'un signal se fait grâce à une

opération de convolution (notée $\star$ et définie par l'équation II.1.19).

$$y(n) = x \star h(n) = \sum_k h(k)x(n-k) \tag{II.1.19}$$

où $x(n)$ désigne le signal à filtrer, $h(n)$ désigne le filtre à utiliser et $y(n)$ représente le signal filtré.

Une fois le signal d'origine filtré par le filtre passe-bas demi-bande et par le filtre passe-haut demi-bande, on obtient respectivement deux signaux $y_{bas}(n)$ et $y_{haut}(n)$ contenant chacun n échantillons. Le problème réside dans le fait qu'après cette première étape de filtrage, on multiplie par deux le nombre de coefficients ce qui n'est pas idéal pour la compression. Toutefois, conformément à la règle de Nyquist, on peut maintenant sous-échantillonner les signaux $y_{bas}(n)$ et $y_{haut}(n)$ par un facteur 2 sans perdre d'information car la plage des fréquences a elle aussi été divisée par deux. Cette opération de sous-échantillonnage est généralement appelée décimation.

Ces deux opérations (filtrage et décimation) peuvent maintenant être décrites dans une seule et même équation :

$$[y]_{\downarrow 2}(n) = y(2n) = x \star h(2n) = \sum_k h(k)x(2n-k) \tag{II.1.20}$$

où $[.]_{\downarrow 2}$ désigne la décimation par un facteur 2.

Après avoir introduit ces différentes notions, voyons maintenant comment est appliquée la DWT sur un signal discret $x(n)$. La DWT va donc analyser le signal d'entrée sur différentes sous-bandes fréquentielles à différentes résolutions en décomposant celui-ci en deux sous-signaux : un signal d'approximation et un autre contenant les détails. Pour cela, la DWT utilise deux types de fonction : une fonction d'échelle (associée à un filtre passe-bas demi-bande g) et une fonction d'ondelette (associée, elle, à un filtre passe-haut demi-bande h).

II.1.2.2.1 JPEG 2000

Dés 1995, le comité JPEG (Joint Photographic Expert Group) décide de mettre au point un nouveau format de compression d'images visant à succéder au précédent standard du groupe : le format JPEG. Ce nouveau format, nommé JPEG 2000 (Taubman et Marcellin (2002), puis normalisé en Mars 2000 par le comité ISO ISO (b)), est basé sur les ondelettes et devra remplir plusieurs objectifs majeurs :

– Contrairement à son prédécesseur JPEG, qui travaille sur des blocs de données de taille 8×8 (approche locale), JPEG 2000 sera basé sur la transformée en ondelettes (approche globale) et permettra de se débarrasser des artefacts de blocs à fort taux de compression.
– L'algorithme JPEG 2000 devra permettre des taux de compression inférieurs à 0.25 bits/pixel et ce, même pour des images niveaux gris comportant beaucoup de détails.
– Il devra aussi effectuer une décompression progressive de l'image.

De manière analogue au format JPEG, le principe général de fonctionnement de JPEG 2000 suit le pipeline décrit par la figure II.1.9 de la section II.1.2. Les différentes sections suivantes décrivent les différentes opérations réalisées dans chacun des blocs du pipeline.

a) Conversion colorimétrique

JPEG 2000 dispose de deux types de transformation colorimétrique différents : dans le cas d'une compression avec pertes, il va appliquer la formule II.1.16 de la section II.1.2.1.1 afin de passer de l'espace RGB vers l'espace YCbCr. Si l'utilisateur souhaite utiliser l'algorithme dans sa version sans perte, JPEG 2000 définit une transformation réalisée à l'aide d'entiers afin de ne générer aucune perte sur le signal. Cette transformation désignée par le terme RCT (pour Reversible Color Transform) est définie par la formule suivante :

$$Y_0 = \lfloor \frac{R + 2G + B}{4} \rfloor, \qquad Y_1 = B - G, \qquad Y_2 = R - G \tag{II.1.21}$$

Une fois la conversion colorimétrique effectuée, une étape de sous-échantillonnage peut être réalisée sur les composantes 1 et 2 de l'image (si la compression avec pertes est utilisée) conformément à l'un des différents schémas de sous-échantillonnage décrit par la figure II.1.11 de la section II.1.2.1.1. L'étape suivante, spécifique à JPEG 2000 et n'apparaissant pas sur le schéma II.1.9 est le découpage de l'image en tuiles. Ces tuiles, désignées par le terme "tiles" dans le standard, sont des zones rectangulaires ne se recouvrant pas et toutes de même taille au sein d'une même composante (sauf dans le cas de celles situés sur les bords de l'image). L'avantage majeur de l'utilisation de tuiles réside dans le fait que le décodeur va pouvoir identifier les différentes tuiles lors du processus de décodage et ainsi permettre la gestion de régions d'intérêt (qui fait partie des impératifs imposés par le comité JPEG pour le design de l'algorithme JPEG 2000) et décompresser uniquement l'information nécessaire. La suite des différents traitements va donc s'effectuer de manière indépendante sur chaque tuile.

b) Application de la DWT

Nous allons maintenant détailler le processus d'application de la transformée en ondelettes discrète (DWT). Cette étape, comme la DCT pour le format JPEG, va nous permettre de séparer l'image en sous-bandes fréquentielles et ce, à différentes résolutions. Le standard JPEG 2000 définit deux types de transformation : une transformation utilisant des coefficients réels (pouvant générer des imprécisions) et une transformation utilisant, elle, des coefficients entiers

pour la compression sans perte.

Dans les deux cas, afin de réaliser une transformation en ondelettes sur deux dimensions, on applique tout d'abord la transformée selon les lignes, puis ensuite selon les colonnes. Cette opération représente un niveau de décomposition et produit 4 sous-bandes : LL1, LH1, HL1, et HH1. Conformément aux explications fournies dans la section II.1.2.2, la sous-bande LL1 représente une version sous-échantillonnée par un facteur 2 (horizontalement et verticalement) de l'image d'origine. Les 3 autres sous-bandes (LH1, HL1, HH1) représentent elles, des versions sous-échantillonnées de l'information résiduelle (correspondant aux hautes fréquences de l'image d'origine respectivement dans les directions horizontales, verticales et diagonales) et nécessaires pour reconstituer le signal d'origine à partir de la sous-bande LL1. Une fois un premier niveau de décomposition effectué, on peut ensuite appliquer la DWT sur la sous-bande LL1, afin d'obtenir les 4 sous-bandes LL2, HL2, LH2, et HH2. Ce type de décomposition est appelée "décomposition pyramidale" et est illustrée par la figure II.1.13 dans le cas d'une décomposition sur deux niveaux.

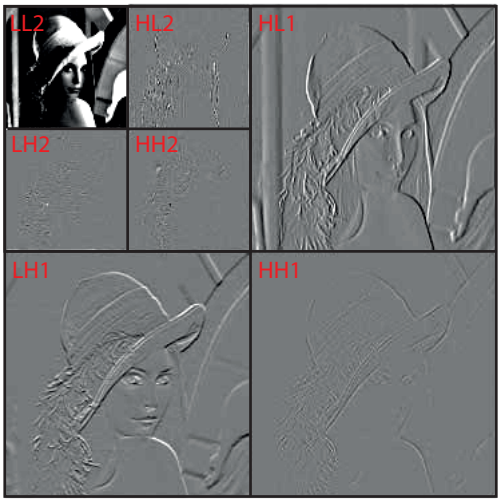


FIGURE II.1.13 – Décomposition pyramidale sur 2 niveaux.

De manière théorique, JPEG 2000 permet d'effectuer jusqu'à 32 niveaux de décomposition (si la taille de l'image le permet), mais dans la pratique on constate de manière générale, qu'aller au-delà de 5 niveaux de décomposition n'apporte aucun bénéfice en terme de compression.

JPEG 2000 propose deux approches possibles pour réaliser sa transformée en ondelettes discrète : l'une basée sur la convolution et l'autre basée sur les ondelettes liftées. Le principe de l'application de la DWT en utilisant la convolution est détaillé dans la section II.1.2.2. Pour plus d'informations sur les ondelettes liftées, le lecteur peut se référer à Stollnitz *et al.* (1996) et Sweldens et Schröder (1996). Voyons maintenant les deux types d'ondelettes utilisées par le standard.

Pour le mode de compression avec pertes, JPEG 2000 utilise les filtres Daubechies (9,7) Daubechies (1992), Antonini *et al.* (1992) constitués de deux filtres d'analyse passe-bas et passe-haut (contenant respectivement 9 et 7 coefficients réels) et deux filtres de synthèse (utilisés pour la transformation inverse) qui contiennent 7 coefficients (pour le passe-bas) et 9 coefficients pour le passe-haut. Ces filtres sont définis par les coefficients suivants :

– Pour le filtre passe-bas d'analyse : $[h_{-4}, h_{-3}, h_{-2}, h_{-1}, h_0, h_1, h_2, h_3, h_4]$ où :

$h_0 = +0.602949018236358$
$h_{\pm 1} = +0.266864118442872$
$h_{\pm 2} = -0.078223266528988$
$h_{\pm 3} = -0.016864118442875$
$h_{\pm 4} = +0.026748757410810$

– Pour le filtre passe-haut d'analyse : $[g_{-3}, g_{-2}, g_{-1}, g_0, g_1, g_2, g_3]$ où :

$g_0 = +1.115087052456994$
$g_{\pm 1} = -0.591271763114247$
$g_{\pm 2} = -0.057543526228500$
$g_{\pm 3} = +0.0912717631142495$

– Pour le filtre passe-bas de synthèse : $[h'_{-3}, h'_{-2}, h'_{-1}, h'_0, h'_1, h'_2, h'_3]$ où :

$h'_0 = +1.115087052456994$
$h'_{\pm 1} = +0.591271763114247$
$h'_{\pm 2} = -0.057543526228500$
$h'_{\pm 3} = -0.0912717631142495$

– Pour le filtre passe-haut de synthèse : $[g'_{-4}, g'_{-3}, g'_{-2}, g'_{-1}, g'_0, g'_1, g'_2, g'_3, g'_4]$ où :

$g'_0 = +0.602949018236358$
$g'_{\pm 1} = -0.266864118442872$
$g'_{\pm 2} = -0.078223266528988$
$g'_{\pm 3} = +0.016864118442875$
$g'_{\pm 4} = +0.026748757410810$

Pour le mode de compression sans perte, JPEG 2000 utilise l'ondelette de Le Gall(5,3) (Calderbank *et al.* (1998)). Cette ondelette utilise des coefficients à valeurs entières afin de permettre un stockage sans perte des coefficients et ainsi ne pas produire de déformation sur le signal d'origine. Elle est constituée de deux couples de filtres (deux filtres d'analyse : passe-bas et passe-haut contenant respectivement 5 et 3 coefficients, et deux filtres de synthèse : passe-bas et passe-haut contenant respectivement 3 et 5 coefficients) définis par :

– Pour le filtre passe-bas d'analyse : $[h_{-2}, h_{-1}, h_0, h_1, h_2]$ où :

$h_0 = 3/4$
$h_{\pm 1} = 1/4$
$h_{\pm 2} = -1/8$

– Pour le filtre passe-haut d'analyse : $[g_{-1}, g_0, g_1]$ où :

$g_0 = 1$
$g_{\pm 1} = -1/2$

– Pour le filtre passe-bas de synthèse : $[h'_{-1}, h'_0, h'_1]$ où :

$h'_0 = 1$
$h'_{\pm 1} = 1/2$

– Pour le filtre passe-haut de synthèse : $[g'_{-2}, g'_{-1}, g'_0, g'_1, g'_2]$ où :

$g'_0 = 3/4$
$g'_{\pm 1} = -1/4$
$g'_{\pm 2} = -1/8$

Lorsque la convolution est appliquée selon les lignes puis ensuite selon les colonnes (à l'aide de l'une ou l'autre des deux ondelettes), le calcul de celle-ci sur les bords des tuiles peut être un problème car on ne possède pas l'information extérieure à chaque tuile. Pour remédier à ce problème, JPEG 2000 réalise une extension symétrique des coefficients selon les lignes puis selon les colonnes. La figure II.1.14 illustre le processus d'extension au bord d'une tuile lors de la convolution, selon les lignes à l'aide de l'ondelette Le Gall (5, 3).

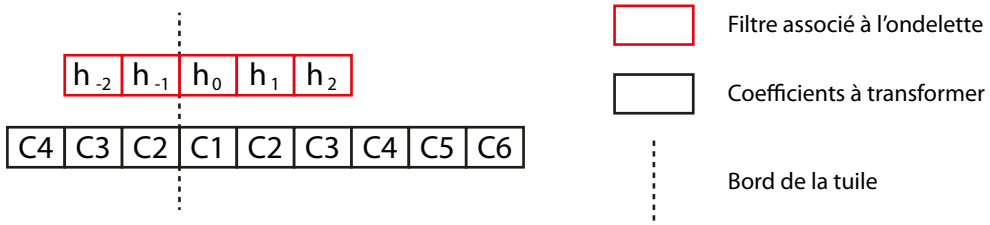


FIGURE II.1.14 – Extension symétrique afin de gérer les bords de la tuile.

c) Quantification

Une fois la DWT appliquée, on peut maintenant procéder à l'étape de quantification durant laquelle on va pouvoir réduire considérablement la taille des données. Cette opération est réalisée en utilisant une quantification scalaire à zone morte utilisant un paramètre Δd appelé pas de quantification. Ce paramètre va être spécifique à chaque sous-bande et est déterminé à partir de l'échelle des coefficients composant celle-ci. A partir de ce paramètre, on peut donc quantifier un coefficient d'ondelette a_x appartenant à une sous-bande x en utilisant le pas Δd_x associé à celle-ci à partir de la formule II.1.22.

$$q_x = signe(a_x)\lfloor \frac{|a_x|}{\Delta d_x} \rfloor \tag{II.1.22}$$

d) Codage entropique

Le codage entropique est de loin l'étape la plus complexe de l'algorithme JPEG 2000. Avant de présenter de manière générale la procédure de codage entropique, il convient de préciser que de manière semblable à l'étape de découpage en tuiles, chaque tuile est à son tour subdivisée en morceaux appelés code-blocks (généralement de taille 32×32 ou 64×64) toujours dans le but de pouvoir se déplacer rapidement dans le flux binaire constituant le fichier compressé et ainsi décompresser de manière efficace une zone donnée de l'image.

Le codage entropique va ensuite être réalisé de manière indépendante sur chaque code-block. Pour cela, JPEG 2000 utilise deux algorithmes distincts : EBCOT (Taubman (2000), Taubman *et al.* (2001)) qui va réaliser un codage par plan de bits sur chaque code-block et un codeur arithmétique binaire nommé MQ-Coder (Taubman et Marcellin (2002)) . La figure II.1.15 illustre le fonctionnement général du processus de codage entropique et la manière dont ces deux algorithmes travaillent ensemble.

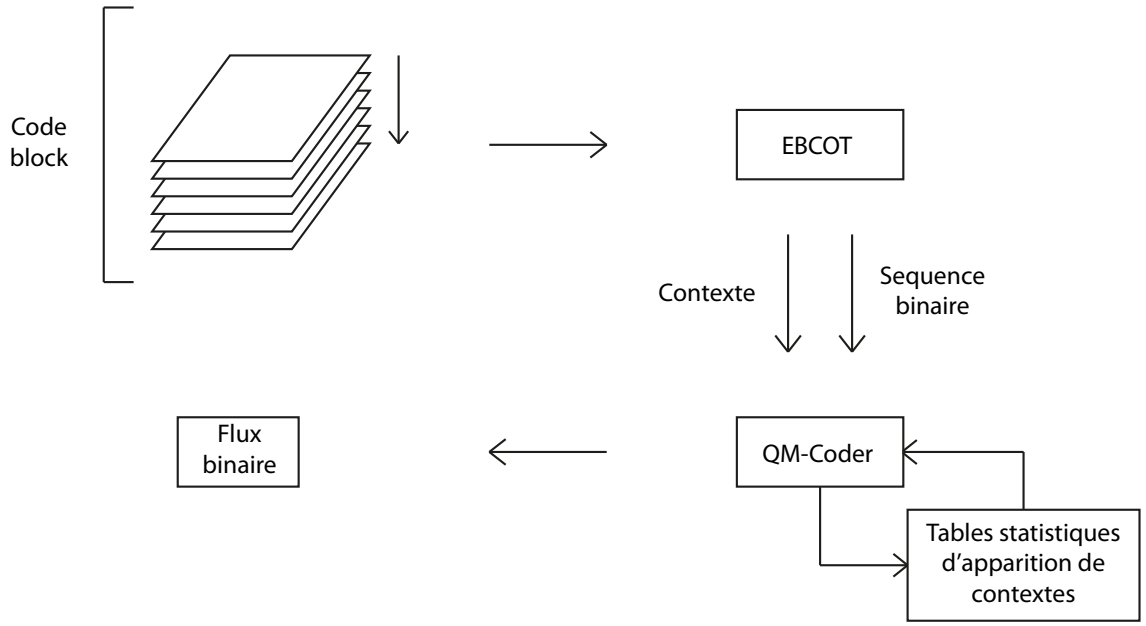


FIGURE II.1.15 – Fonctionnement général du codage entropique de JPEG 2000.

EBCOT est un codeur par plan de bits : pour chaque code-block, celui-ci va considérer les différents plans de bits en commençant par celui contenant les bits les plus significatifs en les parcourant colonne par colonne. Pour chaque plan de bits, l'algorithme va effectuer trois passes dans cet ordre :

- Passe de propagation de la signifiance : Cette passe va encoder tous les bits n'appartenant pas à un coefficient considéré comme significatif, mais ayant un voisin immédiat (parmi les 8) considéré, lui, comme significatif.
- Passe de raffinement de la magnitude : Cette passe va tout simplement encoder les bits appartenant à des coefficients ayant été déclarés significatifs dans un plan de bits antérieur.
- Passe de nettoyage : Tous les bits appartenant à des coefficients non significatifs sont encodés durant cette passe.

Il est clair que pour le premier plan de bits, aucun coefficient n'est considéré comme significatif. C'est pourquoi lorsque l'algorithme démarre, celui-ci effectue uniquement la passe de nettoyage lors de laquelle un coefficient est considéré comme significatif si le bit associé à celui-ci est égal à 1. Une fois le premier plan de bits traité, l'algorithme continue alors son processus normal.

Pour le codage d'un bit, ou d'une suite de bits, EBCOT va générer en sortie deux types d'informations : un symbole correspondant à un contexte donné ainsi qu'une chaîne binaire (de longueur 1 ou plus). Chaque couple (symbole, chaîne binaire) ainsi généré est envoyé au MQ-Coder qui va coder arithmétiquement la chaîne binaire à l'aide des tables de probabilités correspondant au contexte associé au symbole. Le codage arithmétique est réalisé de manière adaptative : pour chaque couple (symbole, chaîne binaire), les différentes tables statistiques sont mises à jour.

D'autres approches alternatives (EZW, SPIHT, voir ci-dessous) sont possibles pour le codage des coefficients d'ondelettes. Même si on constate dans Taubman (2000) que EBCOT permet d'obtenir des résultats en terme de débit/distorsion, très légèrement supérieurs à ceux obtenus avec SPIHT, la complexité algorithmique d'EBCOT reste bien supérieure par rapport à celle de SPIHT. Nous présentons ci-après la méthode SPIHT qui est par ailleurs utilisée dans l'algorithme que nous proposons dans la section III.3.1.2.2.

II.1.2.2.2 SPIHT : Set Partitioning In Hierarchical Trees

Cette section présente l'algorithme SPIHT mis au point par Said et Pearlman (1996). Cet algorithme est en fait une extension de l'algorithme EZW (Embedded Zerotree Wavelets) proposé par Shapiro (1993) qui vise à coder une image ayant subi une transformation par ondelettes afin de produire un train binaire progressif. La principale caractéristique d'un flux binaire progressif est que quelle que soit la taille du fichier compressé, le décodeur peut s'arrêter à tout moment dans ce flux et décompresser l'image (plus le nombre de bits lus est important plus la qualité de l'image décompressée sera meilleure).

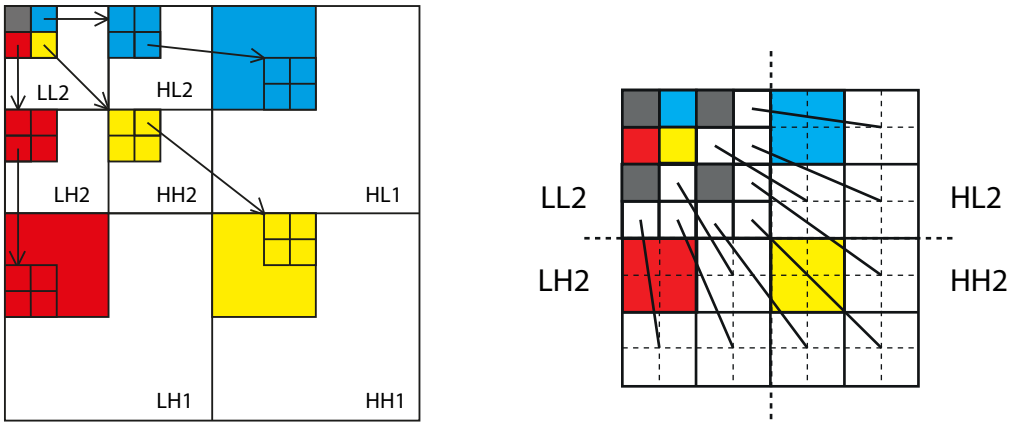
Afin de permettre ce codage progressif, l'algorithme SPIHT doit effectuer une transmission progressive des coefficients d'ondelettes appartenant aux différentes échelles de décomposition. Pour être réalisée de manière efficace, cette transmission doit tenir compte des deux propriétés importantes suivantes :

(1) Pour chaque coefficient d'ondelettes, on doit transmettre en priorité les bits de poids fort afin de retrouver une valeur proche de la valeur originale lors de la décompression.
(2) Les coefficients à forte amplitude (désignés par la suite comme coefficients "significatifs") sont généralement les plus représentatifs du signal et doivent être transmis en priorité.

L'algorithme SPIHT fonctionne en effectuant n itérations successives basées sur le nombre minimal de plans de bits nécessaires pour couvrir l'amplitude de tous les coefficients d'ondelettes après application de la DWT. A chaque itération k, le $k^{\text{ième}}$ bit de poids fort associé aux coefficients d'ondelettes considérés comme significatifs est transmis. De cette façon, la propriété (1) est naturellement satisfaite. Pour la propriété (2), on utilise la notion de "significativité" : à

chaque itération de l'algorithme, la "significativité" des différents coefficients d'ondelette est évaluée, ceux qui sont considérés comme significatifs le resteront par la suite, les autres, quant à eux sont mis de coté jusqu'à la prochaine itération de l'algorithme. Cette notion de "significativité" sera détaillée un peu plus loin.

Afin d'exploiter le caractère hiérarchique spécifique à la transformée en ondelettes, l'algorithme utilise une structure de données spéciale : un arbre à orientation spatiale (voir II.1.16(a)). Cette structure permet de se déplacer à travers les différents niveaux hiérarchiques de la transformée en ondelettes étant associés aux même localisations spatiales de l'image d'origine.



(a) (b)

FIGURE II.1.16 – (a) Structure générale de l'arbre à orientation spatiale, appliquée sur la décomposition hiérarchique de la DWT, (b) construction du premier niveau de l'arbre

Chaque arbre à orientation spatiale est construit conformément à la figure II.1.16(b) : on part de la sous-bande LL du niveau de décomposition le plus haut (dans notre exemple, le niveau 2) et on forme des groupes de coefficients de taille 2×2. Pour chaque groupe de coefficients, on garde le coefficient situé en haut à gauche isolé, les trois autres, quant à eux, sont considérés comme racines des différents arbres à orientation spatiale. On associe alors à chaque racine 4 descendants directs situés relativement à la position de celle-ci, dans son propre groupe de coefficients (conformément à la figure II.1.16(b)). Cette première étape de construction de l'arbre est particulière dans le sens où les racines des arbres et leurs descendants directs sont situés sur le même niveau hiérarchique de la DWT. Une fois ce premier niveau de l'arbre créé, on peut maintenant généraliser le processus de construction en appliquant la règle suivante : pour tout nœud de l'arbre situé à l'emplacement (i, j) et n'appartenant pas au niveau 1 de décomposition (là où se situent les feuilles de l'arbre), celui-ci a comme descendants directs les 4 nœuds situés aux emplacements suivants : $(2i, 2j)$, $(2i + 1, 2j)$, $(2i, 2j + 1)$ et $(2i + 1, 2j + 1)$.

Afin de pouvoir gérer correctement le déplacement dans cette structure d'arbre, l'algorithme SPIHT désigne les différents nœuds (incluant les feuilles) des différents arbres via leur localisation (i, j) et les sépare en 4 ensembles distincts :

- $\mathcal{O}(i, j)$: contient les différentes localisations (k, l) des descendants directs du nœud (i, j) ($\mathcal{O}(i, j) = \emptyset$ si le nœud est une feuille).
- $\mathcal{D}(i, j)$: contient les différentes localisations (k, l) des descendants directs et indirects du nœud (i, j). Les descendants indirects du nœud (i, j) est l'ensemble de tous ses descendants

moins ses descendants directs.
- $\mathcal{H}$: contient les différentes localisations des racines des différents arbres, toutes situées dans la sous-bande LL du plus haut niveau de décomposition de la DWT.
- $\mathcal{L}(i,j)$: contient les différentes localisations (k,l) des descendants indirects du nœud (i,j) ($\mathcal{L}(i,j) = \mathcal{D}(i,j) - \mathcal{O}(i,j)$).

Les différentes listes ci-dessus permettent ainsi une gestion simplifiée des différents niveaux hiérarchiques de la DWT via les arbres à orientation spatiale. La figure II.1.17 reprend l'exemple de la figure II.1.16(a) et illustre les différents ensembles présentés ci-dessus pour un nœud (i,j) racine d'un des arbres à orientation spatiale (appartenant à $\mathcal{H}$).

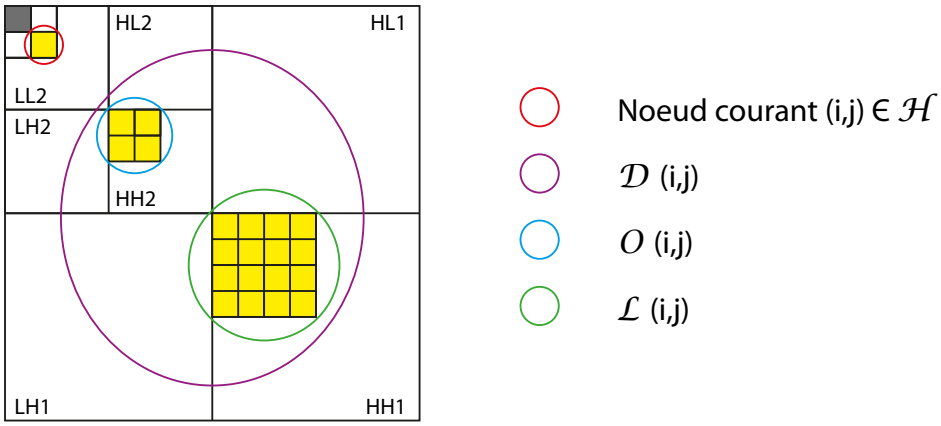


FIGURE II.1.17 – Les différents ensembles permettant la gestion des arbres à orientation spatiale.

L'algorithme utilise en outre trois listes ordonnées afin de différencier les éléments suivants :

- LIS (List of Insignificant Sets) : Cette liste contient des entrées de type $\mathcal{D}(i,j)$ ou de type $\mathcal{L}(i,j)$ contenant les localisations (k,l) des nœuds associés à des coefficients d'ondelettes considérés comme non significatifs.
- LIP (List of Insignificant Pixels) : Cette liste contient la localisation particulière d'un nœud unique, appartenant à un des arbres à orientation spatiale, et considéré comme non significatif.
- LSP (List of Significant Pixels) : Cette liste contient la localisation particulière d'un nœud unique, appartenant à un des arbres à orientation spatiale, et considéré comme significatif.

Nous avons spécifié plus haut que la propriété (2) était assurée grâce à la notion de "significativité" des différents coefficients (ou groupe de coefficients) d'ondelettes. Pour tester la "significativité" (notée $S_k(i,j)$) d'un coefficient d'ondelette $c_{i,j}$ associé à l'emplacement (i,j) et contenu dans la liste LIP lors de la $k^{\text{ième}}$ itération de l'algorithme, on utilise la formule II.1.23. Dans le cas d'un ensemble de coefficients $\mathcal{T}$ (de type $\mathcal{D}$ ou de type $\mathcal{L}$) associé à l'emplacement (i,j) et appartenant

à la liste LIS lors de la $k^{\text{ième}}$ itération de l'algorithme, on utilise cette fois-ci la formule II.1.24.

$$S_k(i,j) = \begin{cases} 1 \text{ si } |c_{i,j}| \geq 2^{n-k} \\ 0 \text{ sinon} \end{cases} \quad \text{où } n = \lfloor \log_2 \left(\max_{(i,j)}(|c_{i,j}|) \right) \rfloor \tag{II.1.23}$$

$$S_k(\mathcal{T}) = \begin{cases} 1 \text{ si } \max_{(i,j)\in\mathcal{T}} |c_{i,j}| \geq 2^{n-k} \\ 0 \text{ sinon} \end{cases} \tag{II.1.24}$$

Lorsque le coefficient associé à la localisation (i,j) et appartenant à la liste LIP est jugé significatif à la $k^{\text{ième}}$ itération de l'algorithme ($S_k(i,j) = 1$), alors celui-ci est déplacé vers la liste LSP et commencera à être codé lors de la prochaine itération. Dans le cas d'un ensemble $\mathcal{T}$ de localisations associées à une entrée de type $\mathcal{D}(i,j)$ ou de type $\mathcal{L}(i,j)$ dans la liste LIS, lorsque celui-ci est jugé significatif ($S_k(\mathcal{T}) = 1$), les descendants directs du noeud (i,j) ($= \mathcal{O}(i,j)$) sont déplacés vers les listes LIP et LSP suivant leur "significativité", et une entrée de type $\mathcal{L}(i,j)$ est ensuite ajoutée à la liste LIS afin de pouvoir descendre d'un niveau dans la hiérarchie de l'arbre. Tant que des entrées sont ajoutées dans la liste LIS, celles-ci sont traitées directement afin de pouvoir extraire les différentes localisations associées à des coefficients significatifs jusqu'au niveau contenant les feuilles de l'arbre.

L'algorithme (dont le pseudo-code est donné par l'algorithme II.1.1) se déroule en n itérations successives de deux passes distinctes : une passe de tri (sorting pass) et une passe d'affinement (refinement pass). La passe de tri se charge de détecter la "significativité" des coefficients d'ondelette associés aux localisations contenues dans la liste LIP, mais aussi de parcourir successivement les différents niveaux des arbres à orientation spatiale afin de séparer l'information significative de celle non significative en examinant les différentes entrées de types $\mathcal{D}$ et $\mathcal{L}$ dans la liste LIS. La passe d'affinement permet, elle, d'envoyer le $n^{\text{ième}}$ bit significatif des coefficients associés aux localisations contenues dans la liste LSP. Le fonctionnement en détail de l'algorithme SPIHT est donné par l'algorithme II.1.1.

Pour plus d'informations concernant l'algorithme SPIHT, le lecteur peut se référer à Said et Pearlman (1996).

Algorithme II.1.1 Pseudo-code de l'algorithme SPIHT.

Entrées:
$c_{i,j}$: Les différents coefficients d'ondelettes
Données:
n : Le nombre d'itération de l'algorithme
LSP : La liste des localisations (i,j) associées aux coefficients significatifs
LIP : La liste des localisations (i,j) associées aux coefficients non significatifs
LIS : La liste des ensembles de localisations associées aux coefficients non significatifs
début
Initialisation :
$n \longleftarrow \lfloor \log_2(\max_{i,j}(|c_{i,j}|)) \rfloor$
LSP $\longleftarrow \varnothing$
LIP $\longleftarrow \varnothing$
LIS $\longleftarrow \varnothing$
pour tout $(i,j) \in \mathcal{H}$ **faire**
ajouter (i,j) à la LIP
si $\mathcal{D}(i,j) \neq \varnothing$ **faire** ajouter (i,j) à LIS en tant qu'entrée de type A.

tant que $n \geq 0$ **faire**
1. Passe de tri :
1.1 **pour tout** $(i,j) \in$ LIP **faire**
1.1.1 écrire $S_n(i,j)$
1.1.2 **si** $S_n(i,j) = 1$ **faire**
déplacer (i,j) de la LIP vers la LSP
écrire $signe(c_{i,j})$
1.2 **pour tout** $(i,j) \in$ LIS et non évalué lors de la même passe **faire**
1.2.1 **si** $type(i,j) = A$ **faire**
écrire $S_n(\mathcal{D}(i,j))$
si $S_n(\mathcal{D}(i,j)) = 1$ **faire**
pour tout $(k,l) \in \mathcal{O}(i,j)$ **faire**
écrire $S_n(k,l)$
si $S_n(k,l) = 1$ **faire**
déplacer (k,l) vers la LSP
écrire $signe(c_{i,j})$
sinon faire
déplacer (k,l) vers la LIP
si $\mathcal{L}(i,j) \neq 0$ **faire**
ajouter (i,j) à la fin de la LIS en tant qu'entrée de type B
aller à l'étape 1.2.2
sinon supprimer (i,j) de la LIS
1.2.2 **si** $type(i,j) = B$ **faire**
écrire $S_n(\mathcal{L}(i,j))$
si $S_n(\mathcal{L}(i,j)) = 1$ **faire**
pour tout $(k,l) \in \mathcal{O}(i,j)$ ajouter (k,l) à la fin de la LIS en tant qu'entrée de type A
supprimer (i,j) de la LIS
aller à l'étape 1.2
2. Passe d'affinage :
pour tout $(i,j) \in$ LSP exceptés ceux ajoutés lors de la dernière passe de tri **faire**
écrire le $n^{\text{ième}}$ bit le plus significatif de $|c_{i,j}|$
fin

II.1.3 Généralisation à la vidéo

II.1.3.1 Pourquoi une nouvelle approche ?

L'objectif majeur des méthodes de compression est de réduire au maximum le volume d'un ensemble de données de manière à faciliter son stockage ou pour permettre une transmission sur un réseau. Comme nous l'avons précisé plus haut (II.1.2), une image présente deux types de corrélation : une corrélation colorimétrique entre les différentes composantes chromatiques de celle-ci, mais aussi une corrélation spatiale entre des groupes de pixels situés dans une même portion d'image.

Dans le cas d'une vidéo, on se retrouve confronté à un nouveau type de corrélation, la corrélation temporelle : deux images (ou *frames*) successives vont présenter de fortes similarités, similarités qui vont être d'autant plus fortes que la fréquence de capture sera élevée. Les différentes méthodes de compression présentées précédemment sont généralement utilisées afin d'exploiter la corrélation colorimétrique et spatiale au sein d'une même image. Il est clair que compresser une séquence vidéo comme étant uniquement une suite d'images indépendantes et ainsi omettre cette très forte corrélation temporelle est un choix non optimal. Il a donc fallu mettre au point de nouveaux mécanismes afin d'exploiter au maximum cette corrélation et permettre d'atteindre des taux de compression acceptables.

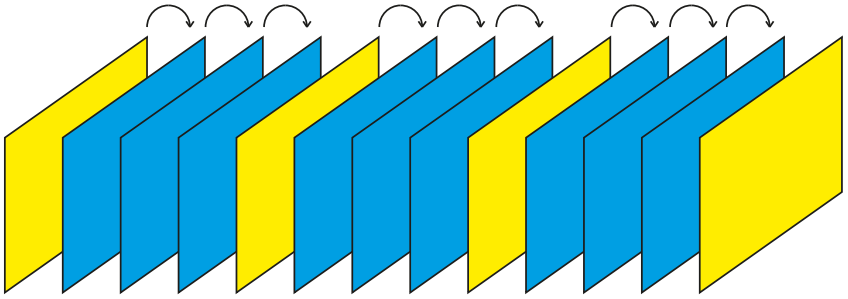
Un bon algorithme de compression vidéo devra donc comprendre un module d'exploitation de la redondance spatiale (via des transformations telles que la DCT ou la DWT) ainsi qu'un module d'exploitation de la redondance temporelle.

De manière générale, le flux vidéo est organisé en GOP (Group Of Pictures) regroupant un certain nombre de *frames* consécutives suivant le sens de lecture du flux. Ces GOP constituent une unité d'information indépendante (décoder une frame appartenant à un GOP donné nécessite uniquement l'information présente dans celui-ci). Un GOP peut contenir trois types de *frames* :

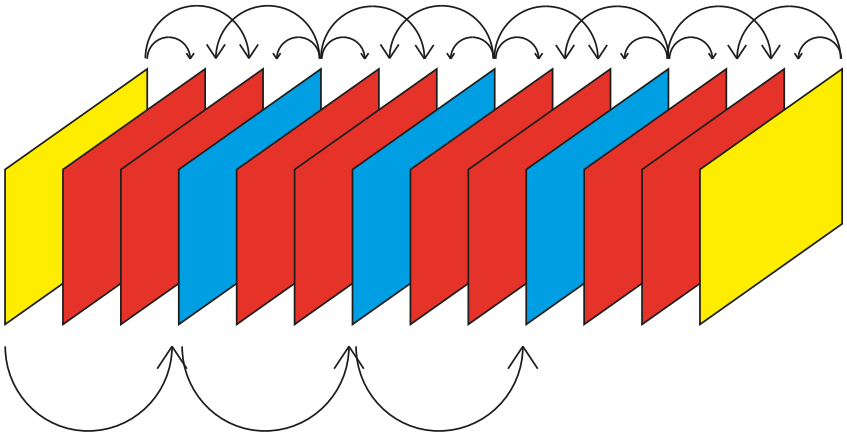
- Les *frames* I (pour Intra coded picture) : Ces *frames* sont codées en exploitant uniquement la redondance spatiale au sein de l'image via les schémas classiques utilisés pour la compression d'images fixes avec pertes (transformation dans le domaine fréquentiel/quantification/codage entropique). Une frame I au sein d'un GOP G n'a donc besoin d'aucune information provenant d'autres *frames* appartenant à G.
- Les *frames* P (pour Predictive coded picture) : Les *frames* P sont codées en utilisant le mécanisme de prédiction inter-frame (décrit dans la section II.1.3.2) à partir d'une frame I ou d'une frame P antérieure et appartenant au même GOP G.
- Enfin les *frames* B (pour Bi-predictive coded picture) sont elles aussi codées grâce à la prédiction inter-frame à partir de *frames* I ou P (antérieures et/ou postérieures) du même GOP G.

Un GOP G est défini à l'aide de deux paramètres que l'on va désigner par p_I et $p_{I/P}$ où p_I désigne le nombres de *frames* (P ou B) entre deux *frames* I consécutives et $p_{I/P}$ représente

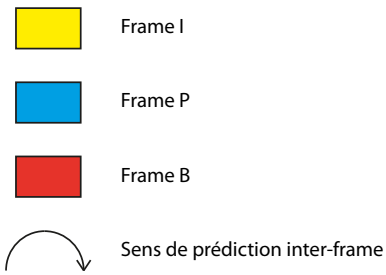
le nombre de *frames* entre deux *frames* I ou P consécutives. La figure II.1.18(a) et II.1.18(b) représentent la structure du GOP pour respectivement $(p_I, p_{I/P}) = (4, 0)$ et $(p_I, p_{I/P}) = (12, 3)$.



(a)



(b)



(c)

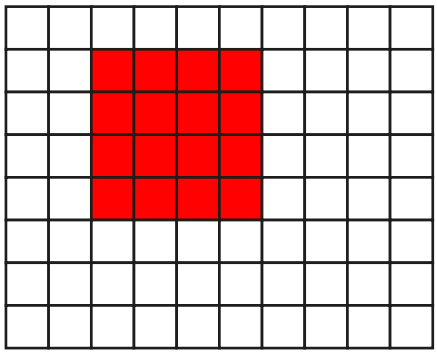
FIGURE II.1.18 – Structure d'un GOP pour : (a) $(p_I, p_{I/P}) = (4, 0)$, (b) $(p_I, p_{I/P}) = (12, 3)$.

II.1.3.2 Le système de prédiction/compensation de mouvement

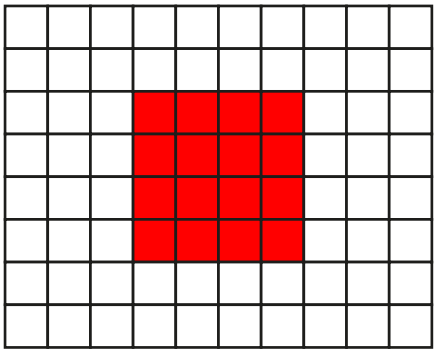
Dans cette section, nous allons présenter le mécanisme utilisé afin d'exploiter la forte redondance temporelle présente dans toute séquence vidéo : la prédiction/compensation de mouvement (généralement désignée par le terme prédiction inter-frame). Le mouvement d'une frame à la suivante dans une séquence vidéo est due à 4 facteurs :

– Le déplacement d'un objet d'une image à la suivante ;
– Le déplacement de la caméra ;
– Le dévoilement du fond de la scène (causé par un objet en mouvement) ;
– Des changements d'intensité lumineuse.

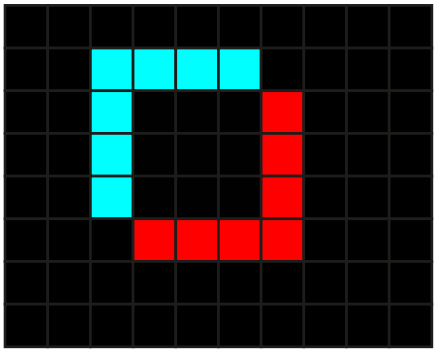
Exceptés pour les deux derniers facteurs de mouvement, cela correspond à un déplacement de pixels entre deux *frames*. L'ensemble des différentes trajectoires de ces pixels est généralement désigné par le terme de flux optique (optical flow). Considérons un exemple simple, une scène sur fond blanc constituée d'un carré rouge se déplaçant en diagonale entre les deux *frames* consécutives (II.1.19(a) et II.1.19(b)). La figure II.1.19(c) met en évidence la forte corrélation entre ces deux *frames* en calculant la différence entre celles-ci. La figure II.1.19(d) présente le flux optique correspondant.



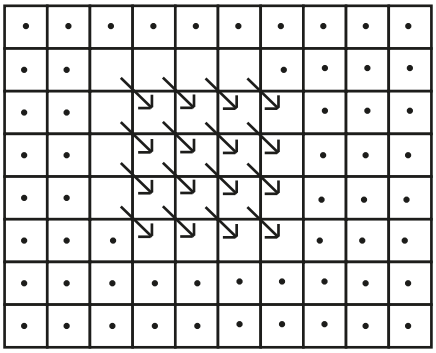
(a) Frame au temps t



(b) Frame au temps $t+1$



(c) Frame de différence entre t et $t+1$



(d) Flux optique associé aux temps t et $t+1$

FIGURE II.1.19 – Deux *frames* consécutives (a) et (b), la *frame* de différence (c) et le flux optique associé (d).

Comme on peut le voir sur la figure II.1.19(c), une grande quantité de l'information est redondante entre la frame au temps t et celle au temps $t+1$. Une manière efficace de reconstituer l'information contenue dans la frame au temps $t+1$ consisterait à utiliser la frame au temps t ainsi que les différents vecteurs de mouvement contenus dans le flux optique associé. De cette façon, seule l'information non redondante de la frame au temps t à la suivante serait codée via les vecteurs de mouvement. Toutefois, avec ce type d'approche, on se retrouve vite confronté à deux problèmes majeurs : tout d'abord, la constitution du flux optique au niveau du pixel requiert

énormément de calcul et serait trop lourd algorithmiquement, ensuite le codage des vecteurs de mouvement n'est pas gratuit (en terme de poids mémoire) et celui-ci pourrait facilement outrepasser le gain gagné par l'exploitation de la redondance spatiale.

Une manière de pouvoir contourner ces deux problèmes consiste à non pas calculer le flux optique au niveau du pixel, mais au niveau du bloc de pixels. Ainsi le nombre de vecteurs de mouvement à calculer s'en trouve réduit et donc le volume d'information à coder par la suite aussi. La figure II.1.20 illustre le flux optique non pas calculé au niveau du pixel mais cette fois-ci en utilisant des blocs de pixels de taille 2×2.

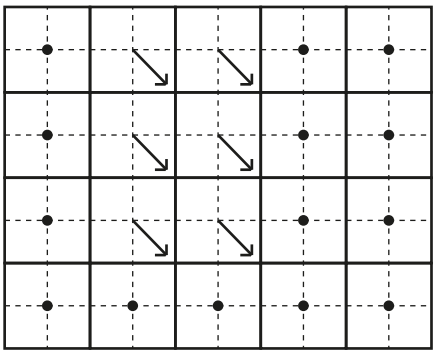


FIGURE II.1.20 – Flux optique obtenu en utilisant des blocs de pixels de taille 2×2.

On voit clairement qu'en utilisant des blocs de pixels au lieu des pixels eux-mêmes, le processus de calcul du flux optique va être de complexité moindre (le nombre de vecteurs de mouvement à calculer étant inférieur), et la quantité d'information à compresser est, de plus, divisée par 4. C'est sur cette approche que se base le principe de la prédiction/compensation de mouvement utilisée par la majorité des algorithmes de compression vidéo afin d'exploiter la redondance temporelle. Toutefois, la prédiction/compensation de mouvement ne marche pas exactement de cette façon et nous allons maintenant détailler la procédure générale de celle-ci.

Cette procédure est effectuée sur chaque bloc (de taille $M \times N$) de la frame à prédire et se décompose en deux étapes distinctes :

- L'étape d'estimation de mouvement : durant cette étape, on cherche à faire coïncider le bloc B_p en cours de traitement dans la frame à prédire F_p avec un bloc B_r d'une frame de référence F_r (qui peut être antérieure ou postérieure dans le sens de lecture du flux vidéo). Pour cela, on considère une région plus ou moins grande autour de l'emplacement de B_p et pour chaque bloc de taille $M \times N$ possible dans cette région, on compare le bloc B_p avec celui de la frame de référence. A chaque position *pos* différente contenue dans la zone de recherche, on estime la similarité S entre B_p et le bloc candidat situé à l'emplacement *pos* dans la frame de référence. Le but final est de trouver une position *pos* dans la zone de recherche, permettant de minimiser S. Afin de mesurer la similarité S, on utilise généralement une fonction de coût telle que la EQM (Erreur Quadratique Moyenne). Le bloc qui permet d'obtenir la valeur minimale de S est alors considéré comme bloc de référence (B_r) pour le bloc courant B_p.
- L'étape de compensation de mouvement : Une fois l'étape d'estimation de mouvement effectuée, on connait donc le bloc B_r à utiliser comme référence pour prédire le bloc B_p.

B_r est alors soustrait à B_p afin d'obtenir un bloc résiduel B_{res}. Ce bloc B_{res} ainsi que le vecteur de mouvement V (contenant le vecteur 2D associé au décalage entre la position centrale du bloc B_p dans la frame F_p et la position p dans la frame F_r) sont ensuite transmis pour être codés.

II.1.3.3 Le standard actuel : H.264/AVC

La norme H.264/AVC (pour Advanced Video Coding), Richardson (2010), est la norme actuelle dédiée au codage vidéo. Elle a été mise au point par le comité JVT [3] (Joint Video Team) qui est en fait la collaboration entre deux groupes d'experts spécialisés dans le codage vidéo : le groupe VCEG (Video Coding Experts Group) de l'ITU (International Telecommunication Union) ainsi que le groupe MPEG [4] (Moving Picture Experts Group) de l'ISO (International Organization for Standardization). La première version approuvée de la norme a été produite en mai 2003. La norme H.264/AVC réalise un grand pas en avant vis à vis de ses prédécesseurs car elle introduit de nouveaux concepts permettant de réduire de façon considérable le débit nécessaire au codage des vidéos tout en proposant une qualité visuelle supérieure. Nous allons présenter une partie de ces nouveaux mécanismes (à nos yeux les plus importants) qu'introduit la norme H.264/AVC à savoir :

- Le système de prédiction intra-frame
- L'utilisation d'une DCT optimisée utilisant des coefficients entiers.
- Le filtrage anti-blocs
- Le nouveau codeur entropique CABAC (Context-Adaptative Binary Arithmetic Coding).

II.1.3.3.1 Mécanisme de prédiction Intra-frame

Cette partie est dédiée à la présentation du système de prédiction intra-frame du codec H.264/AVC. A l'instar de la prédiction inter-frame qui utilise des blocs appartenant à des $frames$ différentes pour prédire le bloc courant, la prédiction intra-frame va se servir des blocs voisins au bloc courant pour réaliser sa prédiction. Ce nouveau système de prédiction apporte les deux améliorations suivantes :

- Les $frames$ I utilisent maintenant la prédiction intra-frame en amont du processus de compression DCT/quantification/codage entropique. La frame ainsi prédite est ensuite soustraite à la frame d'origine afin d'obtenir une image résiduelle dont l'entropie est bien inférieure. Cette frame résiduelle est ensuite envoyée au pipeline DCT/quantification/codage entropique pour compression.
- Les $frames$ P et B peuvent aussi utiliser la prédiction intra-frame qui peut dans certains cas se révéler plus efficace que la prédiction inter-frame.

La prédiction intra-frame du standard H.264 permet la prédiction de blocs de taille 16×16 et 4×4 pour la luminance et de taille 8×8 pour la chrominance (ceci étant dû au sous-

3. http ://www.itu.int/en/ITU-T/studygroups/com16/video/Pages/jvt.aspx
4. http ://mpeg.chiariglione.org/

échantillonnage 4:2:0 généralement utilisé). Les figures II.1.21 et II.1.22 décrivent les différents modes de prédiction disponibles pour respectivement les blocs de taille 4×4 et de taille 16×16 pour la luminance. Pour les blocs associés à la chrominance, on dispose de 4 modes de prédiction disponibles équivalents aux 4 modes de prédiction de la figure II.1.22. Il faut aussi noter que lors de l'utilisation du profile "high" de H.264/AVC, on dispose en plus d'un système d'intra-prédiction pour des blocs de luminance de taille 8×8. Dans ce cas, les différents modes de prédiction disponibles sont similaires aux 9 modes utilisés pour la prédiction des blocs de taille 4×4.

Lors de la décompression, le décodeur doit nécessairement savoir quel mode de prédiction a été utilisé afin de pouvoir reconstruire le bloc courant. On doit donc nécessairement coder le mode de prédiction utilisé. H.264/AVC utilise un système de prédiction basé sur les deux blocs voisins situés au dessus et à gauche de celui-ci du bloc courant. Le mode de prédiction le plus probable est alors défini celui ayant l'identifiant le plus petit. Si c'est effectivement le mode utilisé pour coder le bloc courant alors un bit 1 est envoyé. Sinon la valeur 0 est envoyée ainsi que l'identifiant du mode de prédiction. Dans le cas où l'information pour les blocs voisins n'est pas disponible (typiquement dans le cas du bord d'une frame), alors le mode "moyenne" (c) est considéré comme le plus probable.

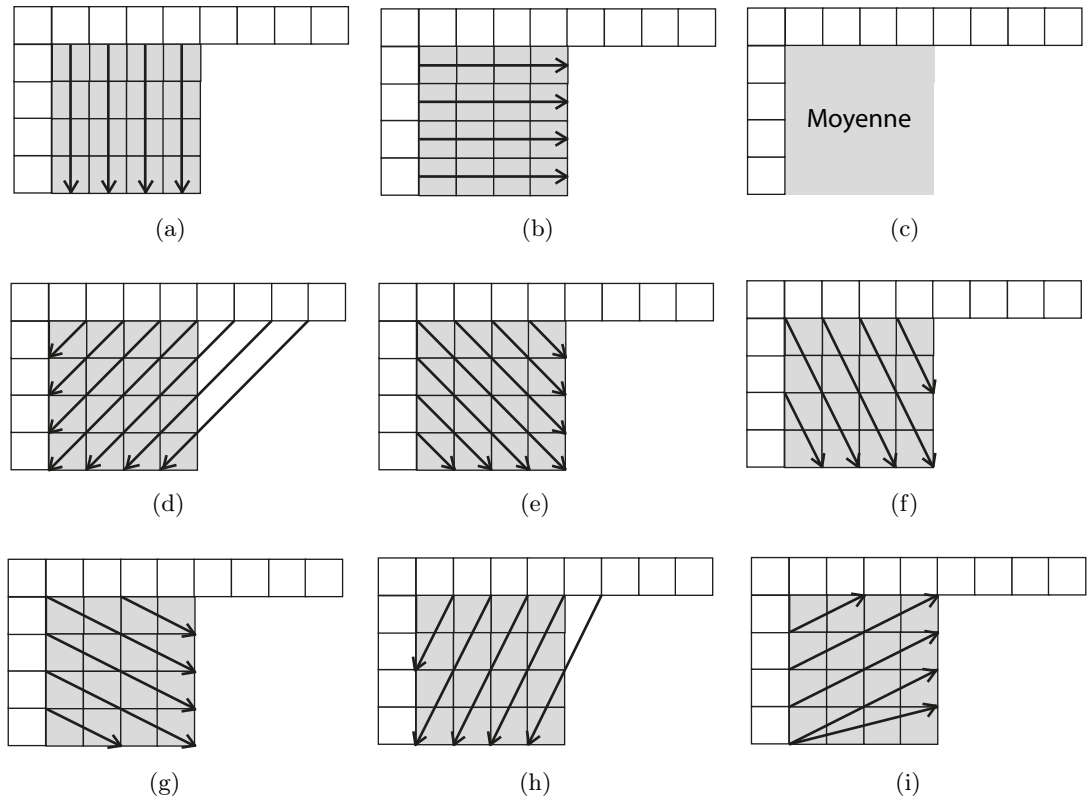


FIGURE II.1.21 – Les 9 modes de prédiction possibles pour des blocs de luminance de taille 4×4

Afin de bien montrer l'intéret d'un tel mécanisme, la figure II.1.23(a) présente une frame issue d'une séquence vidéo. La figure II.1.23(b) présente la frame prédite en utilisant le mécanisme d'intra-prédiction décrit ci-dessus. La frame résiduelle résultant de la différence entre la frame d'origine et la frame prédite est présentée sur la figure II.1.23(c). On se rend tout de suite compte

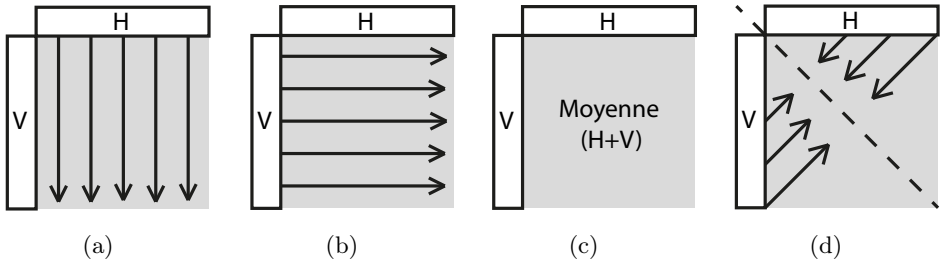


FIGURE II.1.22 – Les 4 modes de prédiction possibles pour des blocs de luminance de taille 16×16

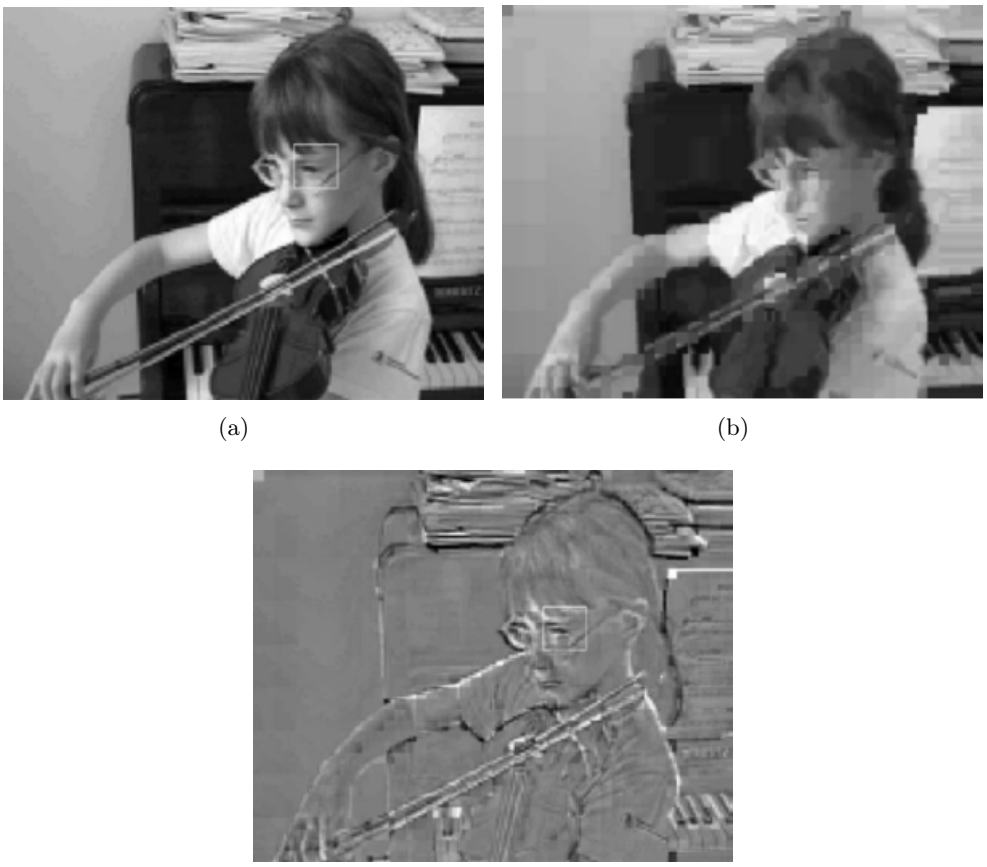
que la frame résiduelle contient moins d'information et donc produira de meilleurs résultats en terme de compression que la frame d'origine. L'utilisation du mécanisme d'intra-prédiction a donc permis de réduire de manière non négligeable la quantité d'informations nécessaires pour coder les *frames* I (qui constituaient jusqu'à H.264/AVC un goulot d'étranglement au niveau du débit), mais aussi une intra-prédiction pour les *frames* P et B (notamment lors de brusques changements de plans de caméra où l'inter-prédiction était impossible pour des *frames* P).

II.1.3.3.2 Filtrage anti-blocs

L'une des améliorations majeures qu'apporte de même H.264/AVC est le filtrage anti-blocs. Le problème majeur avec les schémas de compression basés sur la DCT réside dans le fait que lorsqu'on atteint des facteurs de quantification élevés, on commence à voir apparaître des carrés sur l'image. Ces artefacts bien connus dans le domaine de la compression sont généralement désignés par le terme "blocking artifacts". H.264/AVC propose un système de filtrage permettant de réduire de manière significative ces artefacts pour deux raisons distinctes : améliorer la restitution des vidéos encodées pendant le décodage, mais aussi optimiser le processus d'estimation de mouvement qui s'effectue à l'aide non pas des *frames* d'origine mais de *frames* décodées.

Ce filtrage s'effectue au niveau des bords de chaque bloc 4×4 constituant un macro-bloc (au nombre de 16 pour un macro-bloc de luminance et de 4 pour un macro-bloc de chrominance). Conformément à la figure II.1.24, on a donc 4 frontières verticales ((a), (b), (c) et (d)) ainsi que 4 frontières horizontales ((e), (f), (g), (h)) à filtrer pour un macro-bloc de luminance. De manière analogue, pour la chrominance, on a 2 frontières horizontales ((i) et (j)) et 2 frontières verticales ((k) et (l)) à filtrer.

Cette opération de filtrage s'effectue à l'aide d'un filtre 1D plus ou moins grand et permettant de modifier jusqu'à trois pixels de part et d'autre de la frontière. Ce filtrage est bien évidement adaptatif et est paramétré par deux notions : la "force" de la frontière ainsi que les gradients sur cette frontière. On détermine la "force" de la frontière selon plusieurs critères tel que le type de prédiction (intra ou inter) utilisée : en effet, les bords appartenant à des blocs intra-codés présentent généralement des distorsions. On retrouve ce problème dans le cas de l'inter-prédiction, lorsque les deux blocs adjacents n'ont pas la (ou les) même frame de référence ou des vecteurs de mouvements différents. L'autre paramètre est le calcul des gradients sur la frontière : grâce à ceux-ci, on va pouvoir faire la différence entre, par exemple, le contour d'un objet de la scène se situant à l'endroit même de la frontière et de réels artefacts de "blocking" liés à la quantification



(a) (b)

(c)

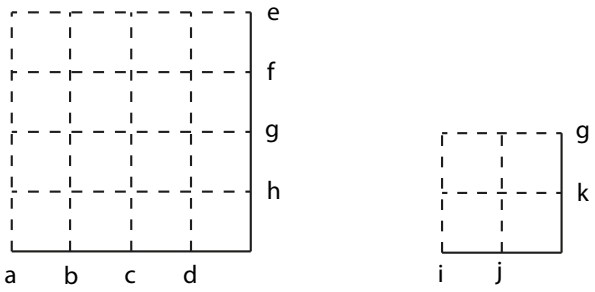
FIGURE II.1.23 – Processus d'intra-prédiction : (a) frame d'origine, (b) frame prédite et (c) frame résiduelle

des coefficients de la DCT. Tous ces paramètres vont ainsi définir une "force" de frontière. Plus la frontière va être considérée comme "forte", plus le filtrage va être important et influer sur un nombre plus important de pixels de part et d'autre de la frontière. La figure II.1.25 présente une frame vidéo reconstruite sans utilisation du filtrage anti-blocs (II.1.25(a)) et en l'utilisant (II.1.25(b)).

On constate bien que tous les blocs ont disparu et qu'à l'œil nu, la qualité de l'image semble bien meilleure.

II.1.3.3.3 Processus de transformation DCT et de quantification

Le standard H.264/AVC utilise un nouveau type de transformation pour l'application de la DCT et du processus de quantification afin d'optimiser ces deux processus. Dans les précédents standards, l'application de la DCT était réalisée en utilisant la formulation standard de la DCT 2D définie par l'équation II.1.14. Le problème avec cette implémentation de la DCT est qu'elle requiert des approximations pour le facteur contenant le cosinus. Or ces approximations peuvent



(a) (b)

FIGURE II.1.24 – Les différentes frontières à filtrer au sein d'un macro-bloc de luminance (a), et de chrominance (b)

(a) (b)

FIGURE II.1.25 – Exemple de frame reconstruite sans filtre anti-blocs (a) et avec filtre anti-blocs (b)

différer d'une implémentation à l'autre de l'encodeur et du décodeur, ceci pouvant entraîner des discordances lors de la restitution. De plus, l'application de la DCT sous cette forme requiert des multiplications sur des flottants qui sont des opérations relativement coûteuses pour le processeur. Le but de cette nouvelle formulation de la DCT va donc être de standardiser le processus de transformation et de quantification, mais aussi de pouvoir réaliser le calcul de celle-ci en se passant d'opérations sur les flottants et de limiter au maximum le nombre d'opérations coûteuses sur les entiers (multiplications et divisions). La figure II.1.26 illustre le processus de déduction de la nouvelle formulation d'application de la DCT à partir de l'approche traditionnelle (sur la première ligne).

Sur la ligne du haut, on peut voir l'approche de base généralement appliquée lors d'une transformation DCT suivie d'une étape de quantification.

La première étape du développement a été de séparer l'application de la DCT (4×4 et 8×8) qui utilise des coefficients flottants en deux matrices C_f et S_f. Ces deux matrices correspondent respectivement à une matrice de transformation de base et à une matrice d'échelle. La première matrice C_f correspond à une approximation de la DCT et contient uniquement des coefficients appartenant à $-2, -1, 1, 2$, permettant ainsi d'effectuer la multiplication du bloc à transformer en n'utilisant que des additions et des décalages de bits. La matrice S_f permet quant à elle de

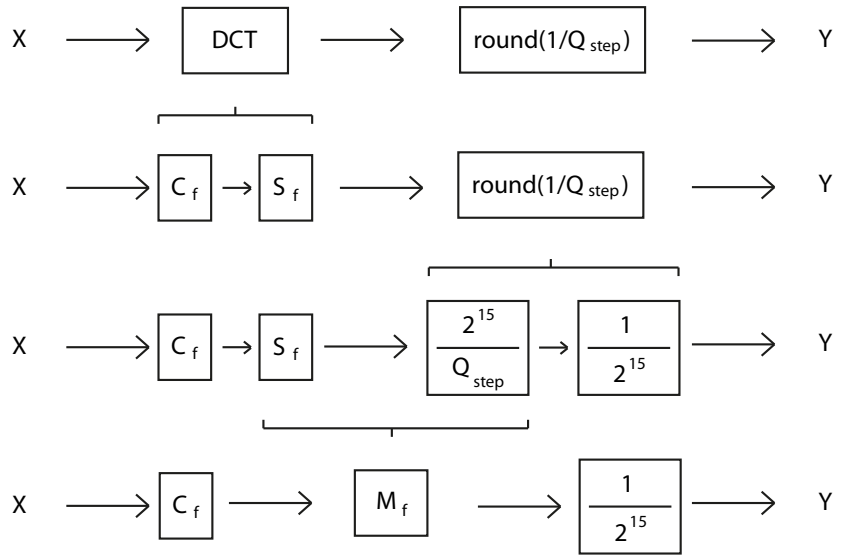


FIGURE II.1.26 – Nouveau développement du processus d'application de la DCT et de quantification

récupérer la propriété orthonormale de la matrice d'origine de la DCT.

La seconde étape consiste à séparer le facteur de quantification en deux parties : la première contenant le pas de quantification et multiplié par 2^{15}, la deuxième contenant uniquement la division par 2^{15} pour retrouver la valeur d'origine.

La troisième étape combine la matrice d'échelle S_f précédemment définie et le facteur $\frac{2^{15}}{Q_{step}}$ (Q_{step} étant le pas de quantification associé au paramètre de quantification QP) afin d'obtenir une seule matrice M_f utilisant elle aussi uniquement des coefficients entiers toujours pour optimiser le coût algorithmique du processus.

Ainsi le processus de transformation/quantification sur un bloc 4×4 noté X est donné par l'équation II.1.25.

$$Y = \text{round}\left([C_f] \cdot [X] \cdot [C_f^T] \bullet M_f \cdot \frac{1}{2^{15}}\right) \tag{II.1.25}$$

où :
– $\cdot$ représente le produit matriciel.
– $\bullet$ représente le produit terme à terme.
– C_f^T représente la transposée de C_f.

La transformation inverse, déduite d'après un raisonnement similaire est donnée par l'équa-

tion II.1.26.

$$Z = \text{round}\left([C_i^T] \cdot [Y \bullet V_i] \cdot [C_i] \cdot \frac{1}{2^6}\right) \tag{II.1.26}$$

où :
- C_i représente la matrice d'approximation de la DCT inverse.
- C_i^T représente la transposée de C_i.
- V_i permet de retrouver M_f par la relation :

$$M_f = \text{round}\left(\frac{S_i \cdot S_f \cdot 2^{21}}{V_i}\right)$$

avec S_i la matrice d'échelle associée à C_i.

Les matrices M_f et V_i ne contiennent que trois coefficients distincts dépendant du paramètre de quantification QP. Entre deux valeurs successives de QP, le pas de quantification associé (Q_{step}) évolue d'un facteur $\sqrt[6]{2}$. On ne stocke donc que les coefficients distincts des matrices M_f et V_i associées aux six premières valeurs de QP (de 0 à 5). Pour des valeurs de QP supérieures à 5, les coefficients sont multipliés par le facteur $2^{\lfloor QP/6 \rfloor}$.

De manière générale, les processus d'application de la DCT et de la quantification ont été optimisés (ainsi que la procédure inverse) : plus aucune opération sur des flottants n'est utilisée et la majorité des opérations s'effectue à l'aide d'additions et de décalages de bits. De façon analogue, le processus de transformation pour des blocs de taille 8×8 est déduit à partir du même mécanisme que celui décrit ci-dessus.

II.1.3.3.4 CABAC : Context-Adaptative Binary Arithmetic Coding

H.264/AVC propose un nouvel algorithme pour réaliser l'étape du codage entropique aussi bien des informations propres au flux vidéo (telles que le type de macro-bloc, ou le mode de prédiction utilisé ...), que des données image résiduelles : CABAC (Context-Adaptative Binary Arithmetic Coding). L'algorithme CABAC mis au point par Marpe *et al.* (2003), est un codeur arithmétique adaptatif binaire : en effet, celui-ci couple l'efficacité du codage arithmétique et celle des méthodes basées sur la modélisation de contextes et permet d'obtenir des résultats significativement meilleurs que les méthodes de codage entropique utilisées dans les précédents standards comme CAVLC (Context-Adaptative Variable Length Coding) (Marpe *et al.*, Richardson (2010)). Pour cela l'algorithme procède en trois étapes majeures :

- Une étape de binarisation durant laquelle toute information non binaire (telle que le type d'un macro-bloc ou d'un vecteur de déplacement) est transformée en une chaîne binaire (généralement désignée par le terme *bin string*) à l'aide de schémas de binarisation précis.

– Une étape de modélisation de contexte durant laquelle chaque bit constituant la chaîne binaire générée durant l'étape précédente va être associé à un modèle probabiliste déterminé à partir d'un template.
– Une étape de codage arithmétique durant laquelle le bit actuel est codé à l'aide des statistiques associées au modèle probabiliste précédemment déterminé.

a) Processus de binarisation

L'algorithme CABAC est un codeur arithmétique binaire, ce qui signifie que l'alphabet de symboles Ω est réduit à deux symboles : 0 et 1. Pourtant, des informations de types non binaires tels que les flags permettant de renseigner sur le type de bloc en cours de traitement ou autre, doivent aussi être codées. Pour pouvoir être prise en charge par le codeur arithmétique CABAC, toute information non binaire devra tout d'abord être transformée en une chaîne binaire. Pour cela, CABAC utilise plusieurs schémas de binarisation bien distincts et spécifiques au type d'information à coder.

b) Modélisation de contexte

Une fois l'étape de binarisation terminée (durant laquelle les différents éléments de syntaxe propres à H.264/AVC ont été mis sous forme d'une chaîne binaire $B = b_0 b_1 ... bn$), chacun de ces n bits doit maintenant être envoyé au codeur arithmétique. Comme nous l'avons déjà précisé dans la section II.1.1.3, les méthodes basées sur la modélisation de contexte utilisent un template $\mathbf{T}$ (généralement constitué de symboles précédemment codés) afin d'associer un contexte $\mathbf{C}$ (et donc un modèle probabiliste) pour coder le symbole courant x. CABAC dispose de 399 contextes différents, chacun spécifique à l'élément de syntaxe à coder ainsi qu'au type du slice courant (I/SI, P/SP ou B). Chaque contexte est identifié par un index unique γ et est associé à un modèle probabiliste donné grâce à deux paramètres : un état de probabilité σ_γ ainsi que la valeur binaire du symbole le plus probable ϖ_γ. Ces deux paramètres permettront par la suite d'estimer la probabilité à utiliser lors du codage arithmétique à proprement parler (voir le paragraphe suivant). La table II.1.10 présente les différents éléments de syntaxe propres à H.264/AVC et les différentes plages de contextes associées à ceux-ci. La troisième colonne de la table précise le type de méthode utilisée pour associer au bit b_i son contexte $\mathbf{C}$.

Suivant le type d'information à coder, CABAC dispose de quatre méthodes pour déterminer le contexte associé au bit courant b_i :

– Le premier schéma de détermination de contexte repose sur un template $\mathbf{T}$ constitué de deux symboles précédemment codés et appartenant au voisinage du symbole en cours de codage. De manière générale, on associe au bit b_i le contexte $\mathbf{T} = \{b_{i_t}, b_{i_l}\}$ où b_{i_t} et b_{i_l} désignent respectivement les $i^{\text{ième}}$ bits des symboles précédemment codés et situés au-dessus et à gauche de l'élément courant. Ce type de détermination de contexte est utilisé pour les informations pouvant présenter une corrélation spatiale telles que les vecteurs de mouvement ou le mode de prédiction intra-frame lorsque celui-ci est utilisé.
– Le deuxième schéma est spécifique aux symboles binarisés à l'aide d'arbre de Huffman. Dans ce cas, le contexte $\mathbf{T}$ est constitué des bits $b_0, b_1, b_2, ..., b_{i-1}$ du symbole courant,

bits correspondant au parcours de l'arbre où chaque noeud est associé à une probabilité spécifique.

– Les troisième et quatrième schémas sont tous les deux dédiés au codage de l'information résiduelle dans chaque bloc de l'image. Le premier est utilisé pour coder l'information binaire associée à la carte de "significativité" du bloc courant. Le deuxième, quant à lui, se base sur le nombre de coefficients non nuls du bloc, déjà codés, pour associer un contexte à chaque bit b_i.

Elément de syntaxe	Type de slice			Méthode d'association de contexte
	I/SI	P/SP	B	
mb_type	0/3-10	14-20	27-35	2
mb_skip_flag		11-13	24-26	1
sub_mb_type		21-23	36-390	2
mvd (horizontal)		40-46	40-46	1
mvd (vertical)		47-53	47-53	1
ref_idx		54-59	54-59	1
mb_qp_delta	60-63			
intra_chroma_pred_mode	64-67			1
prev_intra4x4_pred_mode_flag	68			
rem_intra4x4_pred_mode	69			
mb_field_decoding_flag	70-72			1
coded_block_pattern	73-84			1
coded_block_flag	85-104			1
significant_coeff_flag	105-165, 277-337			3
last_significant_coeff_flag	166-226, 338-398			3
coeff_abs_level_minus1	227-275			4
end_of_slice_flag	276			

TABLEAU II.1.10 – Contextes associés aux différents éléments de syntaxe du flux H.264/AVC

Une fois le contexte déterminé, le bit b_i en cours de codage et les deux paramètres σ_γ et ϖ_{gamma} sont transmis au codeur arithmétique.

c) Codage arithmétique

La dernière étape de CABAC consiste en l'étape du codage arithmétique à proprement parler. La section II.1.1.1.3 présente le principe du codage arithmétique et de la subdivision récursive d'intervalles plus en détails pour réaliser la dernière étape de codage arithmétique. Dans le cas d'un codeur arithmétique binaire, l'alphabet ne compte que deux symboles 0 et 1 dont les probabilités d'apparition sont données respectivement par p_0 et p_1. Dans le cas général où $p_0 \neq p_1$, le symbole ayant la probabilité d'apparition la plus grande est désigné par le terme MPS (Most Probable Symbol) tandis que l'autre est désigné par le terme LPS (Least Probable Symbol). Si l'on considère l'intervalle courant I de taille R et de borne inférieure L, on définit alors $R_{LPS} = p_{LPS} \times R$. Il faut noter que le choix de se baser sur le LPS est totalement arbitraire, on aurait tout aussi bien pu utiliser le MPS. A la lecture du bit courant, deux cas de figure peuvent alors se présenter :

- Soit le bit lu correspond au MPS, dans ce cas le nouvel intervalle I' à pour taille $R' = R - R_{LPS}$ et on garde la même borne inférieure L.
- Soit le bit lu correspond au LPS et dans ce cas, on considère alors I' comme nouvel intervalle courant ayant pour taille $R' = R_{LPS}$ et ayant comme borne inférieure $L' = L + R - R_{LPS}$.

Le problème majeur lors de l'implémentation pratique de l'algorithme réside dans l'évaluation de la variable R_{LPS} qui requiert une multiplication. Cette multiplication est un processus relativement coûteux aussi bien au niveau software que hardware et peut constituer un goulot d'étranglement. C'est sur ce point que CABAC est réellement innovant : plutôt que de calculer directement R_{LPS}, celui-ci va optimiser cette étape via l'utilisation de LUT (Look-Up Tables).

Lorsque le codeur arithmétique reçoit un bit b_i à coder, il reçoit aussi les deux paramètres σ_γ et ϖ_γ spécifiques au contexte γ précédemment associé à b_i. Ainsi à partir de ces deux paramètres, on peut alors directement déterminer la probabilité p_{LPS} à l'aide de l'état σ_γ mais aussi réaliser la mise à jour du modèle probabiliste associé au contexte γ en appliquant la règle de transition associée à l'état.

Chapitre II.2

MICA : Multiview Image Compression Algorithm

Dans cette première approche, nous nous sommes fixés pour but de mettre en place un algorithme de compression temps réel, adapté à notre système de capture multi-vues et générant le moins de distorsions possibles sur le signal d'entrée. Ces trois contraintes ont conduit au développement de l'algorithme MICA (Multiview Image Compression Algorithm) qui repose sur l'exploitation de la redondance inter-vues spécifique à ce type de média. La mise en évidence de cette redondance fera l'objet de la première partie de cette section, puis nous détaillerons le fonctionnement de l'algorithme et enfin, nous présenterons et commenterons les résultats obtenus par notre approche. MICA a fait l'objet en 2010 d'une publication (Battin *et al.* (2010)) dans le cadre de la conférence Stereoscopic Displays and Application XXI.

II.2.1 Caractérisation de la double redondance

Notre système de capture génère 8 flux vidéos parallèles pouvant être représentés sous la forme d'une matrice d'images $[V_{i,t}]$ où $V_{i,t}$ désigne la frame provenant de la $i^{\text{ème}}$ caméra à l'instant t. De telles données sont caractérisées par une double redondance temporelle et spatiale. La première est généralement exploitée à l'aide de techniques de compensation de mouvement sur les macro-blocs de l'image. Pour la redondance spatiale, plusieurs solutions sont possibles : soit des approches basées sur la compensation de disparités (telles que le standard H.264/MVC Merkle *et al.* (2006, 2007), voir section III.1.2.2.1), soit des approches utilisant des cartes de profondeur (Yoon *et al.* (2007), Jantet *et al.* (2009), voir section III.1.2.1.2). De telles méthodes possèdent un coût algorithmique élevé qui les rendent plus ou moins viables pour des applications temps réel.

Nous choisissons donc de commencer notre étude en considérant l'approche la plus intuitive, en analysant l'histogramme des différences entre une frame $V_{i,t}$ et une frame $V_{i+1,t}$. Pour cela, nous utilisons un ensemble de plusieurs images multi-vues. La figure II.2.1 décrit les courbes

moyennes d'amplitude des différences pour le domaine spatial.

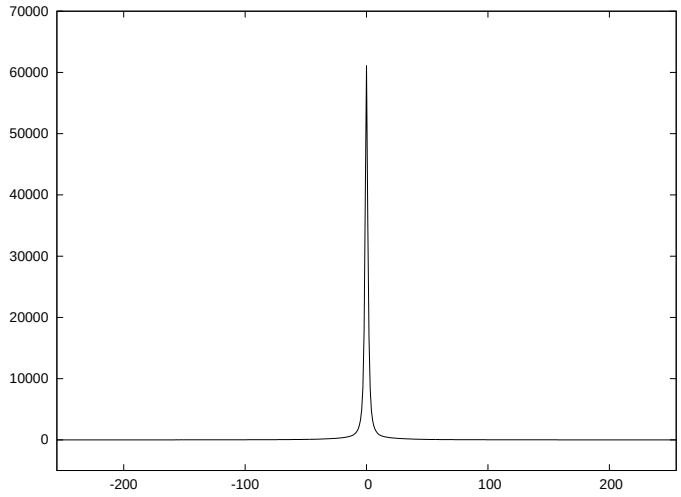


FIGURE II.2.1 – Courbe moyenne d'amplitude des différences spatiales.

D'après la figure II.2.1, on peut faire ressortir les deux propriétés suivantes :

- Plus de 90% des pixels sont situés dans l'intervalle $[-50; 50]$
- Plus de 75% des pixels sont situés dans l'intervalle $[-5; 5]$

Ces deux observations nous ont conduit à penser que l'utilisation d'un code à longueur variable pourrait être relativement efficace et raisonnable vis-à-vis du coût algorithmique. De plus, ce code devra être construit en prenant en compte la distribution des valeurs de différences afin d'être le plus optimal possible (un peu à la manière de l'algorithme FELICS, détaillé en section II.1.1.3.1). Pour cela, examinons de manière plus précise la distribution des différentes valeurs.

Pour une certaine valeur de pixel A d'une image de différence ($A \in [-255, 255]$), la probabilité $p(A)$ pour que le pixel courant ait la valeur A est donnée par la formule II.2.1 :

$$p(A) = \frac{nb(A)}{nb(total)} \tag{II.2.1}$$

où $nb(A)$ définit le nombre d'occurrences de la valeur A et $nb(total)$ est le nombre total de pixels dans l'image de différence. Conformément à l'histogramme II.2.1, plus A est éloigné de la valeur 0, plus la longueur du mot binaire associé devra être longue. Le code binaire associé à la valeur A devra avoir la même longueur que celui associé à la valeur $-A$ car on peut voir, toujours d'après la figure II.2.1, que $p(A) \approx p(-A)$.

Dans la section suivante nous allons présenter le schéma de compression ainsi mis au point.

II.2.2 Description de l'algorithme

Dans cette section, nous présentons en détail l'algorithme MICA (Multiview Image Compression Algorithm) dédié à la compression des images multi-vues. Celui-ci se compose de trois blocs majeurs (voir figure II.2.2) :

(1) Conversion colorimétrique.
(2) Génération des images de différence.
(3) Encodage des images de différence et des images de référence.

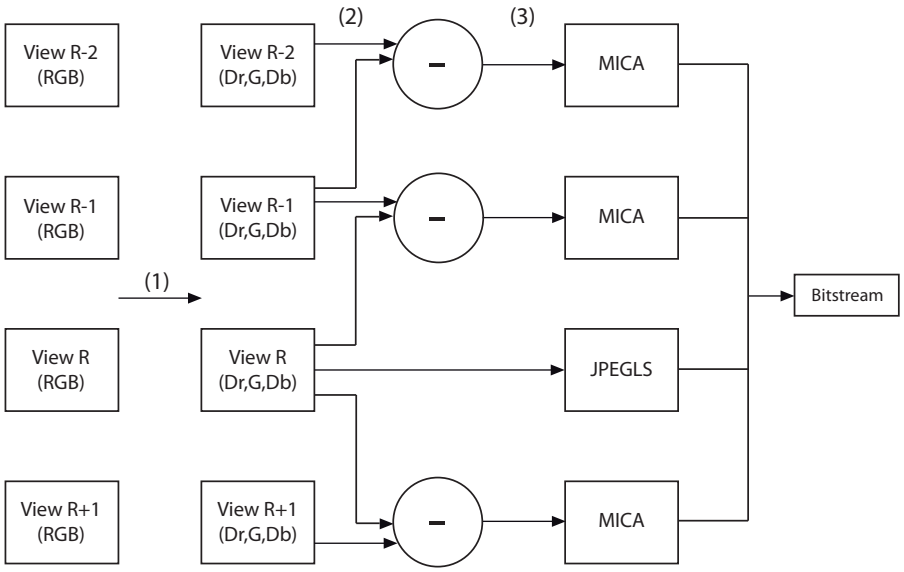


FIGURE II.2.2 – Fonctionnement général de MICA.

Pour notre approche, nous décidons tout d'abord de considérer la redondance spatiale et de concevoir l'algorithme pour la compression d'images multi-vues, puis d'étendre celui-ci afin de prendre en compte la redondance temporelle des séquences multi-vues. Nous allons ci-après exposer les différents blocs de traitement constituant l'algorithme.

II.2.2.1 Conversion colorimétrique

Nous commençons par effectuer une conversion colorimétrique de l'espace R,G,B vers un espace Dr, G, Db en utilisant les relations suivantes :

$$\begin{cases} Dr = G - R \\ Db = G - B \end{cases}$$

Dr (resp. Db) correspond en fait à la différence entre la composante verte et la composante

rouge (resp. bleue) de l'image. Cette opération va nous permettre de réduire en amont la corrélation spectrale entre les différents canaux R,G et B (Zhang *et al.* (2005)) et de générer des images contenant des variations plus faibles que sur les composantes d'origine. Étant donné le fait que nous travaillons sur des images d'entrée bayérisées (BE et Company (1975)), la relation ci-dessus requiert l'interpolation de la composante verte sur les localisations associées aux composantes rouge et bleue.

II.2.2.2 Génération et codage des images de différence

Comme nous l'avons indiqué dans le début de ce chapitre, l'objectif principal de MICA est de permettre une compression qualitative des séquences multi-vues en codant de manière différente les zones de l'image soumises à l'effet parallaxe (critiques pour le rendu auto-stéréoscopique) de celles étant statiques d'une vue à l'autre. L'utilisation des images de différence nous permet de localiser ces zones (correspondant à des coefficients de haute amplitude) et ainsi d'adapter notre système de codage afin de minimiser la distorsion du signal à ces endroits de l'image.

Considérons le domaine de définition des images de différence ($[-255; 255]$). Nous décidons de restreindre le sous-intervalle associé aux zones critiques conformément au découpage suivant :

- l'intervalle $[-255; -34[\cup]34; 255]$ correspond aux zones cruciales soumises à l'effet parallaxe
- l'intervalle $[-34; 34]$ correspond aux zones statiques de l'image

Pour les pixels appartenant au premier intervalle, nous désirons avoir le moins de déformations possibles sur leurs valeurs. Nous choisissons donc de les coder en utilisant un code ASCII modifié : le bit de poids faible (Least Significant Bit) est utilisé en tant que bit de signe, les 7 autres restant inchangés. Cela va permettre de minimiser l'erreur possible à une valeur de 1 pour ces pixels. L'autre intervalle (correspondant aux zones statiques de l'image), peut quant à lui être compressé de manière plus efficace sans altérer la vision auto-stéréoscopique. L'intervalle $[-34; 34]$ est divisé en plusieurs sous-intervalles, chacun étant associé à un mot binaire spécifique (voir table II.2.1).

Intervalle	Mot binaire
$[-34; -29]$	1111110
$[-29; -24]$	1111100
$[-24; -19]$	111100
$[-19; -14]$	11100
$[-14; -9]$	1100
$[-9; -4]$	100
$[-4; 4]$	0
$[4; 9]$	101
$[9; 14]$	1101
$[14; 19]$	11101
$[19; 24]$	111101
$[24; 29]$	1111101
$[29; 34]$	1111111

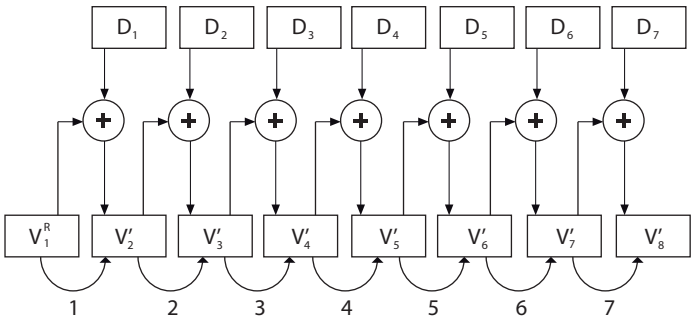
Tableau II.2.1 – Codes binaires associés à chaque sous-intervalle.

On peut remarquer que les mots binaires présentés dans la table II.2.1 sont conformes à la distribution des valeurs de différences et disposent des propriétés suivantes :

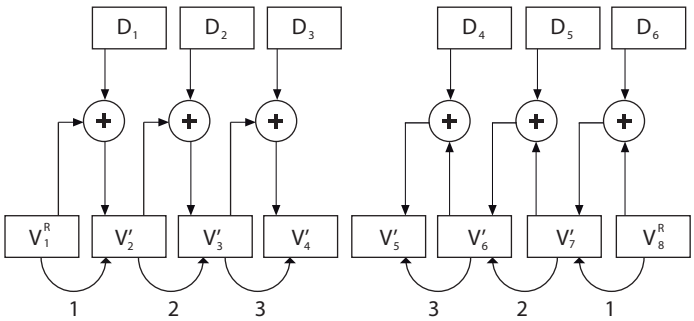
- Le code associé à l'intervalle $[-X, 5-X[$ est de même longueur que celui associé à l'intervalle $]X-5, X]$ (ils ont approximativement la même probabilité d'apparition).
- Plus l'intervalle est éloigné de la valeur 0, plus la séquence de bits associé à celui-ci est longue.
- Le mot binaire le plus court (0) est réservé à l'intervalle central du domaine de définition correspondant à 75% des pixels de l'image de différence.

II.2.2.3 Codage des images de référence

Nous allons maintenant présenter la notion d'image de référence. L'utilisation d'images de différence nécessite l'emploi d'une ou plusieurs images de référence afin de reconstruire l'image multi-vues lors du processus de décompression. Afin de préserver au maximum la qualité de l'image, nous choisissons de compresser celle-ci sans perte grâce à l'algorithme JPEG-LS (Weinberger *et al.* (1996), ISO (c), voir la section II.3.1). Il faut maintenant définir le nombre d'images de référence à utiliser mais aussi leur positionnement parmi les N vues. Pour cela on considère la reconstruction des vues d'origine à partir des images de référence et de différence. La figure ci-dessous (figure II.2.3) illustre le processus de reconstruction pas à pas en utilisant deux scénarios possibles.



(a)



(b)

FIGURE II.2.3 – Scénarios de reconstruction : (a) une image de référence sur la vue 1 (7 étapes de reconstruction), (b) deux images de référence situées sur les vues 1 et 8 (3 étapes de reconstruction).

Le problème principal réside dans le fait que les images de différence sont compressées à l'aide d'un algorithme générant des pertes : une vue reconstruite V' contient des erreurs qui vont être propagées lors de la prochaine étape de reconstruction. Ainsi, un nombre élevé d'étapes de reconstruction (voir figure II.2.3) va produire des erreurs de plus en plus grandes d'une étape à l'autre. Pour limiter ce nombre d'étapes et éviter une accumulation trop forte des erreurs, nous choisissons d'utiliser deux vues de référence situées sur les vues 3 et 6. De cette manière, le nombre maximal d'étapes de reconstruction successives est réduit à 2 (voir la figure II.2.4).

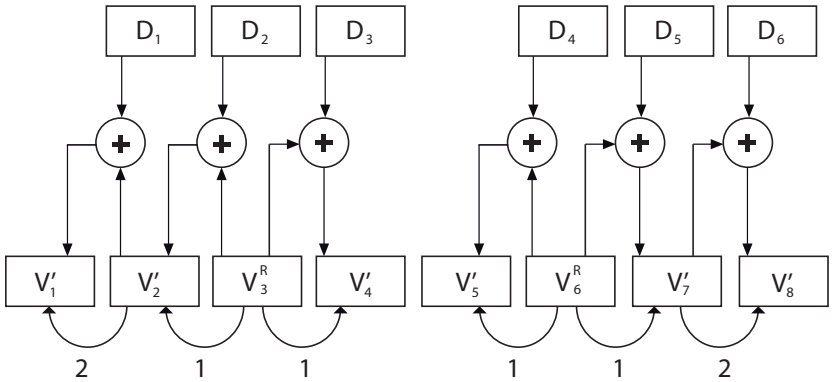


FIGURE II.2.4 – Notre schéma actuel de reconstruction.

II.2.2.4 Généralisation au temps

Afin de pouvoir prendre en charge la compression de séquences multi-vues, nous avons étendu le principe des images de différence sur le temps pour aboutir au fonctionnement décrit par la figure II.2.5.

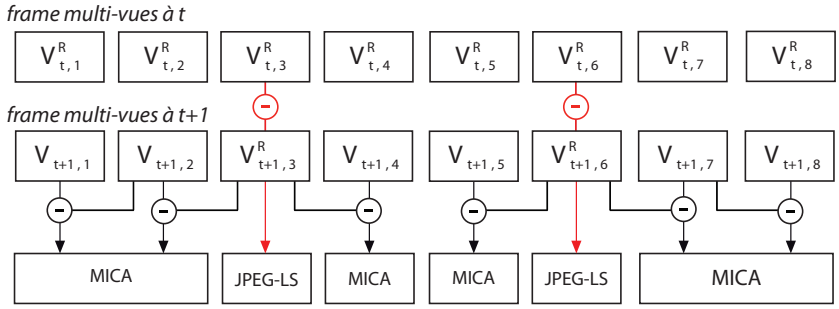


FIGURE II.2.5 – Fonctionnement de l'algorithme généralisé au temps.

La prise en compte de la redondance temporelle est réalisée uniquement sur les images de référence. Pour cela, on génère l'image de différence à partir de deux frames de référence successives $V^R_{t,i}$ et $V^R_{t+1,i}$ $(1 \leq i < N)$. Cette image, qui a pour domaine de définition $[-255; 255]$, doit être compressée sans perte pour éviter toute génération d'erreurs le long de l'axe temporel. Toutefois, l'implémentation standard de l'algorithme JPEG-LS permet de gérer des valeurs de pixels codées sur 8 bits pour chaque composante. Une phase de modification de l'algorithme JPEG-LS a été nécessaire : nous avons étendu l'alphabet afin de permettre la gestion de valeurs codées sur 9 bits (correspondant à une plage contenant 512 valeurs possibles).

II.2.2.5 Implémentation pratique

Dans cette section, nous détaillons nos choix en matière d'implémentation pour l'algorithme MICA. Un des points forts de MICA réside en sa faible complexité algorithmique mais aussi au fait qu'il peut faire l'objet d'une implémentation multi-threads afin de permettre l'encodage temps réel de séquences multi-vues.

Pour cela, nous avons choisi de travailler avec la classe de threads fournies par la librairie Qt (basée threads WIN32 sous windows, ou sur les pthreads dans le cas d'un système de type Linux).

Deux types de threads sont ainsi définis :

- Les threads MICA : chacun de ces threads travaille sur 1 ou 2 vues en entrée et exécute successivement les 3 blocs de traitement définis dans la section II.2.2. Les threads MICA sont destinés à la génération et à la compression des images de différence.
- Les threads JPEG-LS : ces threads, quant à eux, permettent la compression sans perte des images de référence (ou des images de différence temporelle si l'on utilise l'algorithme généralisé au temps).

La figure II.2.6 décrit la répartition des différents threads sur les 8 vues pour le processus de compression.

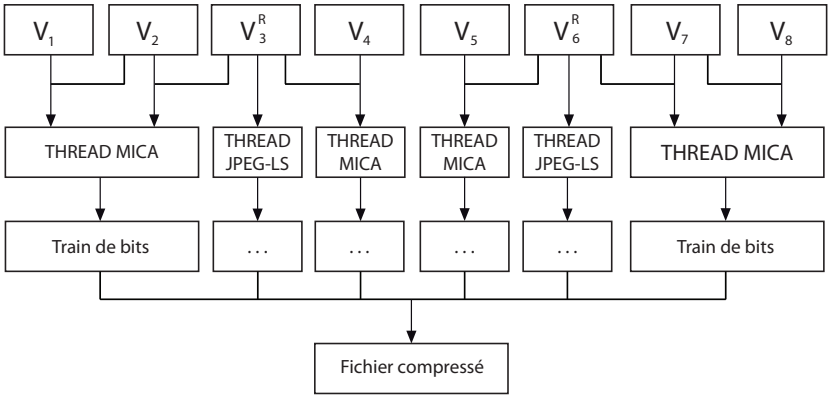


FIGURE II.2.6 – Répartition des différents threads durant le processus de compression.

II.2.3 Résultats et discussions

Nous allons maintenant présenter les divers résultats obtenus, aussi bien en terme de compression qu'en terme de temps. Afin de simuler le comportement de l'Octocam (qui délivre en sortie des images bayérisées), nous appliquons au préalable une étape de bayérisation sur nos différents jeux d'images multi-vues. La table II.2.2 présente les résultats obtenus en terme de compression et de distorsion sur les différents jeux de données. Chaque séquence multi-vues est

composée de 8 vues, chacune comprenant 25 frames temporelles. Pour mesurer la distorsion par rapport aux images sources, nous utilisons la moyenne du PSNR (Peak Signal to Noise Ratio, dont la formule est donnée par l'équation II.2.2) sur toute la séquence. MICA étant un algorithme de compression temps réel, on y retrouve aussi les temps moyens de compression et de décompression d'une frame multi-vues à un instant t (les mesures de temps ont été effectuées sur un ordinateur doté d'un processeur i7-950).

$$\text{PSNR} = 10 \times \log_{10}\left(\frac{255^2}{EQM}\right) \tag{II.2.2}$$

où EQM est l'Erreur Quadratique Moyenne entre l'image source et l'image traitée.

Séquence	Ratio (bits/pixel)	PSNR (dB)	Compression (ms)	Décompression (ms)
Cartes	1.65	41.91	22	31
Pinceau	1.05	45.20	17	22
Poupées	1.89	41.07	27	34
Rose	2.22	40.04	28	39
Statue	1.63	39.67	25	26

TABLEAU II.2.2 – Ratios de compression, PSNR et mesures des temps pour les différentes séquences.

Nous avons aussi choisi de comparer MICA à l'algorithme JPEG (implémentation de la librairie GFL SDK [1]). Pour la compression JPEG, nous ne pouvons pas laisser les images constituant chaque jeu multi-vues sous leur forme originale propre aux images bayérisées. Deux pixels voisins (horizontaux et verticaux) appartiennent à des composantes chromatiques différentes et la corrélation spatiale au sein de l'image est inexistante. Afin de remédier à ce problème, nous choisissons de séparer les différentes composantes chromatiques selon le schéma décrit par la figure II.2.7. Ensuite, nous compressons successivement les N différentes images constituant chaque jeu multi-vues de sorte que la taille totale des fichiers compressés soit égale (à quelques Ko près) à celle obtenue avec l'algorithme MICA.

(a) Avant séparation (b) Après séparation

FIGURE II.2.7 – Pré-traitement effectué pour la compression JPEG.

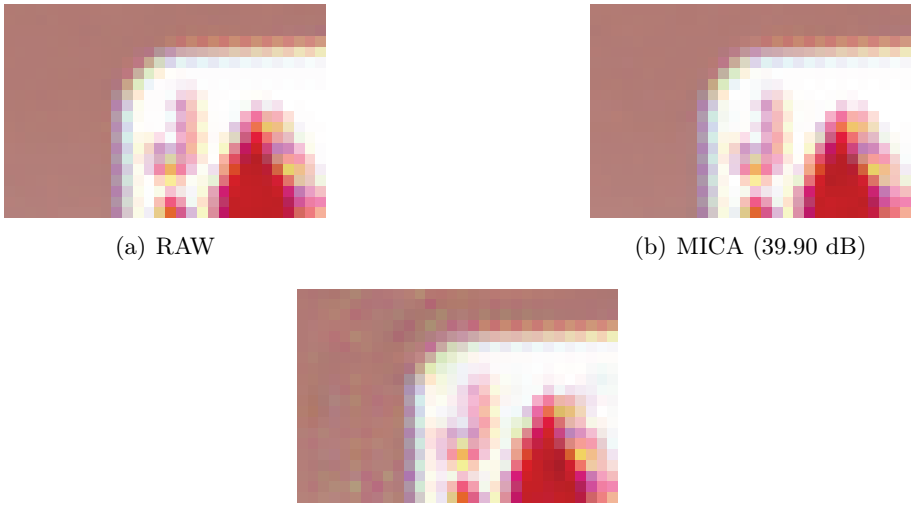
La table II.2.3 présente les différents résultats obtenus avec MICA et l'algorithme JPEG (les chiffres présents dans les deux dernières colonnes correspondent au PSNR moyen sur les 8 vues).

1. http ://www.xnview.com/fr/gfl.html

	Taille du fichier (Ko)		PSNR (dB)	
Séquence	MICA	JPEG	MICA	JPEG
Cartes	472	473	41.46	32.50
Pinceau	228	229	43.51	44.50
Poupées	518	525	40.51	43.37
Rose	348	346	40.79	42.34
Statue	633	624	38.94	37.83

TABLEAU II.2.3 – Résultats obtenus avec MICA et comparés à JPEG à taille de fichier égale.

Afin de permettre une comparaison aussi bien objective (via le PSNR) que subjective, les figures II.2.8, II.2.9 et II.2.10 présentent des portions d'images sur différents jeux d'images multi-vues. Nous présentons des zones uniquement issues de la première vue où MICA génère le plus de distorsions à cause de la propagation d'erreurs, nous plaçant ainsi dans le cadre le plus défavorable pour MICA.



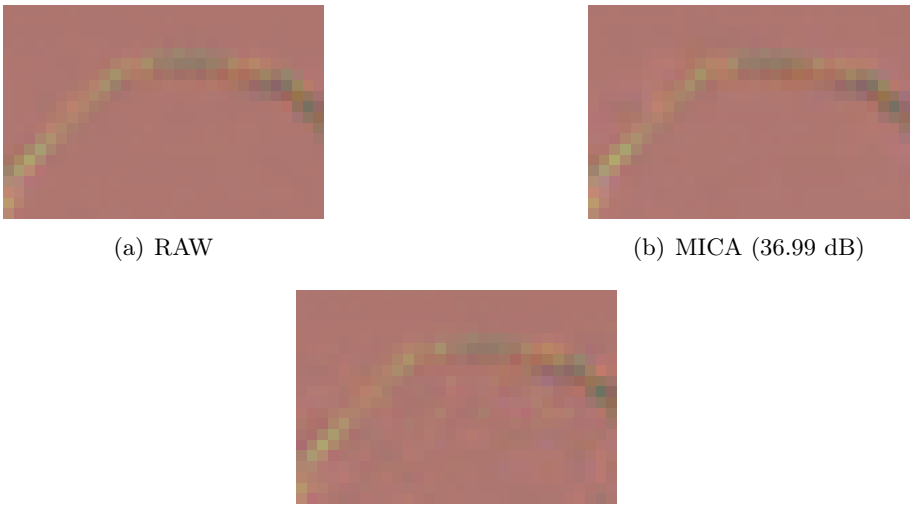
(a) RAW (b) MICA (39.90 dB)

(c) JPEG (32.52 dB)

FIGURE II.2.8 – Comparaison subjective sur la séquence "Cartes".

D'après les différents résultats présentés ci-dessus, on peut extraire plusieurs informations. D'après la table II.2.3, on peut voir que pour les séquences "Cartes" et "Statue", MICA permet une compression plus efficace (en terme de qualité d'image) que JPEG à taille de fichier égale. Pour les autres séquences, celui-ci est moins performant que l'algorithme JPEG.

Toutefois, MICA remplit quand même son objectif de base à savoir permettre une compression adaptative : pour les zones statiques d'une vue à l'autre (non soumises à l'effet parallaxe), MICA autorise une distorsion relativement importante du signal d'entrée. Par contre, dans les zones critiques des différentes vues extraites grâce au principe des images de différence, zones cette fois-ci soumises à l'effet parallaxe et importantes pour le rendu auto-stéréoscopique, MICA permet une conservation des contours supérieure à celle de l'algorithme JPEG et ce même lorsque le PSNR obtenu avec l'algorithme JPEG est supérieur (voir les figures II.2.8, II.2.9 et II.2.10).



(a) RAW (b) MICA (36.99 dB)

(c) JPEG (37.83 dB)

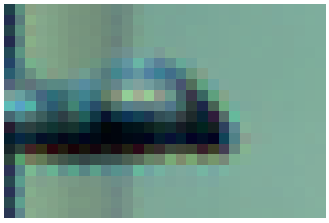
FIGURE II.2.9 – Comparaison subjective sur la séquence “Statue”.

Cependant, l’approche MICA présente quand même un certain nombre de défauts, à savoir :

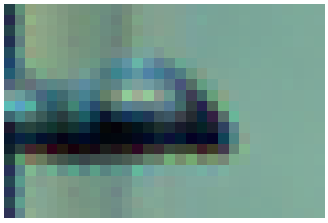
- MICA ne dispose pas d’un facteur de qualité permettant de faire varier la taille du fichier compressé. Ceci peut constituer un problème critique pour des applications telles que la télé-présence où l’on dispose d’une bande passante qui varie généralement dans le temps.

- MICA ne permet pas de s’adapter à la géométrie de capture multi-vue. En effet, il suffit que la distance inter-caméra soit plus importante pour que les images de différence contiennent beaucoup plus de pixels à forte amplitude et que la taille du fichier compressé augmente de façon critique. Cet algorithme est donc fortement dépendant du système de capture développé par 3DTV Solutions.

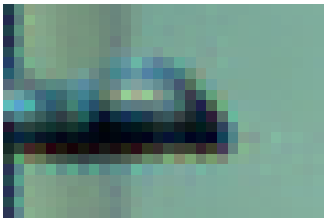
En conclusion, notre algorithme MICA présente des résultats intéressants mais pour être viable, il doit être utilisé sur l’Octocam pour la compression d’images fixes. Afin de satisfaire la première remarque ci-dessus (à savoir disposer d’un facteur de qualité), nous devrons donc nous tourner vers des méthodes de compression avec pertes plus classiques et contenant une étape de quantification. Pour pallier au deuxième défaut, la méthode devra prendre en compte la géométrie du système de capture afin de produire des résultats satisfaisants.



(a) RAW



(b) MICA (38.90 dB)



(c) JPEG (43.28 dB)

FIGURE II.2.10 – Comparaison subjective sur la séquence “Poupées”.

Chapitre II.3

Multiview-LS

Dans cette section, nous présentons notre seconde contribution qui consiste en un schéma de compression sans perte basé sur l'algorithme JPEG-LS. Celui-ci permet d'encoder des séquences multi-vues afin de permettre un stockage sans générer de distorsions sur le signal d'entrée. Bien que la compression sans perte puisse paraître relativement superflue dans le cadre de séquences multi-vues, nous pensons qu'un algorithme de ce type pourra convenir pour le stockage temporaire de scènes destinées à la post-production. En effet, dans ce cas, l'ajout ultérieur de multiples effets sur les différentes images composant la séquence multi-vues pourrait être perturbé par la présence d'artefacts générés par une compression avec pertes. Ainsi, dans ce cas précis, le stockage temporaire de ces séquences compressées à l'aide de notre algorithme peut être bénéfique. Dans un premier temps, nous décrirons l'algorithme utilisé par JPEG-LS (LOCO-I) sur lequel repose notre approche. Nous détaillerons ensuite le principe de notre adaptation aux séquences multi-vues, puis nous comparerons les résultats obtenus par notre approche face à plusieurs méthodes de compression sans perte généralement citées dans l'état de l'art.

II.3.1 L'algorithme de base JPEG-LS

JPEG-LS, qui contitue le standard ISO (ISO (c)) en matière de compression sans perte (ou presque sans perte), est basé sur l'algorithme LOCO-I (LOw COmplexity LOssless COmpression for Images). Celui-ci a été développé par Weinberger *et al.* (1996) et permet d'obtenir des ratios de compression approximativement équivalents à ceux obtenus avec l'algorithme CALIC, mais avec un coût logarithme comparable à des méthodes de type FELICS. Contrairement à CALIC, l'algorithme repose sur une modélisation de contextes asymétriques (180°) et reprend les principes généraux des méthodes basées sur la modélisation de contextes (voir section II.1.1.3).

L'algorithme fonctionne de la façon suivante : l'image I à compresser est parcourue en "raster-scan" et pour chaque pixel x, les différentes étapes décrites ci-dessous sont effectuées.

a) Détermination du contexte $\mathcal{C}$

Lors de cette étape, JPEG-LS va déterminer le contexte $\mathcal{C}$ associé à x via son template $\mathcal{T}$. Pour chaque pixel x, on considère le template $\mathcal{T}$ constitué par les pixels a, b, c et d conformément à la figure II.3.1.

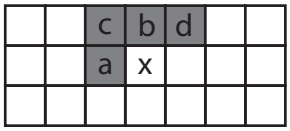


FIGURE II.3.1 – Template du pixel courant x

Afin de déterminer le contexte courant, l'algorithme se base sur les gradients autour de x. Ces gradients (g_1, g_2 et g_3), qui permettent la détection de zones soit uniformes ou au contraire contenant des bords, sont directement liés à l'erreur de prédiction et sont calculés en utilisant l'équation II.3.1.

$$\begin{aligned} g_1 &= d - b \\ g_2 &= b - c \\ g_3 &= c - a \end{aligned} \tag{II.3.1}$$

où chaque gradient g_i $(1 \leq i \leq 3)$ a une valeur comprise dans l'intervalle $[-255; 255]$.

Un des problèmes majeurs concernant les approches basées sur la modélisation de contexte est la "dilution de contexte" qui apparaît lorsque les statistiques associées au modèle probabiliste sont distribuées sur un trop grand nombre de contextes et qui peut réduire l'efficacité du système de codage en aval. Or, associer un contexte $\mathcal{C}$ à chaque triplet $[g_1, g_2, g_3]$ produirait un nombre de contextes égal à 511^3. Afin de remédier à ce problème, JPEG-LS quantifie g_1, g_2 et g_3 sur un certain nombre de régions équiprobables afin de produire q_1, q_2, q_3. De plus, une deuxième réduction de ce nombre de contextes est réalisée en fusionnant les contextes de signes opposés car on peut considérer que la probabilité pour que l'erreur de prédiction ϵ appartienne au contexte $\mathcal{C}$ défini par le triplet $[q_1, q_2, q_3]$ est égale à la probabilité que ϵ appartienne au contexte $-\mathcal{C}$ associé au triplet $[-q_1, -q_2, -q_3]$. Lorsque $q_1 = q_2 = q_3 = 0$, on se situe alors dans une zone stationnaire, on entre en mode "Run-length" : on compte le nombre de pixels contenant la même valeur puis on code la longueur de la chaîne à l'aide de l'algorithme "block-MELCODE" (basé sur Ohnishi *et al.* (1977)). Nous ne détaillerons pas le mode "run-Length" ici, pour plus d'informations le lecteur peut se référer à ISO (c).

Pour une image 8-bit/composante, le standard définit alors les 9 régions équiprobables suivantes : $\{0\}$, $\pm\{1, 2\}$, $\pm\{3, 4, 5, 6\}$, $\pm\{7, 8, ..., 20\}$, $\pm\{21, ...\}$. Chaque gradient quantifié q_i peut donc prendre une des 9 valeurs appartenant à l'intervalle $[-4, 4]$ et un contexte $\mathcal{C}$ est alors défini pour chaque triplet $[q_1, q_2, q_3]$. Le nombre total de contextes est ainsi réduit à $(9^3 + 1)/2 = 365$. Chaque contexte $\mathcal{C}$ possède son propre modèle statistique matérialisé par quatre compteurs A, B, C et N :

- A contient l'accumulation de la valeur absolue des erreurs de prédiction (soit $|\epsilon|$) et permet de calculer le paramètre k de l'algorithme de Rice-Golomb pour la phase de codage entropique (voir le paragraphe "Calcul et codage de l'erreur de prédiction ϵ" de cette section). Il est initialisé à 4.
- B contient l'accumulation des erreurs de prédiction (soit ϵ) et est utilisé pour le calcul de la valeur de correction adaptative C (voir ci-après). Il est initialisé à 0 au début de l'algorithme.
- C contient la valeur de correction associée au contexte courant. Comme B, celui-ci est initialisé à 0.
- N contient le nombre de pixels associés au contexte courant. Il est initialisé à 1.

b) Génération de la valeur de prédiction $\hat{x}$

Pour générer la valeur de prédiction $\hat{x}_{MED}$ associée à x, JPEG-LS utilise une fonction de prédiction fixe désignée par le terme "median edge detector" (MED) et donnée par l'équation II.3.2. Ce type de fonction de prédiction permet de détecter la présence ou non de bords (horizontaux ou verticaux) et d'adapter la valeur de $\hat{x}_{MED}$ en conséquence.

$$\hat{x}_{MED} = \begin{cases} \min(a,b) & \text{si } c \geq \max(a,b) \\ \max(a,b) & \text{si } c \leq \min(a,b) \\ a+b-c & \text{sinon} \end{cases} \tag{II.3.2}$$

Soit ϵ_{MED} l'erreur de prédiction fixe telle que $\epsilon_{MED} = x - \hat{x}_{MED}$. On considère généralement dans la littérature (Netravali et Limb (1980)), que les fonctions de prédiction fixes (telle que le MED) génèrent des erreurs de prédiction ϵ_{MED} dont les statistiques d'apparition peuvent être matérialisées par TGSD (Two-Sided Geometric Distribution centrée en zéro, voir figure II.3.2). Toutefois, on constate qu'il peut exister un décalage O entre le pic de la TSGD et l'abscisse zéro. Ce décalage peut être décomposé sous la forme $O = R - s$ où R constitue la partie entière et s appartient à l'intervalle $[0;1[$. L'étape de correction adaptative est destinée à compenser la partie entière R à l'aide des différentes statistiques associées au contexte $\mathcal{C}$ afin de produire des erreurs de prédiction corrigées ϵ appartenant à $]-1;0]$. La façon la plus intuitive de calculer cette valeur de correction C' consiste à prendre la moyenne des erreurs de prédiction associées au contexte courant $\mathcal{C}$ conformément à l'équation II.3.3.

$$C' = \lceil D/N \rceil \tag{II.3.3}$$

où D représente l'accumulation des erreurs de prédiction fixe ϵ_{MED} associées au contexte $\mathcal{C}$ et N contient le nombre d'occurrences associées à $\mathcal{C}$.

Toutefois, JPEG-LS ne retient pas cette solution pour deux raisons majeures : tout d'abord, la division dans le calcul de C' est en contradiction avec la faible complexité algorithmique imposée par JPEG-LS et de plus, la présence d'erreurs de prédiction fixes à forte amplitude

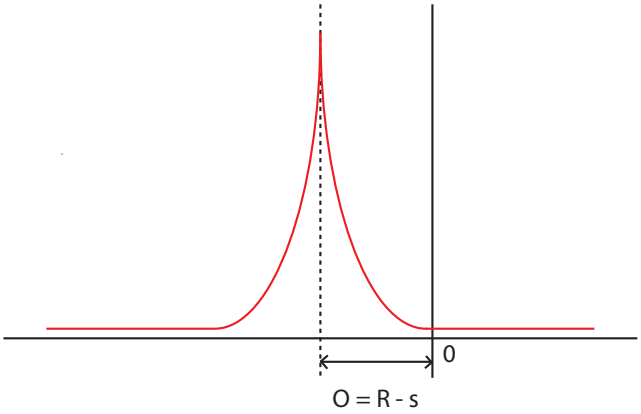


FIGURE II.3.2 – Distribution des erreurs de prédiction fixe

se répercuterait de façon dramatique sur la valeur de correction C'. Afin de palier à ces deux problèmes, JPEG-LS propose une approche différente. Tout d'abord, on peut remarquer que l'équation II.3.3 est équivalente à la formulation suivante (équation II.3.4) :

$$D = N \times C' + B' \tag{II.3.4}$$

où B' vérifie $-N < B' \leq 0$.

L'idée majeure de la procédure de correction adaptative va donc consister à toujours forcer B' à rester dans cet intervalle et à ajuster C' en conséquence. Pour cela, à chaque itération de l'algorithme, on ajoute l'erreur de prédiction corrigée à B', puis tant que B' ne se situe pas dans l'intervalle $(-N; 0]$ il lui ajoute/soustrait N. Les additions (resp. soustractions) successives opérées sur B' sont ensuite répercutées sur C' en décrémentant (resp. incrémentant) celui-ci. Afin de rendre ce mécanisme plus clair, prenons le cas où $B' > 0$ et où on lui a donc soustrait N, dans ce cas :

$$\begin{aligned} N \times C' + B' = N \times C'' + (B' - N) &\leftrightarrow N \times C' = N \times C'' - N \\ &\leftrightarrow N \times C' = N \times (C'' - 1) \\ &\leftrightarrow C' = C'' - 1 \\ &\leftrightarrow C'' = C' + 1 \end{aligned}$$

Dans l'algorithme JPEG-LS, les valeurs B' et C' sont approximées par les compteurs B et C (présentés dans le paragraphe précédent). De plus, l'algorithme limite le nombre d'additions/soustractions à une par itération, puis fixe B à la valeur $-N + 1$ (resp. 0) si celui-ci vérifie

$B \leq -N$ (resp. $B > 0$). Cette procédure de calcul de la valeur de correction C (désignée par le terme "Bias computation" et résumé par l'algorithme II.3.1), permet ainsi de contourner les deux problèmes induits par l'équation II.3.3. Celle-ci est exécutée à la fin de chaque itération de l'algorithme et permet de mettre à jour la valeur de correction associée au contexte courant $\mathcal{C}$. C constitue alors une approximation de R tandis que $-B/N$ est une estimation de la partie fractionnaire s.

Algorithme II.3.1 Procédure de calcul de la valeur de correction

début
si $B \leq -N$
  $C \longleftarrow C - 1$;
  $B \longleftarrow B + N$;
  si $B \leq -N$
    $B \longleftarrow -N + 1$;
  fin si
sinon si $B > 0$
  $C \longleftarrow C + 1$;
  $B \longleftarrow B - N$;
  si $B > 0$
    $B \longleftarrow 0$;
  fin si
fin si
fin

Il suffit alors d'ajouter (soustraire) C à $\hat{x}_{MED}$ dans le cas où le signe associé à $\mathcal{C}$ est positif (resp. négatif) pour obtenir la valeur de prédiction corrigé $\hat{x}$.

c) Calcul et codage de l'erreur de prédiction ϵ

Une fois la valeur de prédiction corrigée ($\hat{x}$) calculée, on peut maintenant calculer l'erreur de prédiction $\epsilon = x - \hat{x}$. L'algorithme JPEG-LS effectue ensuite une réduction de l'alphabet : pour un alphabet dont les symboles sont codés sur 8 bits, ϵ a des valeurs comprises entre -255 et 255. Toutefois, pour une valeur de prédiction corrigée $\hat{x}$ donnée (connue à cet instant par l'encodeur et le décodeur), on constate que ϵ peut prendre 256 valeurs possibles. Ainsi, on choisit alors de réduire l'intervalle $[-255; 255]$ à l'intervalle $[-128; 127]$ (qui contient bien le nombre de valeurs possibles) correspondant à la fusion des extrémités de la TGSD avec leur partie centrale.

Il faut maintenant coder l'erreur de prédiction ϵ : pour cela JPEG-LS utilise le codage de Rice-Golomb. Comme cela est souligné dans Gallager et Voorhis (1975), le codage de Rice-Golomb est particulièrement adapté pour le codage de symboles dont les statistiques d'apparition suivent une loi géométrique. Toutefois, il est nécessaire de redistribuer les différentes valeurs possibles de ϵ car le codage de Rice-Golomb s'effectue sur des valeurs non négatives. Pour cela, on va devoir projeter l'intervalle $[-128; 127]$ sur l'intervalle $[0; 255]$ en utilisant une des deux relations suivantes (nous rappelons que $-B/N$ correspond à une estimation de la partie fractionnaire de l'offset O définie dans le paragraphe précédent) :

– lorsque $-B/N \leq \frac{1}{2}$ (le centre de la TSGD est plus près de 0 que de -1) alors :

$$M(\epsilon) = 2|\epsilon| - u(\epsilon)$$

où $u(\epsilon) = 1$ si $\epsilon < 0$, ou 0 sinon.

– lorsque $-B/N > \frac{1}{2}$ (le centre de la TSGD est plus près de -1 que de 0) alors :

$$M'(\epsilon) = M(-\epsilon - 1)$$

De cette façon, les différentes valeurs possibles de ϵ sont rangées par ordre décroissant de probabilité : les valeurs de $M(\epsilon)$ 0,1,2,3,4... correspondent en fait aux valeurs de ϵ 0,-1,1,-2,2 (il en va de même pour $M'(\epsilon)$). On peut maintenant coder $M(\epsilon)$ (ou $M'(\epsilon)$) grâce au codage de Rice-Golomb. Cet algorithme de codage utilise un paramètre k (déjà évoqué dans la section II.1.1.3.1 où il est appelé k_m) qui va permettre de contrôler le nombre de bits nécessaire pour coder une information. Plus k sera grand, plus le nombre de bits en sortie sera élevé. D'après Gallager et Voorhis (1975), une bonne estimation pour la valeur optimale de k pour le contexte $\mathcal{C}$ est donnée par l'équation suivante (II.3.5) :

$$k = \lceil \log_2 E[|\epsilon|] \rceil \quad \text{où } E \text{ représente l'espérance mathématique.} \tag{II.3.5}$$

Dans la pratique, JPEG-LS approxime la quantité $E[|\epsilon|]$ par la quantité A/N (ces deux variables ont été définies plus haut et sont spécifiques au contexte courant $\mathcal{C}$). Ainsi une approximation de la valeur optimale de k pour le contexte $\mathcal{C}$ peut être calculée grâce à l'équation II.3.6.

$$k = \min_{k'} \{k' | 2^{k'} N \geq A\} \tag{II.3.6}$$

d) Mise à jour des différents compteurs et ré-initialisation éventuelle de ceux-ci

Une fois l'erreur résiduelle ϵ codée, on doit maintenant mettre à jour les différents compteurs associés au contexte courant. Pour cela, on ajoute ϵ à B, $|\epsilon|$ à A et on incrémente N. Afin de donner plus de poids aux dernières occurrences associées au contexte $\mathcal{C}$, JPEG-LS utilise une procédure de ré-initilisation de ses différents compteurs : lorsque le nombre d'occurences associées à $\mathcal{C}$ est égal à un certain nombre N_0 (défini par l'utilisateur et généralement compris entre 32 et 256), on divise alors A, B et N par deux.

Le compteur C (contenant la valeur de correction utilisée durant la phase de prédiction adaptative) doit lui aussi être actualisé, pour cela on utilise la procédure définie par l'algorithme II.3.1.

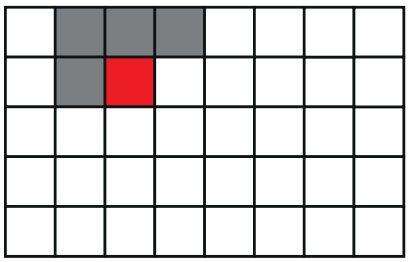
Une fois ces différentes opérations effectuées, l'algorithme reprend alors depuis le début sur le pixel suivant.

II.3.2 Adaptation de l'algorithme aux images multi-vues

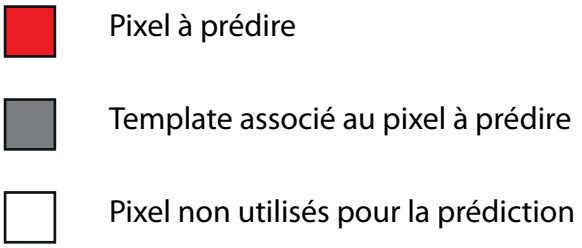
Dans cette section, nous présentons le fonctionnement de l'algorithme. Comme nous l'avons vu précédemment, l'algorithme JPEG-LS fonctionne sur la modélisation de contexte. Chaque contexte possède ses propres statistiques d'apparition de symboles et permet ainsi une compression adaptative. Ce contexte est déterminé à l'aide d'un template (groupe de pixels généralement situé dans le voisinage du pixel courant).

Lors de différents tests sur l'algorithme JPEG-LS, notamment lors de l'extension de celui-ci à la prise en charge de valeurs de pixels comprises entre $[-255; 255]$ pour MICA (voir section II.2.2), nous nous sommes demandés ce qu'il arriverait si au lieu de considérer un voisinage de pixels au sein de la même image, nous considérions le groupe de pixels constitués de ceux situés aux mêmes localisations que le pixel courant mais sur les images voisines.

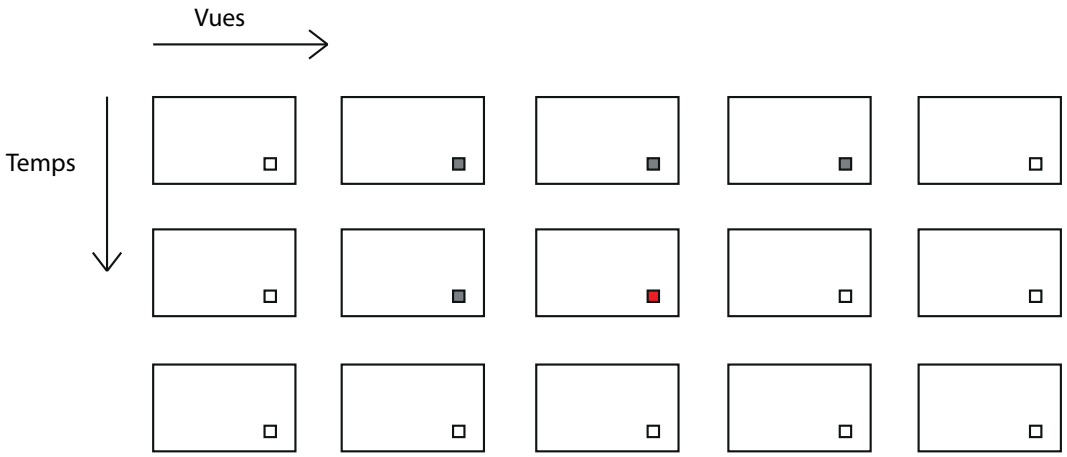
La figure II.3.3 compare le template standard du JPEG-LS avec celui du multiview JPEG-LS.



(a) Approche standard de JPEG-LS



(b) Légende

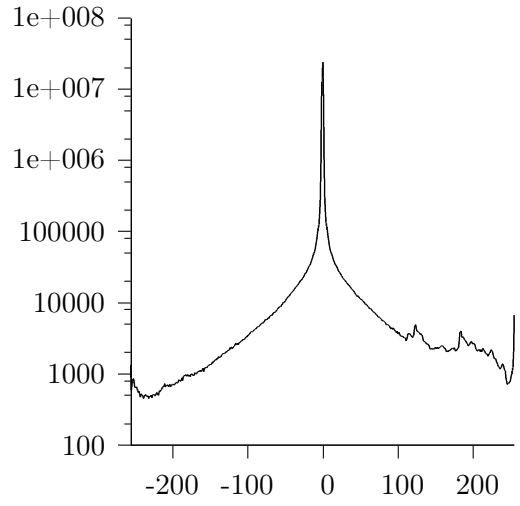


(c) Notre approche

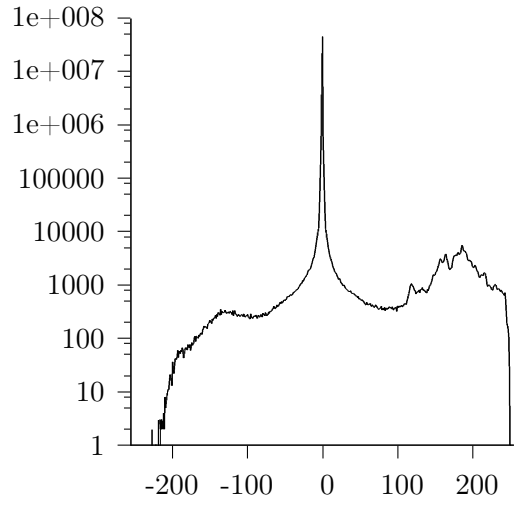
FIGURE II.3.3 – Comparaison entre la structure de prédiction du standard JPEG-LS et notre approche multiview-LS.

D'après la figure II.3.3, on voit très clairement que JPEG-LS utilise un template composé de 4 pixels pour son étape de modélisation de contexte. Soit (i, j) l'emplacement du pixel à prédire (où i désigne le numéro de la ligne et j le numéro de la colonne), celui-ci utilise les pixels situés aux emplacements $(i-1, j-1)$, $(i-1, j)$, $(i-1, j+1)$, $(i, j-1)$.

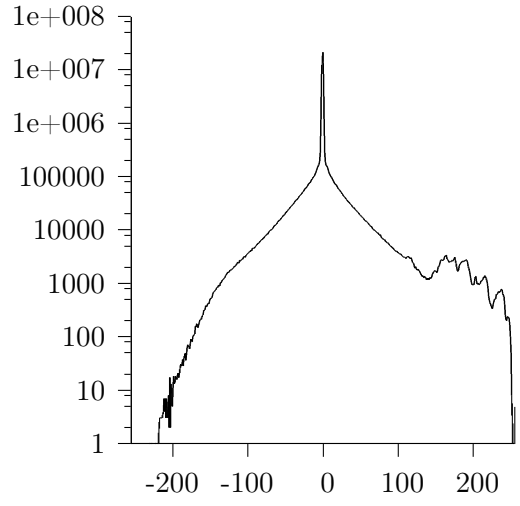
Notre algorithme cherche à exploiter le modèle probabiliste utilisé par JPEG-LS mais sur l'ensemble des images composant une séquence multi-vue. Pour cela, le template $\mathcal{T}$ associé au pixel courant x (où x est situé à l'emplacement (i, j)) va être constitué de pixels situés à l'emplacement (i, j) et appartenant à des images voisines. De manière plus formelle, soit $I_{t,v}(i, j)$ le pixel situé à l'emplacement (i, j) dans la vue v au temps t, l'étape de prédiction fixe va utiliser les pixels $I_{t-1,v-1}(i, j)$, $I_{t-1,v}(i, j)$ et $I_{t,v-1}(i, j)$. Ainsi, au lieu d'exploiter les redondances spatiales au sein de la même image, on exploite cette fois-ci à la fois les redondances spatiales (dans le sens inter-vues) et les redondances temporelles. La modélisation de contexte va quant à elle, être réalisée à partir du template constitué des 4 pixels suivants : $I_{t-1,v-1}(i, j)$, $I_{t-1,v}(i, j)$, $I_{t-1,v+1}(i, j)$, $I_{t,v-1}(i, j)$ (conformément à la figure II.3.3(c)).



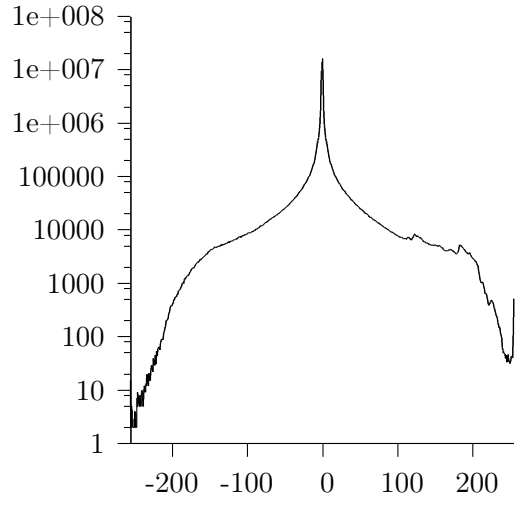
(a) Séquence "Cartes"



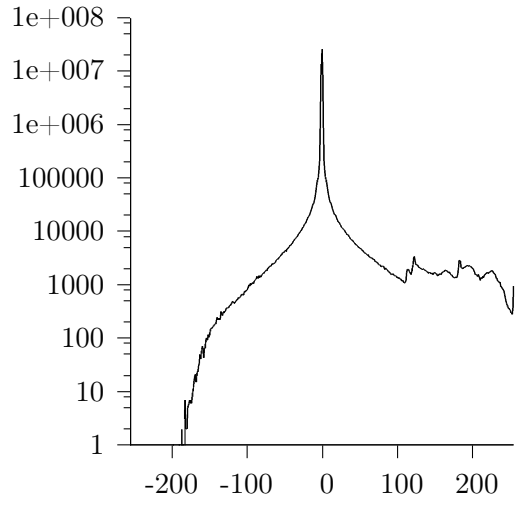
(b) Séquence "Pinceau"



(c) Séquence "Poupées"



(d) Séquence "Rose"



(e) Séquence "Statue"

FIGURE II.3.4 – Distribution des erreurs de prédiction fixes.

De manière analogue à JPEG-LS, lorsque l'information nécessaire à la modélisation de contexte n'est pas disponible (typiquement dans les cas où l'on se situe sur les bords de la matrice d'image), on la remplace alors par la valeur de l'information disponible la plus proche.

Toutefois, il convient de prendre certaines précautions concernant l'adaptation de l'algorithme JPEG-LS au multi-vues : en effet, JPEG-LS est basé sur le fait que la distribution des erreurs de prédiction fixe (ϵ_{MED}) peut être matérialisée par une TSGD (voir II.3.2). Cette distribution est ensuite exploitée à l'aide du codage de Rice-Golomb qui est particulièrement adapté dans ce cas (comme nous l'avons déjà précisé plus haut). Pour que l'adaptation de JPEG-LS aux séquences multi-vues, selon la démarche décrite ci-dessus, puisse être considérée comme viable, nous devons nous assurer que la distribution des ϵ_{MED} suit toujours le même comportement. Pour cela, nous avons donc mesuré l'amplitude des différentes erreurs de prédiction fixe afin de mettre en avant leur distribution, et ce, pour nos cinq jeux de données. La figure II.3.4 présente donc la distribution des ϵ_{MED} pour les différentes séquences multi-vues (afin de permettre une meilleure lisibilité des différents résultats, nous avons utilisé une échelle logarithmique sur l'axe des ordonnées). On constate bien d'après les courbes ci-dessus, que la distribution des erreurs de prédiction fixe suit bien une TSGD et donc que l'emploi du codage de Rice-Golomb reste approprié.

L'autre élément clef de JPEG-LS réside dans son étape de modélisation de contexte via les trois gradients g_1, g_2 et g_3. Dans la version de base de l'algorithme, ceux-ci nous permettaient d'associer un contexte au pixel courant suivant l'activité locale autour de celui-ci. Dans la version multi-vues, ces gradients vont permettre la caractérisation d'une activité temporelle (due au mouvement entre le temps t (temps actuel) et le temps $t-1$) grâce à g_3, et spatiale (due à l'effet parallaxe en entre la vue courante v et ses deux vues voisines $v-1$ et $v+1$) grâce à g_1 et g_2.

Une des autres particularités de notre schéma de compression réside dans le fait que l'exploitation des zones uniformes au sein de l'image en cours de traitement reste possible. En effet, avant de calculer les gradients sur les frames voisines, on calcule tout d'abord ceux définis par l'algorithme JPEG-LS de base. Dans le cas où ceux-ci sont tous les trois égaux à zéro, alors on applique le mode "run-length" afin d'exploiter la zone uniforme de l'image en cours de traitement. Le cas échéant, les pixels associés à l'approche multi-vues sont utilisés.

Dans la partie suivante, nous présentons les résultats obtenus par notre approche et nous les confrontons à ceux obtenus avec différentes méthodes de compression sans perte, de l'état de l'art (non nécessairement propres à la compression d'images) : bzip2, LZMA, CALIC (Wu (1995, 1996), voir section II.1.1.3.2), GIF (Blackstock (1987), voir section II.1.1.2.1), JPEG-2000 (Taubman et Marcellin (2002), ISO (b), voir section II.1.2.2.1) et JPEG-LS (Weinberger *et al.* (1996), ISO (c), voir section II.3.1).

II.3.3 Résultats et discussions

Cette section présente les résultats obtenus en terme de ratio de compression par notre algorithme multiview-LS. Nous avons choisi de comparer ces résultats avec ceux obtenus par plusieurs algorithmes de compression sans perte généralement utilisés dans l'état de l'art. Les

différents tests ont été effectués sur des séquences 8 vues constituées de 26 frames temporelles. La table II.3.1 présente les différents résultats obtenus.

Séquence	bzip2	CALIC	GIF	JPEG-2000	JPEG-LS	LZMA	multiviewLS	PNG
Cartes	5.23	4.90	7.27	5.50	4.98	3.25	2.79	5.40
Pinceau	1.68	2.67	4.28	2.87	2.69	0.24	2.00	3.21
Poupees	4.28	3.72	6.38	3.94	3.77	3.15	3.07	4.41
Rose	5.08	4.37	6.87	4.77	4.43	4.30	4.28	4.92
Statue	5.00	4.67	7.62	5.02	4.74	2.24	2.21	5.21

TABLEAU II.3.1 – Résultats obtenus (bits/pixel) en utilisant différents algorithmes de compression sans perte sur nos différentes séquences multi-vues

D'après la table II.3.1, on voit clairement que de manière générale, notre algorithme de compression sans perte permet d'obtenir de meilleurs taux de compression (en bits/pixel) que les différents algorithmes testés. Toutefois, on constate que pour la séquence "Pinceau" qui contient très peu de détails, notre schéma génère de meilleurs résultats que les algorithmes de compression sans perte dédié à l'image mais que LZMA et bzip2 surpasse celui-ci. Notre algorithme représente donc un bon compromis de compression sans perte dans le cadre d'un stockage de vidéos multi-vues.

Conclusion

La première partie de ce chapitre a permis de dresser un état de l'art des méthodes de compression mono-scopiques (associées à des images 2D). Cet état de l'art avait pour objectif d'assurer au lecteur une meilleure compréhension, soit des différentes contributions que nous avons développées dont certaines sont basées sur ces méthodes, soit des schémas de compression dédiés aux médias multi-vues (présentés dans le chapitre suivant) qui reprennent des concepts issus de la compression 2D (ou 2D+t). Il nous semblait donc nécessaire de décrire ces différentes méthodes.

Dans la deuxième partie de ce chapitre, nous avons présenté notre première contribution : MICA (Multiview Image Compression Algorithme). Cet algorithme de compression multi-vues à faible complexité basé sur la génération d'images de différences entre deux vues adjacentes. Celui-ci prend en compte les diverses spécificités de notre dispositif de capture 8 vues, l'Octocam, telles que la géométrie de capture parallèle décentrée ainsi que l'utilisation de capteurs à filtres de Bayer. MICA permet d'une part, grâce à sa faible complexité algorithmique, un encodage temps réel des séquences produites par l'Octocam, et d'autre part, de préserver les zones sujettes au parallaxe d'éventuelles distorsions. Il peut donc être directement mis en place en sortie de la caméra, afin de réduire le volume de données à envoyer sur le réseau pour des applications de type visio-conférence. L'inconvénient majeur de MICA est que celui-ci ne permet pas d'atteindre des ratios de compression élevés. Cependant, plusieurs améliorations restent possibles comme l'ajout d'un paramètre de quantification, l'utilisation du codage de Rice-Golomb (particulièrement adapté à la distribution des pixels contenus dans les images de différence, voir section II.3.1) ou d'un codeur arithmétique binaire pour le codage entropique de nos images de différence.

La partie suivante est consacrée à notre deuxième contribution : Multiview-LS. Bien qu'il soit rare dans la littérature de trouver des algorithmes de compression sans perte dédiés aux médias multi-vues, il nous semblait important de proposer ce type de solution. En effet, une étape de post-production peut être opérée sur une séquence multi-vues afin d'y ajouter d'éventuels effets grâce à des logiciels de type After Effects ou 3D-Tricks (version multi-vues d'After Effects développé par la société 3DTVSolutions). Le présence d'artefacts liés à la compression de ces séquences (tels que des effets de "blocking" générés par l'emploi de la DCT) peut rendre cette étape fastidieuse et nécessiter des retouches "à la main" de la part de l'utilisateur. Notre algorithme, basé sur JPEG-LS, permet la compression sans perte de séquences multi-vues tout en exploitant la corrélation spatio-temporelle de celles-ci. L'étude des résultats obtenus a permis

de montrer que notre algorithme permet d'obtenir de meilleurs taux de compression dans la majorité des cas, comparé aux méthodes généralement utilisées dans l'état de l'art.

Le prochain chapitre est dédié aux méthodes de compression spécifique aux séquences multi-vues. Nous dresserons tout d'abord un état de l'art dans lequel nous énumèrerons les différents formats et techniques de codage associées à la vision multi-scopique. Nous décrirons ensuite notre approche LDI spécifique à la géométrie de capture de nos différents systèmes d'acquisition pour finir par la présentation des diverses contributions apportées dans ce cadre.

Troisième partie

Méthodes de compression dédiées aux flux multiscopiques

Chapitre III.1

État de l'art

Cette section présente de manière exhaustive les différentes méthodes dédiées à la compression des médias (images et vidéos) N-vues ($N \geq 2$). La présente section est organisée en deux parties distinctes : en effet, pour la compression des médias multi-vues on constate que deux approches différentes sont possibles. La première consiste à appliquer un système de codage directement sur les N vues en entrée (éventuellement accompagnées des N cartes de profondeurs, afin de permettre la synthèse de points de vue intermédiaires via des méthodes DIBR (Depth Image-Based Rendering) telle celle proposée par Smolic *et al.* (2008). La deuxième approche va tout d'abord réaliser une dé-corrélation de l'information redondante d'une vue à l'autre. Les méthodes utilisées seront détaillées dans la suite. Ces méthodes de dé-corrélation en amont reposent sur l'utilisation des cartes de profondeurs et donc implicitement sur des méthodes DIBR. La quantité d'information ainsi réduite sera ensuite compressée à l'aide de méthodes utilisées dans la première approche (si le format de données reste compatible), ou bien à l'aide de schémas de codage spécifiques. Afin de bien marquer la différence entre ces deux processus distincts, on désignera par le terme "format 3D" l'ensemble des techniques permettant une dé-corrélation préalable des données, et par le terme "codage 3D", l'ensemble des méthodes réalisant une compression à proprement parler.

III.1.1 Approches stéréoscopiques

III.1.1.1 Les différents formats 3D spécifiques à la stéréo-vision

Cette section présente les différents formats 3D spécifiques à la stéréo-vision ($N = 2$) et généralement évoqués dans la littérature Vetro *et al.* (2011), Broberg (2011), Dol (2010), Brust *et al.* (2009). Ces différents formats peuvent être regroupés en trois catégories :

– Le format MRS (Mixed Resolution Stereo)
– Les formats "frame-compatible"
– Le format 2D + Depth

Dans la suite de cette section, nous allons décrire ces différents formats et mettre en avant les avantages mais aussi les inconvénients liés à leur utilisation. Avant toute chose, il convient de définir correctement le concept de flux vidéo stéréoscopique. Un flux stéréoscopique consiste en deux flux vidéos distincts produits soit par un couple de caméras disposées selon une distribution géométrique spécifique, soit en utilisant des logiciels de rendu 3D (par exemple 3DSMax) couplés avec un système de caméras virtuelles. La figure III.1.1 décrit un flux stéréoscopique standard. Dans la littérature, on associe généralement ce type de flux au format 3D CSV (Conventional Stereo video) même si celui-ci ne constitue pas réellement un format à proprement parler (aucune transformation n'est opérée sur le couple stéréo).

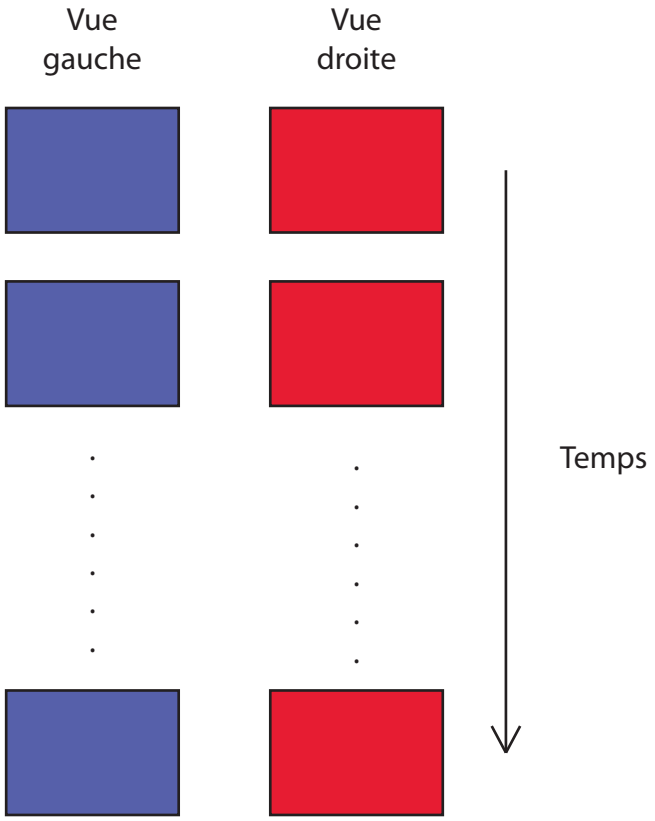


FIGURE III.1.1 – Un flux vidéo stéréoscopique de base (CSV).

III.1.1.1.1 Le format MRS (Mixed Resolution Stereo)

Le format MRS (Mixed Resolution Stereo) est basé sur la "théorie de la suppression binoculaire" (Stelmach *et al.* (2000)). Celle-ci énonce le fait qu'il est possible de sous-échantillonner une des deux vues du couple stéréoscopique sans altérer la qualité subjective perçue par l'utilisateur. Une des deux vues est définie comme référence et son homologue comme vue auxiliaire. Généralement,

on choisit alors une résolution donnée pour la vue de référence et on sous-échantillonne la vue auxiliaire en divisant la résolution horizontale et la résolution verticale par deux (conformément à la figure III.1.2).

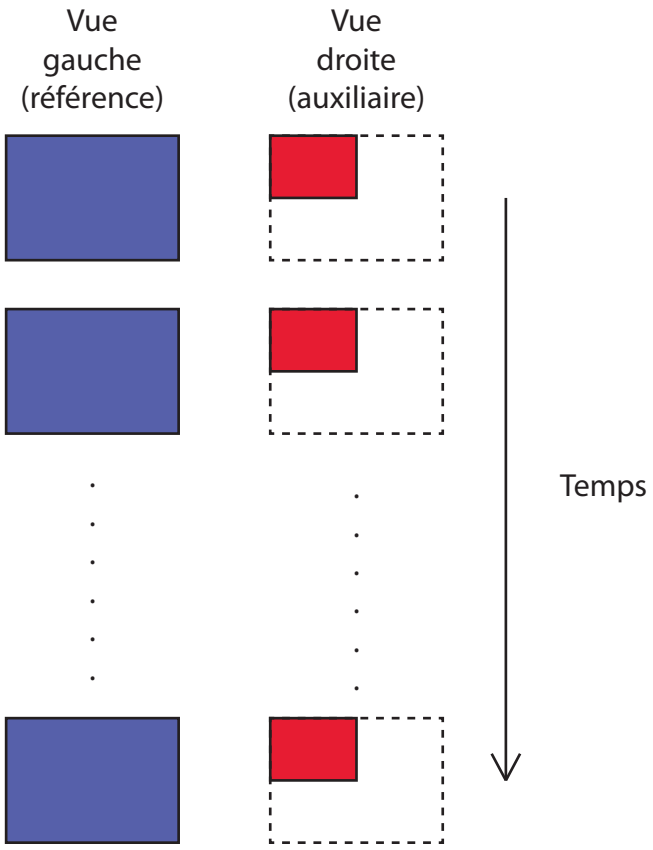


FIGURE III.1.2 – Le format 3D MRS.

Ces deux vues sont ensuite envoyées de manière indépendante au processus de codage en aval. Le format MRS permet ainsi de réduire le volume de données associé à la deuxième vue du couple stéréo, mais via une étape de sous-échantillonnage qui ne permet pas de garder les images à pleine résolution.

III.1.1.1.2 Les formats "frame-compatible"

Le but des formats "frame-compatible" consiste à transformer un flux stéréoscopique en un flux vidéo standard via un multiplexage spatial ou temporel des vues gauche/droite. Ce flux peut ainsi être codé à l'aide de processus déjà existants (non compatibles avec les flux stéréoscopiques). De plus, le volume de données associé à ce nouveau flux est équivalent à celui de la version monoscopique de la séquence.

a) Les formats côte-à-côte (side-by-side)

Contrairement au format MRS où la résolution au sein du couple stéréoscopique est hétérogène, les formats côte-à-côte réalisent la même opération de sous-échantillonnage sur chacune des deux vues. Ainsi, on garde une certaine homogénéité au sein du couple stéréoscopique et le volume de données à coder par la suite reste équivalent à un flux vidéo standard (1 vue). Il existe deux versions du format côte-à-côte : le côte-à-côte vertical (où la résolution verticale des deux vues est divisée par deux) et le côte-à-côte horizontal (où c'est la résolution horizontale qui est divisée par deux). Ces deux vues sont ensuite placées côte-à-côte dans une unique frame ayant la résolution d'origine. Les figures III.1.3(a) et III.1.3(b) illustre respectivement le format côte-à-côte vertical et horizontal. Le flux vidéo résultant est ensuite envoyé tel quel au processus de codage.

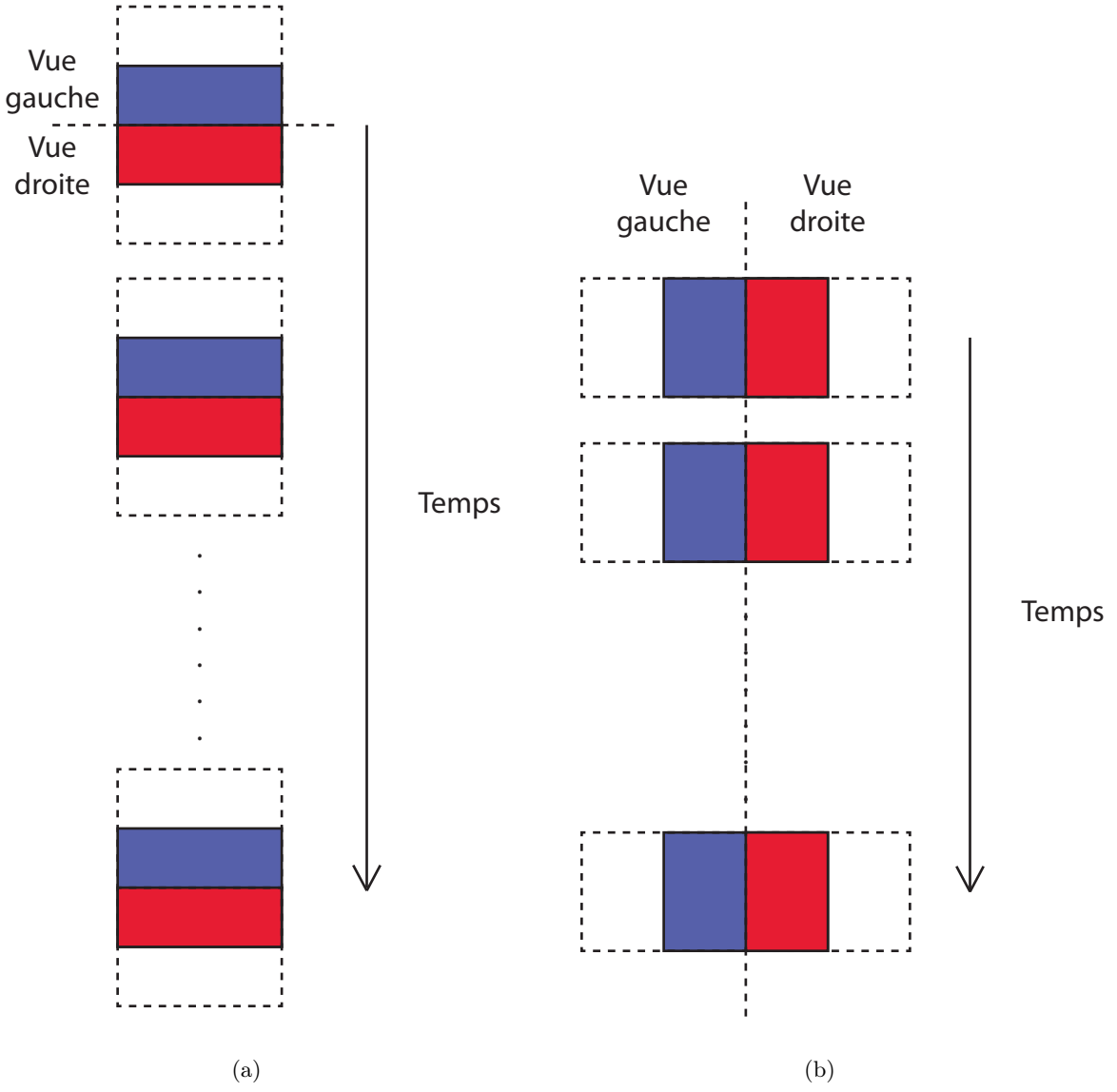


FIGURE III.1.3 – Les formats côte-à-côte : (a) vertical et (b) horizontal.

b) Les formats entrelacés spatialement

L'approche utilisée par ce type de format est relativement équivalente à l'approche précédente (les deux vues du couple stéréo sont échantillonnées de facon homogène) et présente le même avantage concernant le volume de données ainsi généré. Toutefois, elle diffère dans le processus de fusion des deux vues au sein d'une même frame. Cette fois-ci, les deux vues vont être entrelacées suivant trois schémas distincts : un entrelacement colonne par colonne (dans le cas d'un sous-échantillonnage horizontal, voir fig III.1.4(a)), ligne par ligne (dans le cas d'un sous-échantillonnage vertical, voir fig III.1.4(b)) ou en damier (si le sous-échantillonnage est en quinconce, voir fig III.1.4(c)).

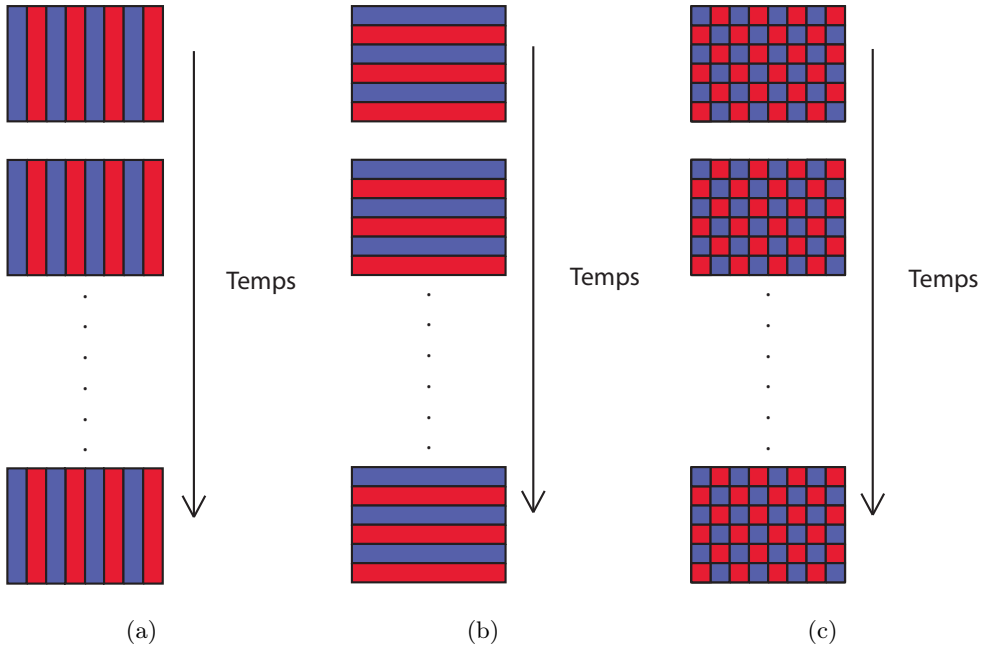


FIGURE III.1.4 – Les formats entrelacés : (a) horizontal, (b) vertical et (c) en damier.

Une chose importante à noter concernant ces formats, c'est que le système de codage en aval doit pouvoir prendre en compte le type d'entrelacement effectué. En effet, l'entrelacement des deux vues constituant le couple stéréoscopique réduit la corrélation spatiale des pixels appartenant à une même frame du flux entrelacé. L'encodeur doit donc pouvoir être capable de gérer cette particularité afin de séparer les pixels appartenant à chacune des deux vues pour les reconstituer.

c) Les formats entrelacés temporellement

Le format 3D utilisant cette approche exploite la résolution non plus spatiale mais temporelle du flux stéréoscopique afin de réduire le volume de données. On le qualifie généralement par le terme "frame sequential" et dans la suite, nous le désignerons par le terme FSS (Frame Sequential Stereo). Pour cela, on alterne chaque vue constituant le couple stéréoscopique conformément à la figure III.1.5. Ce format permet ainsi de garder la résolution spatiale d'origine sur chacune des

deux vues afin d'assurer une qualité d'image optimale. Toutefois, pour que le volume de données reste équivalent à un flux vidéo standard, la résolution temporelle de chaque vue est divisée par deux, ce qui peut rendre gênant le visionnage du média pour le spectateur.

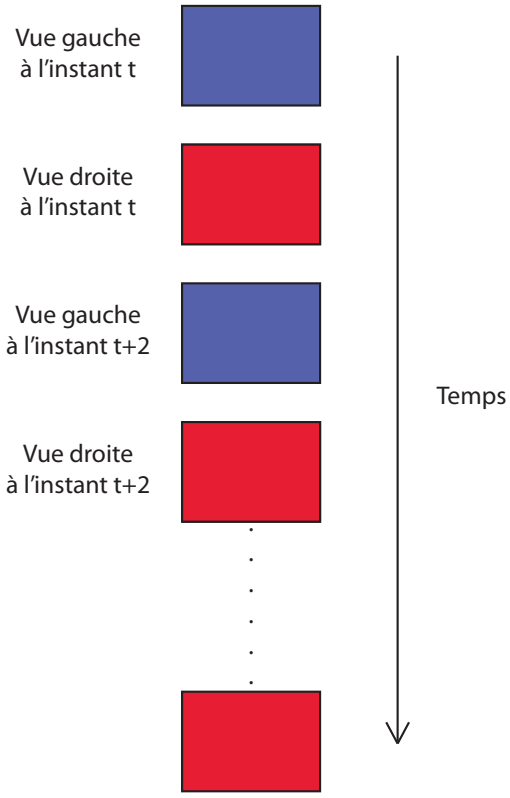


FIGURE III.1.5 – Le format 3D FSS.

III.1.1.1.3 Le format 2D + Depth

Dans cette dernière section, nous présentons le format 2D + Depth qui peut utiliser des cartes de profondeur ou de disparité. Une carte de profondeur est une image permettant de connaître la distance séparant la surface des différents objets d'une scène et un point de vue donné. Une carte de disparité permet quant à elle une représentation de la parallaxe associée aux différents objets de la scène (cette notion de parallaxe est intrinsèquement liée à la notion de profondeur).

Le flux vidéo associé à ce format contient une vue chromatique ainsi que la carte de profondeur (ou de disparité) associée à celle-ci (voir figure III.1.6). Cette carte de profondeur est soit produite à l'aide de matériel spécifique tel qu'une Z-cam, soit générée à l'aide de méthodes d'estimation de profondeur à partir des deux vues d'origine. Grâce à l'information présente dans la carte de profondeur, il est ensuite possible de reconstituer le couple stéréoscopique via des méthodes DIBR. De plus, ce format permet une rétro-compatibilité directe pour l'affichage 2D de la vue chromatique. Cette méthode permet de réduire considérablement le volume de données à compresser (le débit alloué à la carte de profondeur étant généralement bien moindre que celui nécessaire au codage d'un vue). Toutefois, elle souffre de plusieurs inconvénients majeurs :

– La qualité de la vue synthétisée est clairement liée à la précision de la carte de profondeur ainsi qu'à la méthode de synthèse de points de vues utilisée.
– La présence de zones d'occultations importantes dans la vue encodée va générer des incohérences sur la vue synthétisée et altérer la vision stéréoscopique lors de la restitution.

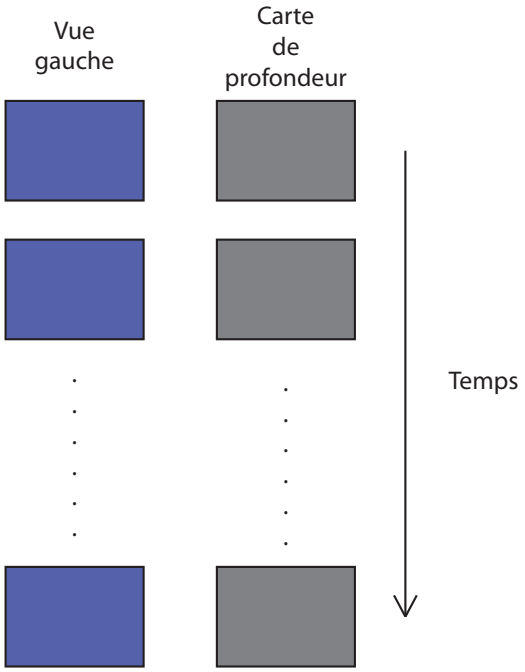


FIGURE III.1.6 – Le format 2D + Depth.

III.1.1.2 Les différentes techniques de codage spécifiques à la stéréo-vision

Le but de la précédente section était de présenter les différents format 3D spécifiques aux médias stéréoscopiques. Nous allons maintenant détailler les différentes approches possibles en terme de codage, associées à ces formats et mettre en avant les avantages et inconvénients liés à celles-ci. De manière générale, elles reposent sur l'utilisation du codec H.264/AVC qui permet grâce à ses fonctionnalités et extensions (SEI, MVC) de gérer le problème du codage des médias stéréoscopiques.

III.1.1.2.1 H.264/AVC Simulcast

L'approche H.264/AVC Simulcast consiste à encoder le flux vidéo associé à chacune des deux vues de manière similaire mais indépendante (conformément à la figure III.1.7). Cette solution a l'avantage d'être simple et peut être réalisée à l'aide du codec H.264/AVC standard. Toutefois elle ne peut pas être considérée comme satisfaisante. En effet, les deux vues constituant le couple stéréoscopique présentent de fortes similarités or cette corrélation doit être exploitée afin de réduire au maximum le débit en sortie. Avec l'approche "Simulcast", le débit en sortie est approximativement deux fois celui nécessaire au codage d'une vue, ce qui n'est pas acceptable.

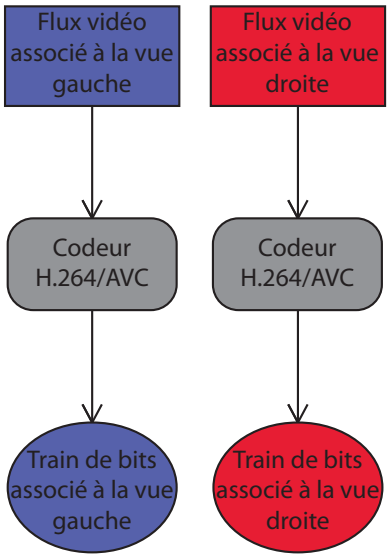


FIGURE III.1.7 – Codage du média stéréoscopique en H.264/AVC Simulcast.

III.1.1.2.2 MPEG-2 MVP (MultiView Profile)

En 1996, MPEG-2 se voit doté d'un profil multi-vues (plus précisément dédié au codage des médias stéréoscopiques). Cette norme permet l'exploitation de la corrélation spatiale entre les deux flux vidéos constituant une séquence stéréoscopique. Pour cela, l'extension MVP adopte la structure de prédiction décrite par la figure III.1.8.

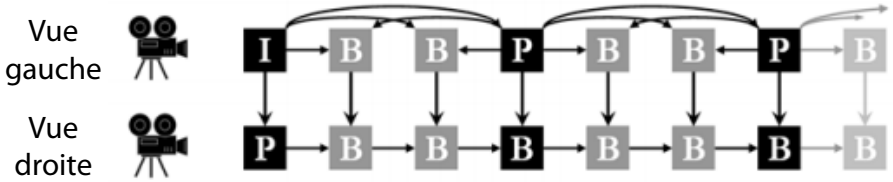


FIGURE III.1.8 – Structure de prédiction adoptée pour l'extension MVP de MPEG-2.

Une des deux vues est considérée comme "base view" et est codée en utilisant uniquement la prédiction temporelle permettant le décodage indépendant de celle-ci et assurant ainsi la rétro-compatibilité pour un affichage 2D. L'autre vue, désignée par le terme "enhancement view", est elle, codée en utilisant à la fois la prédiction temporelle mais aussi la prédiction inter-vues. Toutefois, la prédiction inter-vues est très peu utilisée car on constate généralement que deux frames successives temporelles présentent plus de similarités que deux frames spatialement voisines. La prédiction temporelle étant très souvent utilisée, on constate que MPEG-2 MVP n'offre pas un gain significatif vis-à-vis de la compression par rapport à l'approche Simulcast.

De plus, MPEG-2 MVP semble obsolète comparé aux standard actuels tel que le profil "Stereo" de l'extension H.264/MVC (présenté en section III.1.1.2.5) qui propose exactement la même structure de prédiction mais permettant un codage efficace car basé sur H.264/AVC. Une autre

alternative consisterait à utiliser le profil "Stereo High Profile" de H.264/AVC (voir III.1.1.2.4) qui permet l'utilisation de formats "frame-compatible" via des messages SEI spécifiques et la réduction de manière significative du volume de données en amont du processus de codage.

III.1.1.2.3 MPEG-C part 3 et H.264/AVC APS (Auxiliary Picture Syntax)

Ces deux normes (présentées dans Merkle *et al.* (2009)) sont dédiées au codage des médias stéréoscopiques de type 2D + Depth. En effet, ces normes permettent l'ajout d'un flux auxiliaire associé au flux vidéo standard. Toutefois, on constate que MPEG-C part 3 et H.264/AVC APS présentent certaines différences, détaillées ci-dessous.

MPEG-C part 3 ne constitue pas un système de codage à proprement parler, cependant, il fournit une syntaxe haut-niveau permettant au décodeur d'interpréter les données contenues dans le flux auxiliaire. Pour l'instant, deux types de données sont définies : les cartes de profondeur et les cartes de disparité. Toutefois, la définition de nouveaux types de données reste possible. Par la suite on considérera le cas qui nous intéresse, c'est à dire un flux auxiliaire contenant l'information de profondeur. Celui-ci est compatible avec la majorité des codecs actuels tels que H.264/AVC. Le processus de codage est illustré par la figure III.1.9(a). Les deux flux correspondant respectivement à la video 2D et à l'information de profondeur sont codés de manière indépendante afin de produire deux flux binaires distincts. Ces deux flux sont ensuite rassemblés en un flux unique par entrelacement temporel des frames les constituant. Une des autres possibilités concernant MPEG-C part 3 est le sous-échantillonnage (spatial et temporel) du flux auxiliaire, afin de pouvoir s'adapter à de faibles débits.

H.264/AVC APS, quant à lui, associe une composante auxiliaire au flux video standard et encode ces deux séquences de manière simultanée mais indépendante pour produire un flux binaire unique (le processus d'encodage est illustré par la figure III.1.9(b)). Aucune information additionnelle n'est ajoutée et l'interprétation des données est laissée à la charge de l'utilisateur (contrairement à MPEG-C part 3). De plus, chaque frame constituant le flux auxiliaire doit contenir le même nombre de macro-blocs qu'une frame appartenant au flux principal (le sous-échantillonnage uniquement pour l'information de profondeur n'est donc pas autorisé.)

III.1.1.2.4 H.264/AVC Stereo High Profile

Le standard H.264/AVC définit dans sa norme un profil "Stereo High" afin de pouvoir gérer le codage des flux stéréoscopiques sans avoir à apporter de modification au codec. En effet, H.264/AVC permet le codage de vidéos entrelacées via le mode de codage "field coding". Dans ce mode, la frame en cours de codage est entrelacée selon un schéma spécifique (généralement ligne par ligne) et H.264/AVC prend en compte cette particularité, en codant simultanément plusieurs lignes de même parité pour former ses macro-blocs.

Le profil "Stereo High" reprend ce mode de codage afin de l'adapter au codage des séquences stéréoscopiques utilisant l'un des formats 3D "frame-compatible" décrits dans la section III.1.1.1. Comme nous l'avons précisé dans la section en question, afin de permettre le codage de la séquence

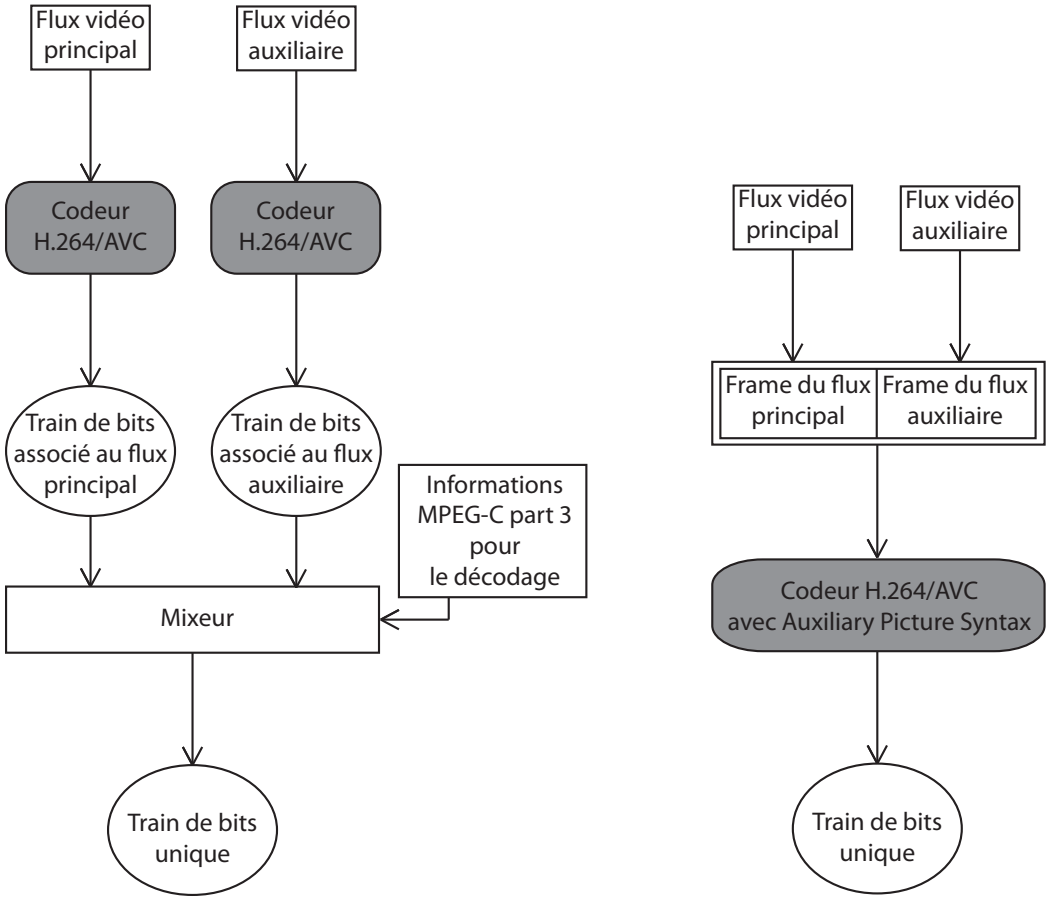


FIGURE III.1.9 – Processus de codage : (a) MPEG-C part 3, (b) H.264/AVC Auxiliary Picture Syntax.

stéréoscopique en utilisant un des formats précédemment décrits, le codeur doit connaître à l'avance le type d'entrelacement utilisé, à quelle image du couple stéréo est associée la première ligne de la frame en cours de codage (dans le cas d'un entrelacement ligne par ligne) .... Pour cela H.264/AVC dispose des messages SEI (Supplemental Enhanced Information) : ces messages permettent de transmettre diverses informations nécessaires soit au codeur soit au décodeur. Dans le cas du profil "Stereo High", ces informations sont transmises via des messages SEI de type FPA (Frame Packing Arrangement) qui décrivent à la fois l'organisation des deux vues constituant le couple stéréo dans le flux vidéo mais aussi si l'une des deux vues est utilisée pour prédire l'autre grâce au mécanisme de prédiction inter-vues (décrit de façon plus détaillée dans la section III.1.2.2.1). La figure III.1.10 décrit le processus d'encodage d'une séquence stéréoscopique à l'aide de l'extension "Stereo High" de H.264/AVC.

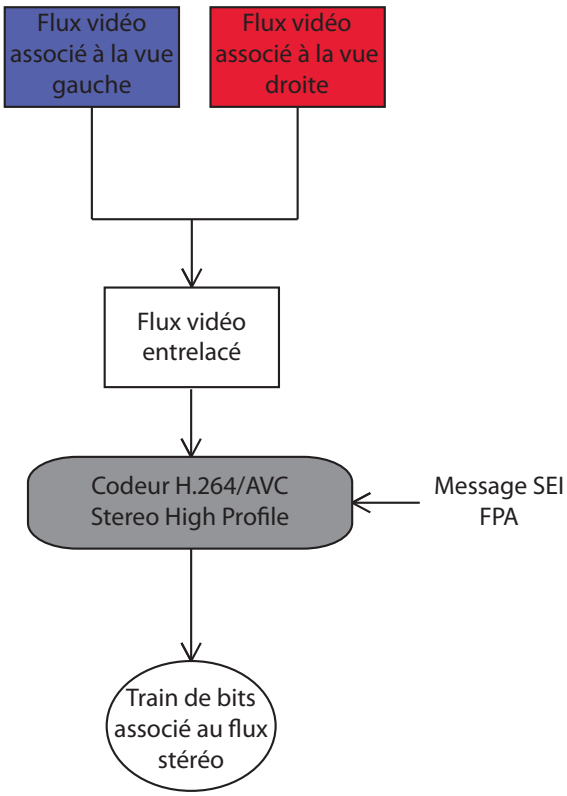


FIGURE III.1.10 – Codage du média stéréoscopique avec le profil "Stereo High" H.264/AVC.

L'utilisation du profil "Stereo High" a l'avantage de permettre le codage ainsi que la diffusion de flux stéréoscopiques sur des plateformes qui n'y étaient pas dédiées à l'origine et il est le seul à être adapté aux flux stéréo entrelacés (H.264/MVC ne dispose pas du mode "field coding"). Toutefois, l'utilisation de ce profil nécessite obligatoirement un sous-échantillonnage (spatial ou temporel) de chacune des deux vues pouvant entraîner une perte de qualité lors de la restitution.

III.1.1.2.5 H.264/MVC

Une alternative possible au profil "Stereo High" consiste à utiliser l'extension MVC (MultiView Coding) du standard H.264/AVC. L'extension MVC (décrite en détails dans la section III.1.2.2.1) permet l'exploitation de la forte redondance existant entre les différentes vues d'une séquence multiscopique via un mécanisme appelé prédiction inter-vues ("inter-view prediction"). Ce mécanisme couplé à la prédiction inter-frame standard (prédiction temporelle) permet ainsi d'étendre la structure de prédiction sur la grille 2D de frames spécifique aux médias multi-vues. Une des deux vues, considérée comme vue de référence, va uniquement utiliser le système classique de prédiction inter-frame, alors que la vue homologue dispose à la fois de la prédiction inter-frame mais aussi de la prédiction inter-vue (conformément à la figure III.1.11). Le mode de prédiction offrant les meilleures performances en termes de débit/distorsion (RDO) est alors utilisé comme prédicteur.

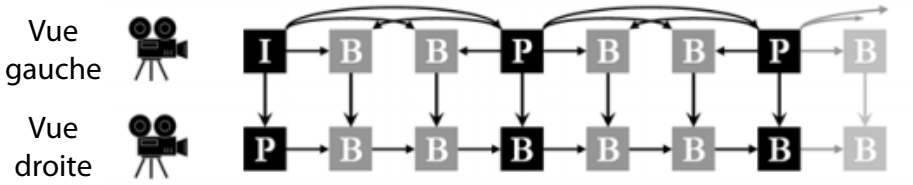


FIGURE III.1.11 – Grille de prédiction pour le profil stéréo de H.264/MVC.

La figure III.1.12 illustre le processus d'encodage d'un flux stéréoscopique réalisé à l'aide du profil "Stereo" de H.264/MVC.

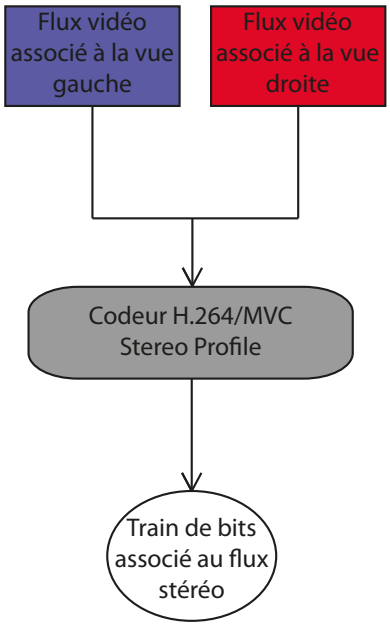


FIGURE III.1.12 – Codage du média stéréoscopique avec le profil stéréo de H.264/MVC.

Contrairement au profil "Stereo High" de H.264/AVC, le profil "Stereo" de MVC ne permet

pas le codage de vidéos entrelacées car il ne supporte pas le mode “field coding”. Toutefois, celui-ci permet d'éviter un sous-échantillonnage spatial ou temporel de chacune des vues constituant le couple stéréoscopique. Ce profil de H.264/MVC, spécifique aux flux stéréoscopiques, a d'ores et déjà été retenu pour la norme Blu-Ray 3D.

III.1.2 Approches multi-vues

Cette section est consacrée à la description des différents formats et techniques de codage spécifiques aux médias auto-stéréoscopiques ($N > 2$). Comme pour le cas de la stéréo-vision, chacune des N vues en entrée présente de fortes similarités. Cette forte corrélation doit être exploitée afin de réduire la dépendance linéaire du nombre de vues en entrée et le débit nécessaire pour transmettre de telles séquences. Ce besoin est d'autant plus important que les écrans permettant l'affichage d'un nombre de plus en plus important de points de vue, le volume de données à traiter en est d'autant plus conséquent.

Comme pour la section III.1.1, nous allons tout d'abord décrire les différents formats 3D spécifiques à l'auto-stéréoscopie puis nous nous pencherons sur les différentes techniques de codage à proprement parler.

III.1.2.1 Les différents formats 3D spécifiques à l'auto-stéréoscopie

Dans cette section, nous présentons les différents formats 3D spécifiques à l'auto-stéréoscopie généralement décrits dans la littérature. Un flux auto-stéréoscopique consiste en la capture simultanée d'une même scène par un ensemble de N caméras ($N > 2$) disposées selon une géométrie spécifique. Ce type de flux, correspondant au format 3D MVV (MultiView Video), est généralement représenté par une matrice 2D d'images conformément à la figure III.1.13. Sur cette figure, on peut voir un flux multi-vues constitué de N flux vidéo (chacun associé à une caméra) et comprenant chacun T frames temporelles.

III.1.2.1.1 Le format MVD (MultiView + Depth)

Le format MVD (MultiView + Depth) reprend le format MVV précédemment décrit mais ajoute, pour chaque vue, la carte de profondeur associée. La figure III.1.14 présente le type d'information contenue dans un flux MVD.

Ce format est généralement utilisé de la façon suivante : sur les N vues en entrée (et les N cartes de profondeur associées), on n'en garde qu'un nombre P ($2 \leq P \leq N$) afin de diminuer de façon significative le volume de données à traiter. Les $N - P$ supprimées vues sont ensuite reconstruites à l'aide de méthodes de synthèse de points de vue intermédiaires (DIBR) Smolic *et al.* (2008). Ce format 3D est donc d'une grande flexibilité : la présence de l'information de profondeur va permettre une compatibilité avec des écrans permettant l'affichage d'un nombre de vues supérieur à N. De plus, le nombre de points de vue conservés étant généralement supérieur

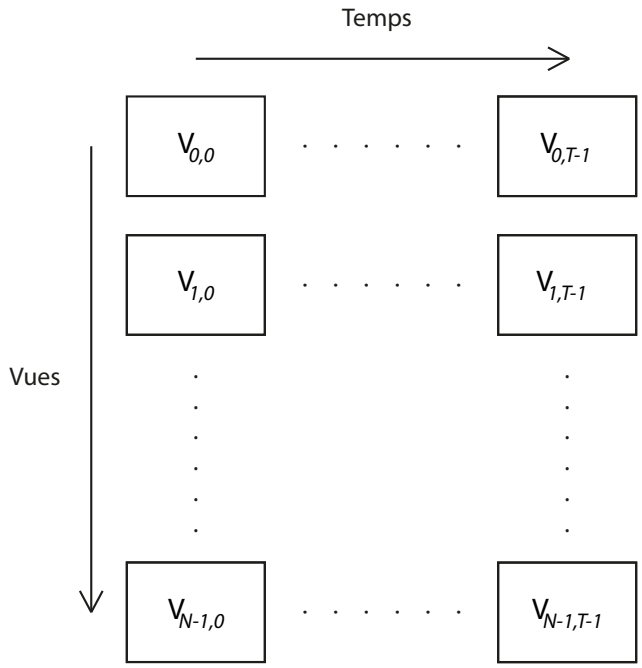


FIGURE III.1.13 – Représentation matricielle d'un flux auto-stéréoscopique.

ou égal à deux, on dispose d'une meilleure gestion des zones d'occultations qu'avec le format V+D. Toutefois, il convient de préciser qu'il subsiste une forte corrélation entre les P vues gardées qui pourrait être exploitée afin de simplifier, voir d'améliorer le processus de codage an aval.

III.1.2.1.2 Le format LDI (Layer-Depth Image)

La notion de LDI (Layered-Depth Image) fut introduite pour la première fois dans les travaux de Shade *et al.* (1998) qui propose une nouvelle méthode IBR (Image-Based Rendering) basée sur les cartes de profondeur (méthodes DIBR). Un LDI correspond à une image comprenant plusieurs couches (aussi désignées par le terme "layers") dont chaque pixel contient non seulement une information colorimétrique mais aussi une information de profondeur. Cette notion de profondeur est très importante car celle-ci va nous permettre d'associer les pixels des différentes vues en entrée afin d'extraire uniquement l'information non redondante. On pourra parfois trouver dans la littérature le terme LDV (Layered Depth Video), celui-ci désignant une succession temporelle de LDI. Nous utiliserons dans la suite de ce document uniquement le terme LDI afin d'éviter toute ambiguïté.

Yoon *et al.* (2007) nous propose une méthode de représentation des images multi-vues permettant de diminuer de façon significative la quantité d'information (en terme de pixels) à fournir au système de compression sous-jacent.

Le principe général de la génération d'une LDI est le suivant : on choisit tout d'abord une vue de référence V_{ref} parmi les N vues en entrée, puis chacune des autres vues V_i (où $V_i \neq$

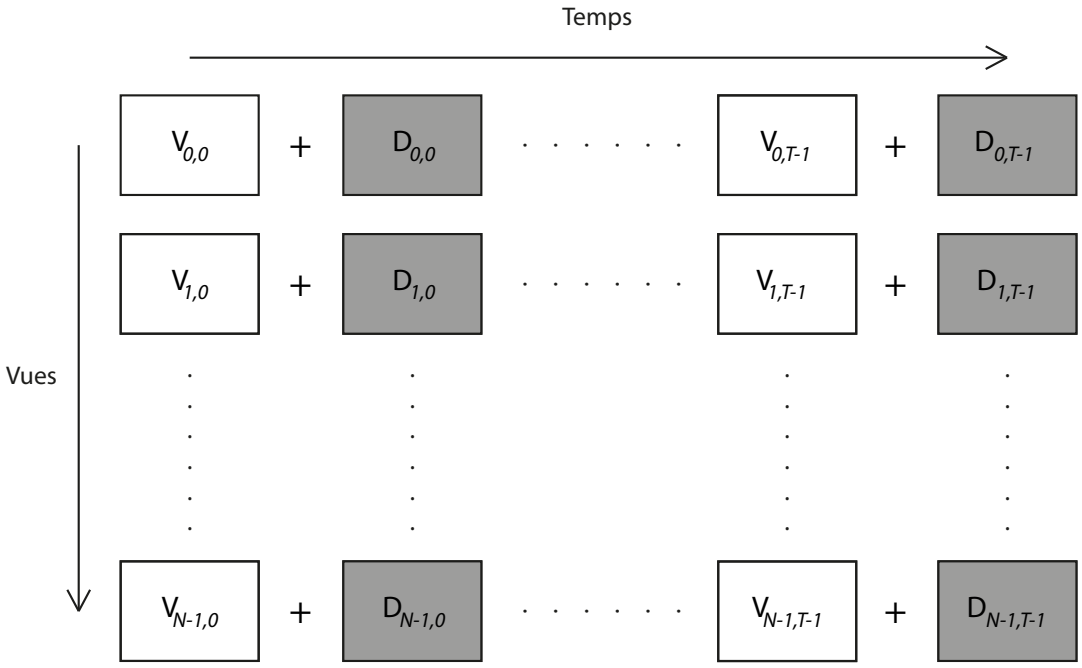


FIGURE III.1.14 – Le format MVD.

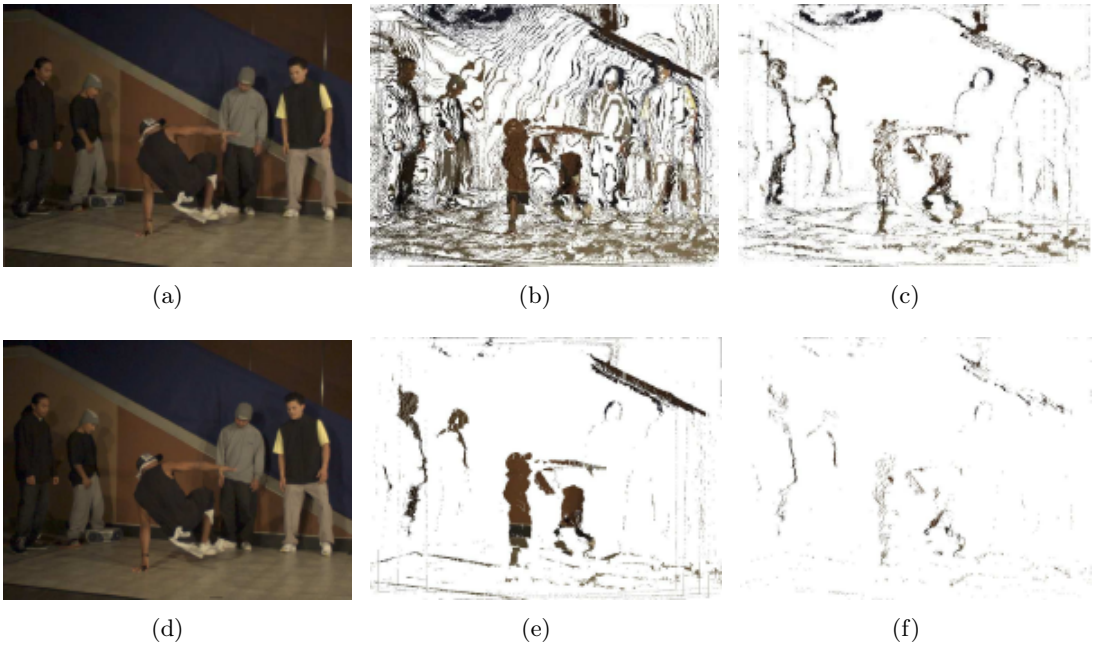
V_{ref}) est projetée sur la vue V_{ref} conformément à leur carte de profondeur et à la matrice de projection $P_{i \to ref}$ associée pour produire la vue $V_{i \to ref}$. Après cette étape de projection, la construction du LDI commence : la première couche de celui-ci est initialisée avec les informations chromatiques et de profondeur associées à la vue V_{ref}. Ensuite pour chaque vue $V_{i \to ref}$ et pour chaque emplacement (j, k) (où j et k désignent respectivement la ligne et la colonne d'une vue), on compare la profondeur du pixel associé à la vue $V_{i \to ref}$ à celles déjà présentes dans la LDI. Si cette différence est supérieure à un certain seuil (fixé par l'utilisateur), alors le pixel est considéré comme non redondant et on ajoute une nouvelle couche à la position (j, k) de la LDI. La figure III.1.15 présente quatre couches de la LDI générée à partir de la séquence multi-vues "Breakdancers".

(a) (b) (c) (d)

FIGURE III.1.15 – 4 couches issues de la LDI générée sur la séquence "Breakdancers" grâce à la méthode de Yoon *et al.* (2007) : (a) couche 0, (b) couche 2, (c) couche 4 et (d) couche 6

Même si les travaux de Yoon *et al.* (2007) ont l'air relativement prometteurs, on constate toutefois que malgré une importante corrélation entre les N vues, le nombre de pixels dans les $N-1$ dernières couches reste important.

Pour pallier à ce problème, Jantet *et al.* (2009) propose une méthode permettant de diminuer fortement le taux d'occupation sur les $N-1$ dernières couches en introduisant la notion de I-LDI (Incremental LDI). Comme pour l'approche de Yoon *et al.* (2007), une vue de référence est définie et permet de constituer la première couche. On va ensuite projeter l'information contenue dans le I-LDI sur chacune des autres vues et extraire l'information résiduelle via un opérateur de type OU exclusif. Cette information résiduelle est ensuite re-projetée sur la vue de référence puis insérée dans la I-LDI. Grâce à ce mécanisme, à chaque étape de l'algorithme (projection de la I-LDI sur la vue i et extraction de l'information résiduelle associée à la vue i), on prend en compte l'information rajoutée durant l'étape précédente permettant ainsi de réduire significativement la corrélation inter-couches et d'améliorer la compression de la LDI. La figure III.1.16 (issue de Jantet *et al.* (2009)) présente une comparaison du taux d'occupation entre l'approche LDI classique (utilisée par Yoon *et al.* (2007)) et l'approche basée sur le I-LDI.



(a) (b) (c)

(d) (e) (f)

FIGURE III.1.16 – Comparaison du taux d'occupation des trois premières couches résiduelles entre l'approche LDI classique ((a), (b) et (c)) et l'approche I-LDI ((c), (d) et (e)).

On voit bien d'après la précédente figure que la méthode proposée par Jantet *et al.* (2009) permet une meilleure décorrélation de l'information que l'approche de Yoon *et al.* (2007) donnant ainsi lieu à de meilleurs taux de compression.

Toutefois ces deux méthodes présentent encore quelques inconvénients, notamment, elles supposent que l'information de profondeur (nécessaire pour la projection des différentes vues et de la I-LDI) est disponible, ce qui n'est pas toujours le cas.

III.1.2.1.3 Le format DES (Depth-Enhanced Stereo)

Le format DES (Depth-Enhanced Stereo), proposé par Smolic *et al.* (2009) combine les avantages du format LDI et du format MVD : sur les N vues en entrées, on choisit deux vues formant un couple stéréo et on génère pour chacune la LDI en la considérant comme vue de référence (conformément à la figure III.1.17).

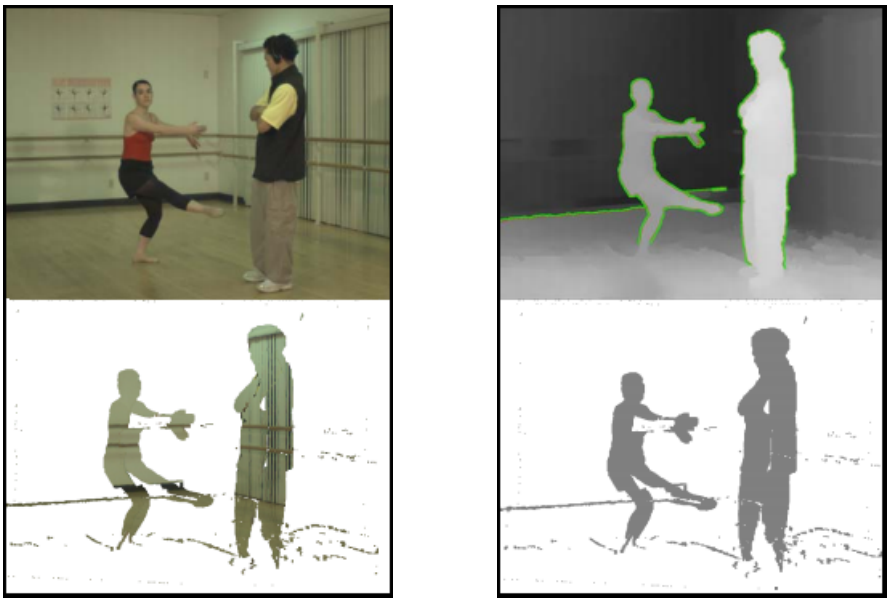
(a) Information contenue dans le LDI de la vue 1

(b) Information contenue dans le LDI de la vue 2

FIGURE III.1.17 – Information fournie par le format DES

Comme le lecteur peut le remarquer d'après la figure précédente, la structure de chaque LDI constituant l'information propre au format DES diffère de celle proposée par Yoon *et al.* (2007) et Jantet *et al.* (2009). Smolic *et al.* (2009) adoptent une approche différente en considérant chaque LDI comme étant constituée de deux couches : le "main layer" contenant l'information redondante au sein de la scène, ainsi que le "background layer" contenant uniquement l'information occultée par les différents objets de la scène sujets à l'effet parallaxe (conformément à la figure III.1.18).

L'avantage majeur concernant ce format est qu'il permet une excellente exploitation de la redondance inter-vues via l'approche LDI, alors que pour le format MVD, une forte redondance subsiste au sein des vues conservées. De plus, la gestion des zones d'occultations pour la synthèse de points de vues intermédiaires (via des méthodes DIBR) est plus robuste qu'avec l'approche LDI car on dispose de plus d'informations. Enfin le format DES est directement compatible avec la vision stéréoscopique via les couches de base contenus dans les deux LDI.



(a) L'information chromatique : en haut, le "main layer" et en bas le "background layer"

(b) L'information de profondeur associée

FIGURE III.1.18 – Structure du LDI relative au format DES décrit par Smolic *et al.* (2009)

III.1.2.2 Les différentes techniques de codage spécifiques à l'auto-stéréoscopie

III.1.2.2.1 H.264/MVC

En 2008, le groupe JVT (Joint Video Team, constitué du groupe VCEG (Video Coding Experts Group) et du groupe MPEG (Moving Picture Experts Group)) propose le nouveau standard en matière de compression multi-vues. Comme spécifié dans le "Call for papers" publié en octobre 2005, ce nouveau standard doit pouvoir répondre aux besoins suivants :

- Permettre un gain en termes de compression par rapport à l'encodage indépendant de chacune des vues (Simulcast).
- Autoriser un accès aléatoire à une vue spécifique v à un instant t donné. (Comme pour un codec vidéo classique (une vue) où l'accès aléatoire temporel est assuré par l'utilisation de frames I).
- Assurer la "view-scalability", permettant d'accéder à une portion spécifique du train binaire afin de n'afficher par exemple qu'un nombre restreint de vues.
- Le flux binaire généré par ce nouveau standard devra être rétro-compatible afin de pouvoir permettre le décodage d'une vue unique par un décodeur H.264/AVC standard.

Structures de prédiction utilisées par H.264/MVC

Le concept majeur de l'extension H.264/MVC consiste à étendre le schéma de prédiction temporel propre à H.264/AVC afin d'exploiter les redondances inter-vues spécifiques aux médias stéréoscopiques et auto-stéréoscopiques. En effet, utiliser le codec H.264/AVC pour un enco-

dage indépendant des flux vidéo associés à chaque caméra ne permet pas l'exploitation de ces redondances et ne peut pas constituer une solution satisfaisante mais plutôt une base pour la comparaison des différents résultats obtenus. La prédiction inter-vue utilise le mécanisme d'estimation/compensation de mouvement sur les différents blocs de la frame en cours de prédiction. Toutefois, là où la prédiction temporelle utilise une ou deux frames situées à des instants t antérieurs et/ou postérieurs, la prédiction inter-vues va, quant à elle, utiliser comme référence des frames situées sur des caméras adjacentes.

Comme pour H.264/AVC, l'extension MVC utilise des frames I ("Intra-frames"), P ("Predicted-frames") et des frames B ("Bi-predicted frames"). Toutefois, les frames P et B vont maintenant être prédites en utilisant la prédiction temporelle, la prédiction spatiale ou bien la prédiction spatio-temporelle (la figure III.1.19 illustre les différents schémas possibles de prédiction dans le cas d'une frame B). L'utilisation de la prédiction spatio/temporelle (en pointillés sur la figure III.1.19) est discutée dans l'étude Merkle *et al.* (2007) et il apparaît que prendre celle-ci en considération n'apporte pas vraiment de gain en terme de compression alors qu'elle augmente grandement la complexité du processus de prédiction. C'est pourquoi ce type de prédiction ne sera pas utilisé par le standard.

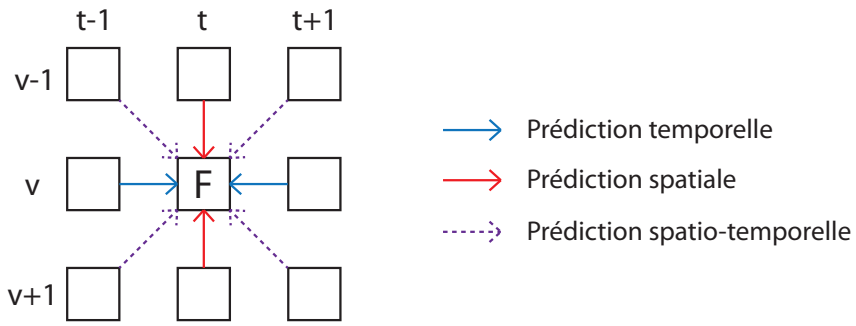


FIGURE III.1.19 – Prédiction temporelle, spatiale et spatio-temporelle de la frame F en cours de codage.

De manière analogue à H.264/AVC où le GOP (Group Of Pictures) définit une unité de codage indépendante, MVC adapte la notion de GOP à la matrice d'images 2D spécifique aux médias multi-vues à travers le GoGOP (Group of Group Of Pictures).

La structure de prédiction par défaut proposée par H.264/MVC définit la première vue comme "base view" : celle-ci va être codée de manière traditionnelle (en exploitant uniquement la prédiction temporelle) et va servir de référence de départ pour les autres vues (désignées par le terme "enhancement views"). La "base view" pourra être décodée sans avoir à procéder à la décompression de frames appartenant à d'autres vues, assurant ainsi une retro-compatibilité pour un affichage standard (mono-vue).

On peut aussi noter que la première frame multi-vue de chaque GoGOP est désignée par le terme "anchor-frame" : chacune des N vues la constituant est codée soit en utilisant l'intra-prédiction (dans le cas d'une frame I), soit en utilisant la prédiction spatiale (pour les frames P ou B). Une "anchor-frame", comme son nom l'indique, va nous servir de point d'ancrage dans le flux H.264/MVC et nous permettre un accès aléatoire dans celui-ci, c'est donc l'équivalent multi-vues de la frame I pour un flux vidéo mono-vue.

Les notions de GoGOP, de "base view", "enhancement views" et d' "anchor-frame" sont représentées dans la figure III.1.20.

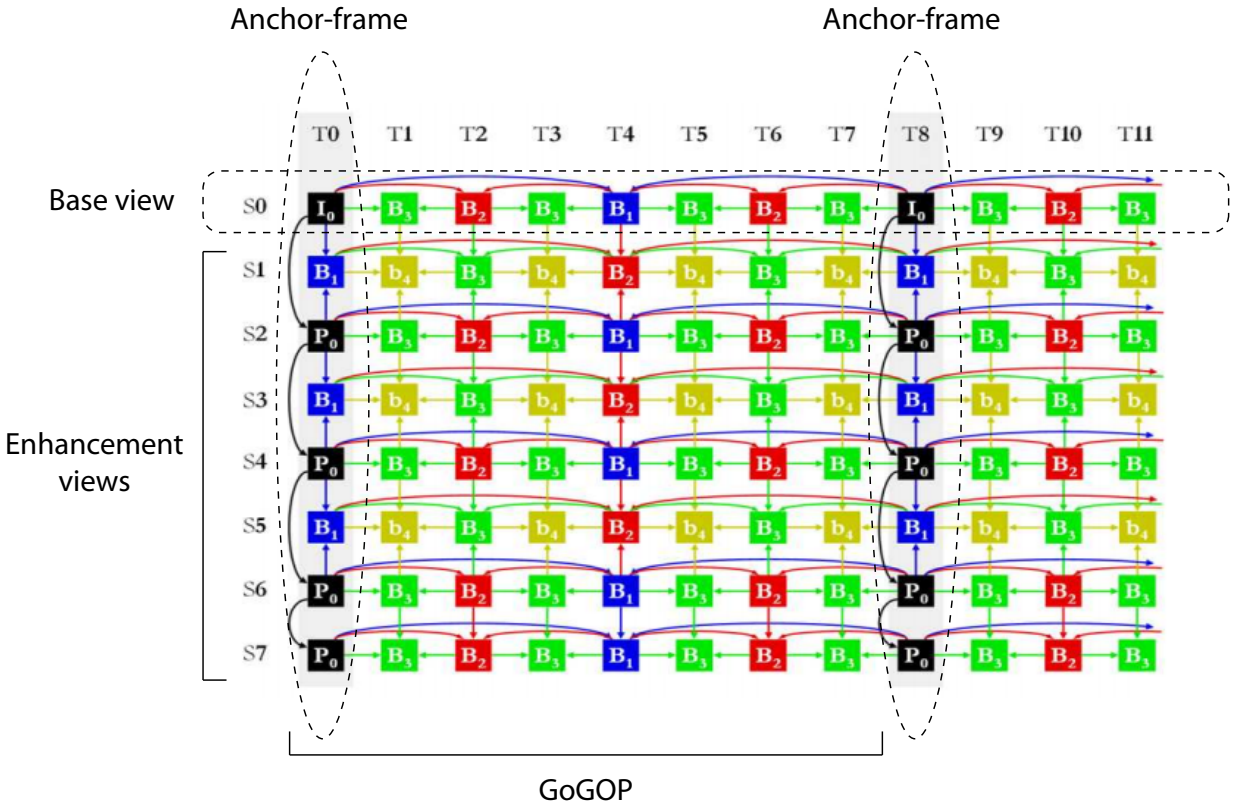


FIGURE III.1.20 – Structure de prédiction adoptée par le standard H.264/MVC pour $N = 8$.

La structure de prédiction décrite par la figure III.1.20 est celle proposée par défaut par l'implémentation standardisée JMVM de l'extension H.264/MVC. Toutefois, l'utilisateur peut entièrement définir cette structure via le fichier de configuration utilisé par JMVM (choix de la "base-view", taille du GoGOP ...). Cela permet une grande flexibilité du schéma de prédiction et ainsi de s'adapter à la distribution géométrique des différentes caméras constituant le dispositif de capture.

Modifications apportées à H.264/AVC

L'extension H.264/MVC est basée sur le profil "High" de H.264/AVC, qui regroupe toutes les fonctionnalités avancées de celui-ci (la majeure partie de ces différentes fonctionnalités a été décrite dans la section II.1.3.3). Il n'y a donc pas eu de modifications "bas-niveau" du codec. Toutefois, il a fallu adapter celui-ci (à travers une syntaxe "haut-niveau") afin de permettre l'utilisation de la prédiction inter-vues.

En effet, l'encodeur H.264/AVC prend en entrée un flux unique de frames, les N flux synchronisés en entrée doivent donc être ré-organisés en un flux unique qui sera envoyé à l'encodeur H.264/AVC. La figure III.1.21 décrit le parcours des différentes frames appartenant aux N flux vidéos.

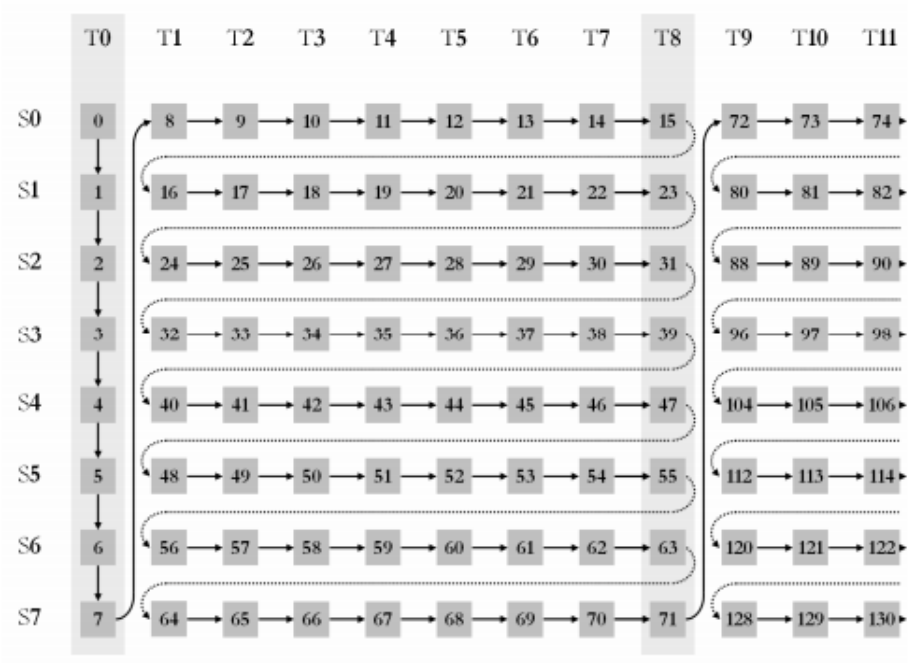


FIGURE III.1.21 – Ré-organisation des N flux vidéos en un flux unique.

Une autre modification apportée concerne la taille des buffers de stockage destinés aux frames de références utilisées pour la prédiction. Celle-ci doit être augmentée afin de prendre en compte cette nouvelle structure de prédiction. Enfin, l'encodeur envoie différentes informations afin de spécifier que le flux binaire en sortie contient N vues. Le décodeur, qui reçoit toutes ces informations, pourra ainsi prendre en compte le type de média (mono-vue ou multi-vues), adapter la taille des différents buffers nécessaires au décodage et reconstruire les N flux vidéos.

Résultats obtenus par H.264/MVC

Plusieurs études Vetro *et al.* (2011) démontrent qu'un gain significatif en terme de compression est obtenu par H.264/MVC par rapport à un encodage Simulcast. Ce gain est en moyenne de 20% pour des séquences multi-vues contenant jusqu'à 8 vues. Bien qu'il soit clair que l'on obtient un gain par rapport à l'approche Simulcast, le bitrate en sorti de l'encodeur reste fortement dépendant du nombre de vues, et plus celui-ci est grand plus ce gain semble diminuer. De plus comme cela est souligné dans Merkle *et al.* (2007), la prédiction temporelle est utilisée dans environ 70% des cas, la corrélation spatiale n'est donc exploitée que dans les 30% restants. Une approche plus appropriée consisterait à, tout d'abord, exploiter au maximum la redondance spatiale au sein d'une frame multi-vue, puis à exploiter la redondance temporelle (comme cela est fait dans l'approche proposée par Jantet *et al.* (2010)).

III.1.2.2.2 Méthodes de codage associées à l'approche LDI

Dans la section III.1.2.1.2, nous présentions les différentes méthodes de génération d'un LDI. Cette LDI qui contient une information chromatique et une information de disparité réparties

toutes les deux sur N couches doit être ensuite compressée. Cette section présente les différentes approches possibles pour le codage de la LDI.

Yoon *et al.* (2007) nous propose deux méthodes d'encodage pour compresser le LDI, toutes les deux étant basées sur le codec H.264/AVC. La première solution consiste à translater horizontalement vers la gauche tous les pixels contenus dans les différentes couches puis à rassembler ceux-ci en une seule et même image conformément à la figure III.1.22. Cette image est ensuite compressée telle quelle grâce à H.264/AVC. Le problème majeur avec l'agrégation de pixels réside dans le fait que l'on va générer des zones dans l'image finale où des pixels voisins ne vont pas présenter de corrélation spatiale entre eux (on peut le voir clairement sur la figure III.1.22, ce qui réduit énormément les performances du codec H.264/AVC en terme de compression mais va aussi générer des artefacts visibles à la décompression. L'autre solution (plus efficace) consiste à remplir tous les endroits où il n'y a pas d'information dans les $N-1$ dernières couches en utilisant celle présente dans le layer 0. Les N couches sont ensuite transmises successivement à l'encodeur H.264/AVC.

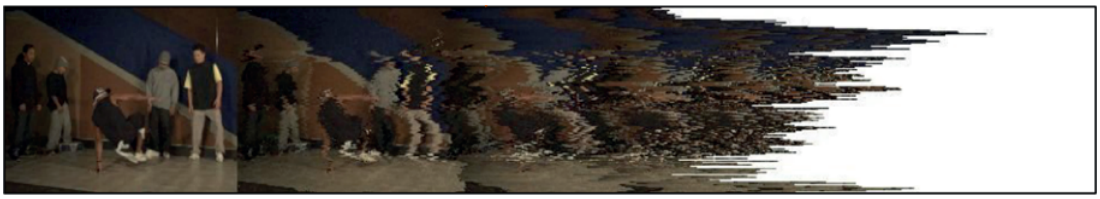


FIGURE III.1.22 – Résultat de l'agrégation des différentes couches.

Dans Jantet *et al.* (2010), celui-ci complète son approche en proposant un schéma de compression basé sur H.264/MVC (Multi-View Coding). Pour cela, il restreint son LDI à deux couches (ce qui lui permet quand même de contenir environ 90% de l'information totale) et complète la deuxième couche (aux endroits où il n'y a pas d'information) par l'information contenue dans le premier layer. On code ensuite l'information chromatique et de disparité séparément (chaque layer étant associé à une vue dans le flux MVD) en utilisant les mêmes paramètres d'encodage. Les résultats obtenus sont ensuite comparés à l'encodage du MVD constitué du même nombre de vues par MVC.

On constate que pour des bitrates inférieurs à 3Mbits/s, l'approche proposée par Jantet *et al.* (2010) permet une réduction significative du bitrate à qualité égale par rapport à l'encodage du MVD.

III.1.2.2.3 Autres méthodes de codage associées aux médias multi-vues

Une alternative possible et intéressante au standard H.264/MVC est l'approche proposée par Zamarin *et al.* (2010) qui est basée sur la projection 3D des différentes vues et sur la DCT 3D. Le processus d'encodage de chaque frame multi-vues est décrit par le diagramme de la figure III.1.23 (issu de Zamarin *et al.* (2010)). Nous ne détaillerons précisément que cette méthode car certains concepts (notamment la quantification et le parcours 3D du cube) sont repris par notre algorithme de codage basé DCT (voir section III.3.1.1.

Dans la suite, nous allons expliquer brièvement les phases principales de cette approche : la

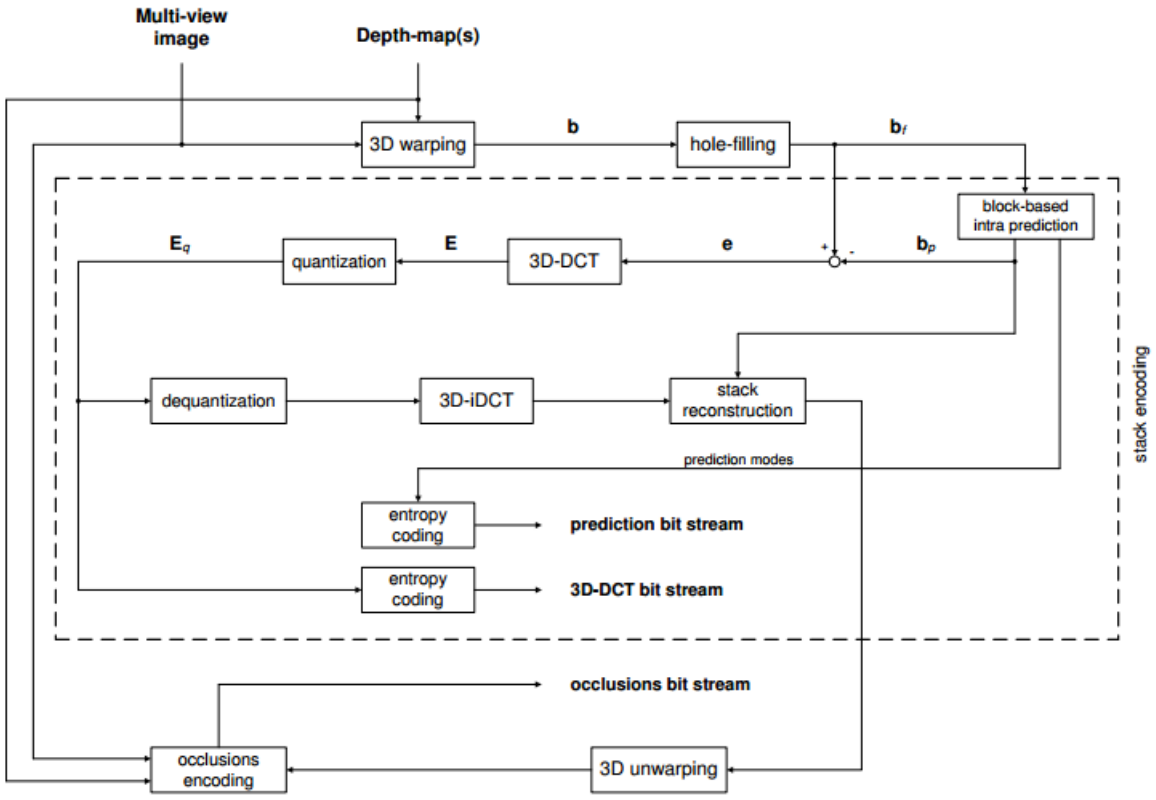


FIGURE III.1.23 – Diagramme correspondant au processus d'encodage d'une frame multi-vues.

génération du volume 3D contenant les différentes vues, le codage de ce volume 3D ainsi que le système de gestion des occultations mis en place.

Chaque vue V_i ,$i = 0, ..., N-1$, est projetée sur une vue de référence V_{ref} grâce aux différents paramètres associés à chaque caméra i et aux cartes de profondeurs afin de produire les N vues $V_{i \to ref}$. Les vues $V_{i \to ref}$ sont ensuite empilées afin de former un volume 3D de profondeur N (voir figure III.1.24).

Les vues projetées contiennent des pixels dont l'intensité lumineuse n'est pas définie, situés soit sur les bords de l'image soit aux localisations des zones d'occultations (zones vertes et rouges sur la figure III.1.24). L'auteur propose alors de combler ces différentes zones par interpolation linéaire via un processus désigné par le terme "layer-filling" : pour chaque emplacement (x, y) du volume 3D, on considère les N échantillons associés à celui-ci. Les valeurs non définies sont ensuite déduites par interpolation linéaire sur les deux valeurs voisines les plus proches. Dans le cas où un seul voisin est défini (il existe toujours au moins un voisin défini car la tranche associée à la vue V_{ref} n'a pas subi de projection), cette valeur est tout simplement répétée. La figure III.1.25 illustre ce processus dans les deux cas présentés précédemment.

Le volume 3D est ensuite découpé en blocs de taille $8 \times 8 \times N$, et chaque bloc, appelé b dans la suite, subit le même traitement. Celui-ci est tout d'abord prédit en utilisant le mécanisme d'intra-prédiction du standard H.264/AVC, décrit dans la section II.1.3.3.1, sur chacune de ses

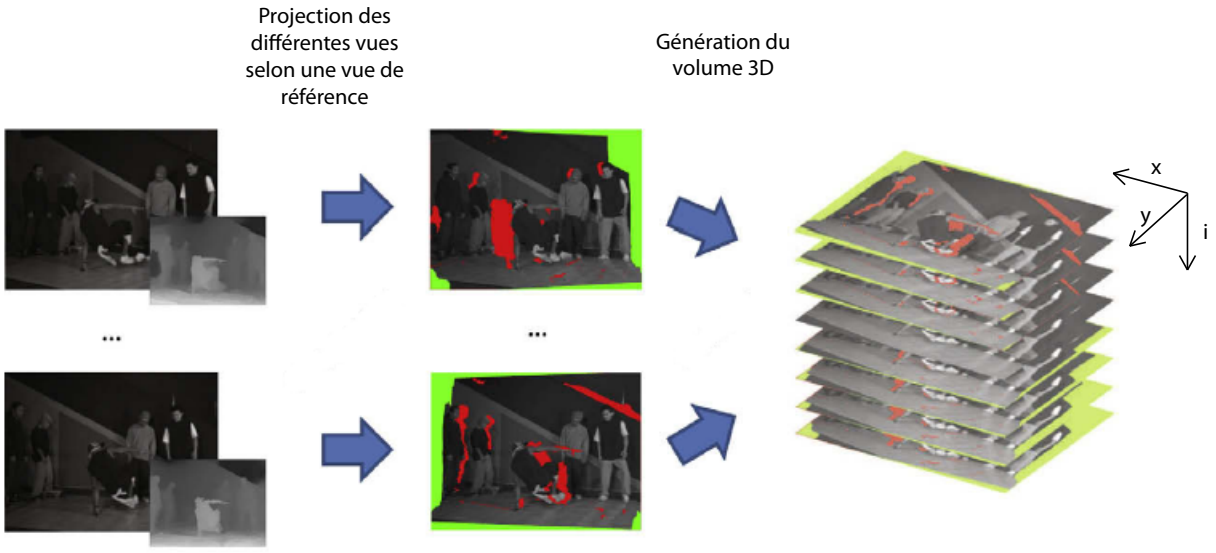


FIGURE III.1.24 – Projection des vues V_i et construction du volume 3D.

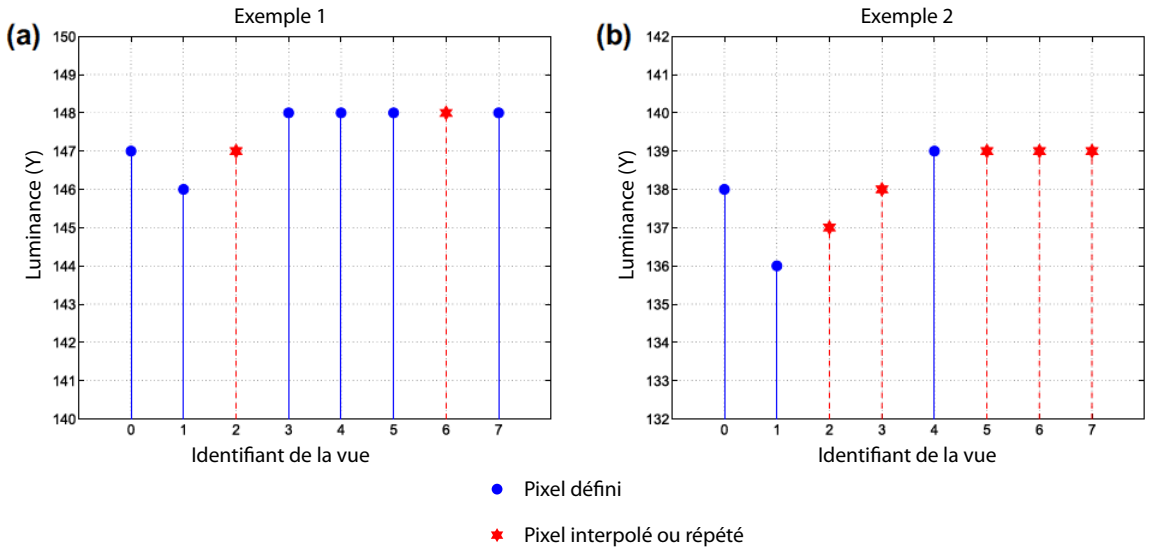


FIGURE III.1.25 – Deux exemples du processus de "layer filling".

tranches (une tranche désigne chacun des N plans de taille 8×8 constituant le bloc b). Le mode de prédiction minimisant l'erreur sur le bloc tout entier est ensuite sélectionné et le bloc b_p prédit est généré (les modes de prédiction sont codés à l'aide du codeur arithmétique CABAC de H.264/AVC). On en déduit ainsi le bloc e correspondant à l'information résiduelle où $e = b - b_p$. On applique ensuite une DCT 3D séparable afin de produire le bloc transformé E. Ce bloc E subit ensuite une étape de quantification : chaque emplacement (x, y, i) $x, y = 0, ..., 8$ et $i = 0, ..., N-1$ du bloc est quantifié en utilisant le pas de quantification $Q(x, y, i)$ définie par l'équation III.1.1 (cette formule est dérivée du standard H.264/MVC et permet d'utiliser des valeurs du paramètre de quantification QP entre 0 et 51). On désignera le bloc quantifié par le terme E_q.

$$Q(x, y, i) = [0.69 \cdot 2^{\frac{QP}{6}} \cdot \Delta_{\max\{x,y,i\}}] \tag{III.1.1}$$

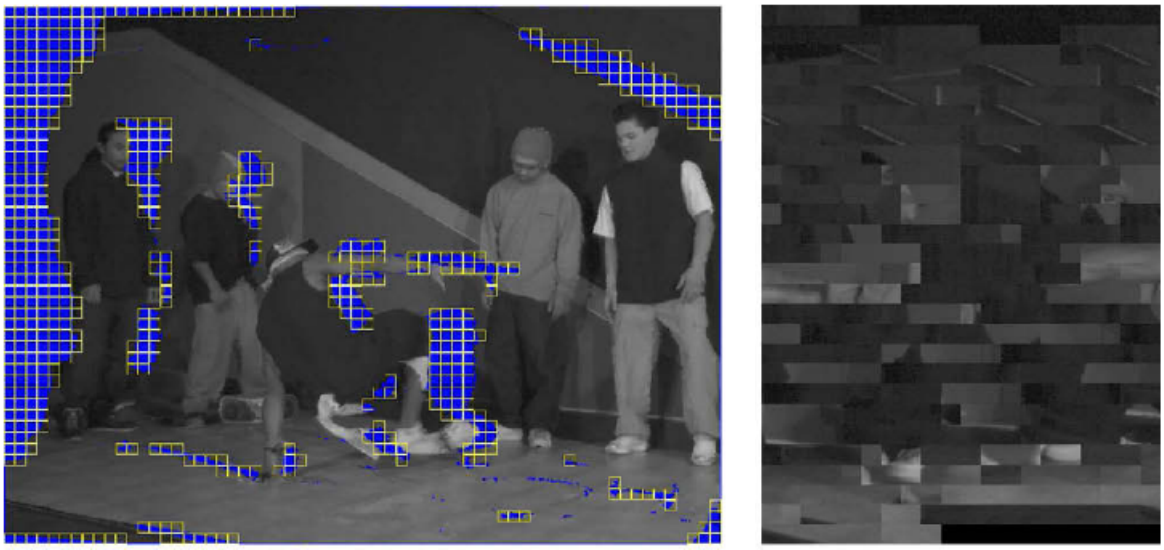
où $\Delta = (9, 16, 19, 22, 26, 27, 29, 34)/8$.

Le bloc E_q est ensuite scanné afin de produire un vecteur 1D contenant les différents coefficients quantifiés. L'approche utilisée pour le parcours du bloc 3D est celle présentée par Chan et Lee (1997) ("Shifted complement hyperboloid"). Les différents coefficients sont ensuite rassemblés en couple $(zeroRun, Size)$ comme pour le format JPEG (voir section II.1.2.1.1) puis codés en utilisant le codeur arithmétique CABAC afin de produire le train de bits associé au volume 3D.

La gestion des occultations de la scène est réalisée de la manière suivante : le volume 3D issu de l'étape de quantification subit l'ensemble des traitements associés à la décompression : déquantification/DCT 3D inverse/reconstruction à partir de l'information associée à l'intra-prédiction/re-projection 3D selon leur position initiale. Une fois re-projetées, les différentes vues V_i' présentent diverses zones d'occultations qui doivent être comblées. Pour cela, l'auteur adopte une approche adaptative. L'ensemble de la frame reconstruite est découpée en blocs de taille 16×16 et on compte combien de pixels sont non définis dans chaque bloc. Lorsque ce nombre de pixels dépasse un certain seuil (fixé à 36 par l'auteur), on va alors récupérer le bloc correspondant dans la vue d'origine V_i et on le stocke dans une image désignée par le terme "Macroblock image". La figure III.1.26(a) présente une frame V' re-projetée contenant des zones d'occultations : chaque carré désigne un bloc de taille 16×16 dont le nombre de pixels non définis (en bleu) dépasse le seuil. La figure III.1.26(b) présente la "macroblock image" associée. Sinon, lorsque le nombre de pixels dans le bloc est inférieur au seuil, les pixels sont déduits par interpolation bi-linéaire sur leurs voisins durant la décompression.

Afin d'optimiser le nombre de macro-blocs à coder, les localisations des macro-blocs dans les images suivantes sont re-projetées sur les images déjà traitées par le processus de gestion des occultations afin de vérifier que celui-ci n'a pas déjà été inclus dans une "Macroblock image" antérieure. Grâce à cette optimisation, le nombre de blocs à coder est réduit considérablement au fur et à mesure du processus de gestion des occultations. Chaque "macroblock image" est ensuite codée en utilisant le mode intra de H.264/AVC pour produire le train de bits associé aux occultations.

Cet algorithme est comparé au standard de compression multi-vues MVC. Celui-ci obtient de bons résultats en terme de débit/distorsion. On observe notamment un gain de 1 dB du PSNR



(a) (b)

FIGURE III.1.26 – (a) Frame V' re-projetée, les zones bleues correspondent aux zones d'occultations, les carrés correspondent aux macro-blocs à inclure dans la "macroblock image", (b) "macroblock image" associée.

pour des fichiers de taille égale, lorsque le nombre de bits par pixel est inférieur à 0.015. Toutefois, cet algorithme présente plusieurs inconvénients : tout d'abord les paramètres (intrinsèques et extrinsèques) des caméras et/ou les cartes de profondeur doivent être disponibles. De plus, l'auteur ne discute pas des possibles schémas de codage concernant les cartes de profondeur.

D'autres approches sont possibles pour la compression multi-vues : Garbas *et al.* (2007) propose une méthode basée sur les ondelettes 4D et sur le filtrage de vues à compensation de disparité (DCVF) afin d'exploiter la corrélation temporelle, inter-vues et spatiale spécifique aux médias multi-vues. Une autre alternative réside dans les techniques de compression multi-vues distribuées (Ouaret *et al.* (2006), Guo *et al.* (2006)) qui encodent chaque frame de la séquence multi-vues à partir de la corrélation existant entre les différentes vues. Ces méthodes ont l'avantage de réduire la complexité au niveau de l'encodeur, les processus tels que la compensation de mouvement étant opérés au niveau du décodeur.

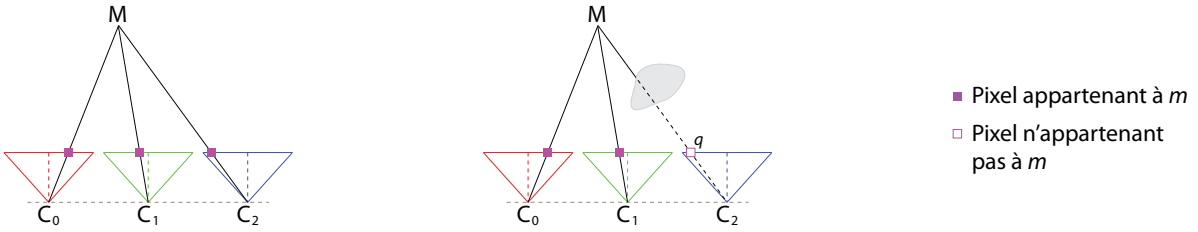
Chapitre III.2

Notre approche LDI

Dans cette section, nous allons présenter la méthode mise au point pour générer une LDI (Layered-Depth Image) à partir d'un jeu d'images multi-vues donné. Cette section est composée de deux parties principales : la première partie décrit la phase d'estimation des disparités qui va permettre, à partir de l'image multi-vue en entrée (n vues), de générer n cartes de disparités. La seconde partie démontre comment, en utilisant les n vues d'origine ainsi que les n cartes de disparités précédemment générées, on peut extraire uniquement l'information non redondante via la phase d'extraction des layers. Les travaux décrits dans ce chapitre sont issus d'une collaboration avec M. Cédric Niquin dont la thèse, effectuée elle aussi dans le cadre du projet ANR "Cam-Relief", portait notamment sur la génération de cartes de disparité. Ces cartes de disparités sont ensuite exploitées pour la génération de notre LDI. Le lecteur pourra se référer à Niquin *et al.* (2009, 2010a,b), Niquin (2011) pour plus d'informations concernant les algorithmes dédiés à la génération des cartes de disparité.

III.2.1 Estimation des disparités

Les difficultés majeures rencontrées avec la plupart des méthodes de compression basées LDI sont majoritairement dues à des problèmes d'homogénéité au sein des cartes de disparité (plus précisément des zones d'occlusions), d'imprécisions et d'erreurs d'arrondis. Dans cette section, nous présentons notre méthode d'estimation des disparités qui tire parti de la géométrie simplifiée de notre périphérique de capture : les lignes épipolaires étant horizontales, les pixels homologues (pixels correspondant au même point 3D dans les différentes vues) ont donc la même ordonnée dans le système de coordonnées propre à chaque vue. Notre méthode, qui permet de générer des cartes de disparités homogènes, utilise des valeurs entières afin d'écarter toute ambiguïté mais aussi de permettre une forte détection des redondances. Dans la suite, nous présentons deux algorithmes, tous les deux basés sur les notions de matchs et de partitions. Attachons-nous d'abord à décrire ces différentes notions avant de passer à la description des deux algorithmes d'estimation des disparités.



(a) Sans occlusion. (b) Avec occlusion.

FIGURE III.2.1 – Illustration d'un point 3D M et de son match m associé.

III.2.1.1 Notions élémentaires

Toutes les notions de cette section sont essentielles pour une bonne compréhension de notre approche basée LDI. Pour plus de détails concernant celles-ci, le lecteur pourra se référer à la section III.1.4 intitulée "Mise en correspondance multi-vues des pixels" de Niquin (2011). Considérons P comme étant l'ensemble des pixels contenus dans l'image multi-vues en entrée. Par $p = (i, x_i, y_i)$, on définit le pixel provenant de la vue i et ayant pour coordonnées (x_i, y_i). Un "match" m est défini comme étant l'ensemble des pixels issus de la projection d'un même point 3D M. La figure III.2.1(a) illustre le point M ainsi que les pixels appartenant au match m issus de la projection de M sur les différentes vues.

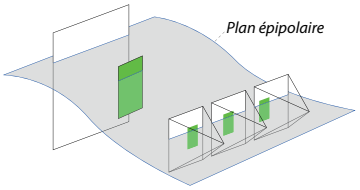
Soit (x_0, y_0) les coordonnées du pixel appartenant à m et provenant de la vue 0. Grâce à notre géométrie simplifiée, les coordonnées des autres pixels de m dans leur base associée sont facilement calculables. Plus précisément, la position du pixel homologue dans la vue k est donnée par l'équation suivante :

$$\begin{pmatrix} x_k \\ y_k \end{pmatrix} = \begin{pmatrix} x_0 - k \times m_d \\ y_0 \end{pmatrix}, \tag{III.2.1}$$

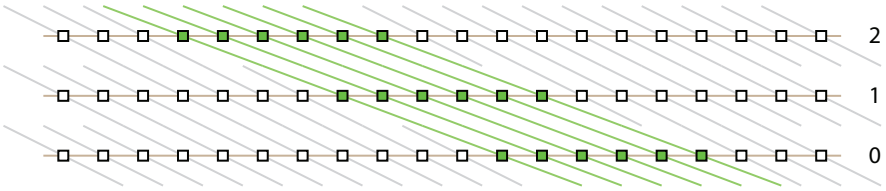
où m_d est la valeur de disparité entière associée au match m.

L'utilisation de valeurs de disparités entières nous permet de nous assurer que $x_0 - k \times m_d$ est une coordonnée entière, et ainsi d'éviter à avoir à gérer une précision de l'ordre du sous-pixel qui est généralement la cause de toutes les ambiguïtés présentes durant la phase d'extraction des layers. On définit, de manière conventionnelle, $m_x = x_0$ et $m_y = y_0$ comme étant la position de la projection du point 3D M dans la vue 0. Un pixel p appartient au match m associé à M seulement si la condition suivante est satisfaite :

$$\begin{aligned} x_i &= m_x - i \times m_d \\ &\text{et} \\ y_i &= m_y. \end{aligned} \tag{III.2.2}$$



(a) Scene.



(b) Le graphe de matchs.

FIGURE III.2.2 – Une scène 3D et son graphe de matchs associé.

Bien qu'un match soit toujours constitué d'au moins un pixel, il peut arriver que celui-ci ne contienne pas de pixel issu d'une vue spécifique lorsqu'une occlusion se produit sur celle-ci. Sur la figure III.2.1(b), le point M est occulté par l'objet gris et n'est donc pas visible sur la vue 2. Ainsi, le pixel $q = (2, m_x - 2 \times m_d, m_y)$ de la vue 2 ne correspond pas à la projection de M et ne fait donc pas partie du match m même si celui-ci satisfait l'équation III.2.2. Dans ce cas, on dit que m est visible dans les vues 0 et 1, mais pas dans la vue 2.

Le but de notre méthode d'estimation des disparités consiste à trouver une partition de tous les pixels (P) des n vues en matchs. Cette partition doit être constituée d'un ensemble de matchs non vides (contenant au moins un pixel) et n'ayant aucun pixel en commun entre eux. De plus, l'union de tous les matchs appartenant à la partition doit être égale à l'ensemble de tous les pixels de départ P. Une partition g doit vérifier les conditions suivantes :

$$\begin{array}{c} \forall m \in g, \quad m \neq \emptyset \\ \forall m, n \in g, \quad m \cap n = \emptyset \\ \bigcup_{m \in g} m = P \end{array} \tag{III.2.3}$$

Grâce à cette notion de match, notre méthode d'estimation des disparités nous permet de définir de manière exacte les pixels homologues (et associés à de l'information redondante) au sein de l'image multi-vues. Avant de présenter nos deux algorithmes dédiés à l'estimation des disparités, il convient de définir la notion de "graphe de matchs". Un graphe de matchs constitue une représentation graphique d'une ligne épipolaire spécifique contenant les pixels des N vues (chaque pixel étant assimilé à un nœud du graphe), ainsi que les matchs de la partition associée (assimilés, eux, aux arcs du graphe). La figure III.2.2(a) illustre une scène 3D de base constituée d'un fond blanc et d'un objet vert au premier plan. La figure III.2.2(b), quant à elle, décrit le graphe de matchs correspondant.

Chaque ligne horizontale correspond à la ligne épipolaire d'une vue (résultant de l'intersection entre le plan épipolaire et la vue en question), les carrés représentent les pixels et les lignes non-

horizontales symbolisent les matchs qui relient les pixels homologues. Les pixels colorés en vert sont issus de la projection de l'objet 3D vert. A partir de la figure III.2.2(b), on peut voir que les matchs représentés en vert sont plus inclinés que ceux en gris, signifiant ainsi que leur valeur de disparité est supérieure (leur déplacement d'une image à l'autre est plus important), ceci étant dû au fait qu'ils correspondent à l'objet vert situé plus près de la caméra (effet parallaxe). Ce type de représentation est utilisé dans la suite du document (notamment pour la partie concernant l'extraction des layers), afin de pouvoir illustrer de manière efficace nos propos.

L'autre élément clé de notre approche d'estimation des disparités est la mise en place d'une contrainte d'homogénéité. Cette contrainte, définie de manière plus précise dans Niquin *et al.* (2010b), permet de s'assurer qu'un point 3D n'est pas visible dans une vue que s'il existe un autre point 3D placé devant lui et qui l'occulte. Traduite en terme de disparité, cette contrainte signifie qu'un match ne contient pas le pixel d'une vue que si celui-ci appartient à un match ayant une valeur de disparité supérieure. Cette contrainte va permettre de fournir une grande robustesse à notre algorithme d'estimation des disparités et est essentielle lors de la phase de reconstruction des vues d'origines durant la décompression (voir section III.2.3). Dans les sections suivantes, nous présentons nos deux algorithmes d'estimation des disparités dédiés à la construction d'une partition.

III.2.1.2 Estimation des disparités

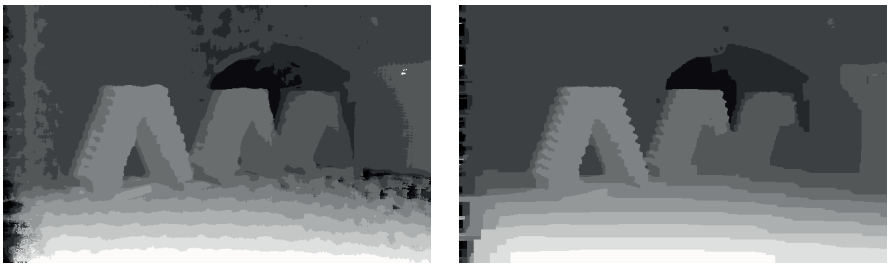
III.2.1.2.1 L'algorithme Far-Near

La première phase de l'algorithme consiste à initialiser la partition avec des matchs ayant la valeur de disparité la plus petite possible (correspondant aux points 3D provenant du fond de la scène). Puis, les différentes valeurs de disparité sont parcourues de manière croissante. Pour chaque valeur de disparité α, on considère tous les matchs associés à α et l'on évalue ensuite si ceux-ci sont valides (i.e. si le point 3D correspondant appartient effectivement à la scène). Cette évaluation est faite de manière locale (par match) via la minimisation d'une fonction de coût de type SSAD (Sum of the Sum of Absolute Differences). C'est grâce à cette minimisation de coût que l'algorithme va permettre de prendre en charge le bruit pouvant être présent dans les N vues. Le lecteur pourra se référer au chapitre IV.1 de Niquin (2011) pour plus de précisions concernant l'algorithme Far-Near.

Ce processus d'estimation des disparités peut être perturbé par deux problèmes majeurs : les réflexions lumineuses et les régions uniformes de l'image. Nos algorithmes sont basés sur l'hypothèse que la scène est Lambertienne (un point 3D d'une scène conserve la même intensité lumineuse quelque soit le point de vue, i.e. pas de réflexions spéculaires). Quand une réflexion spéculaire se produit, la fonction de coût (SSAD) pour le match actuellement évalué va être perturbée et donc générer une valeur supérieure à ce qu'elle devrait être. Ceci résultant à considérer le match actuel comme non-existant. De même, dans le cas de grandes régions uniformes, deux matchs associés à des valeurs de disparité différentes peuvent produire le même coût. Ces deux phénomènes produisent certaines ambiguïtés dans la phase de sélection du match, et ainsi produisent des valeurs de disparité pouvant différer de leur valeur réelle. Toutefois, le processus d'estimation des disparités étant dédié à la compression via la dé-corrélation spatiale des

(a) Initialisation. (b) $\alpha = 3$. (c) $\alpha = 6$. (d) $\alpha = 10$.

FIGURE III.2.3 – Exécution de l'algorithme Far-Near d'estimation des disparités.



(a) Algorithme Far-Near. (b) Algorithme Near-Far.

FIGURE III.2.4 – Deux cartes de disparité générées en utilisant les deux algorithmes.

différentes vues, la mise en correspondance des différents pixels est quand même effectuée et l'occurrence de valeurs de disparités erronées n'affecte que de manière insignifiante le processus de compression en aval.

L'évolution de la partition au fur et à mesure de l'algorithme Far-Near est illustrée par le nuage de points 3D pour plusieurs valeurs de disparité α (voir figure III.2.3). On peut remarquer que l'algorithme commence par considérer tous les points de l'arrière-plan, puis rajoute des points appartenant aux plans plus proches des caméras (correspondant à des matchs de disparité supérieure) pour construire la scène finale. L'image multi-vues de la figure III.2.3 est constituée de 8 photographies prises dans les caves Ruinart [1], et générant des valeurs de disparité allant de 0 à 10. L'algorithme III.2.1 contient le pseudo-code de l'algorithme Far-Near et la figure III.2.4(a) illustre la carte de disparité produite par l'algorithme Far-Near et associée à la première vue du jeu de donnée "Ruinart".

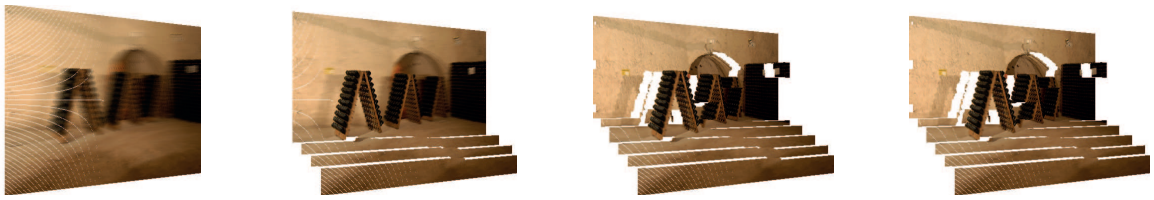
Cet algorithme retourne une partition g (i.e. un ensemble de matchs), contenant tous les pixels de l'image multi-vues (P). Chaque match étant caractérisé par une valeur de disparité α spécifique et chaque pixel n'appartenant qu'à un unique match, les différentes cartes de disparité peuvent être générées en associant chaque valeur de disparité à un niveau de gris spécifique.

1. www.ruinart.com

Algorithme III.2.1 L'algorithme Far-Near.

Entrées:
P : Tous les pixels de l'image multi-vues
F : Une fonction de coût (typiquement SSAD)
α_{min} : Valeur de disparité minimale (si non disponible $\alpha_{min} = -127$)
α_{max} : Valeur de disparité maximale (si non disponible $\alpha_{max} = 127$)
Sorties:
$Part$: La partition contenant tous les pixels de l'image multi-vues
Données:
α : La valeur de disparité courante
M_α : L'ensemble des matches de disparité α de P
$m_{i,\alpha}$: Le $i^{\text{ième}}$ match de disparité α
$C_{i,\alpha}$: Le coût associé au match $m_{i,\alpha}$
$C_{i,\alpha_{optimal}}$: Le coût miminum associé au match $m_{i,\alpha_{optimal}}$

début
$\alpha \longleftarrow \alpha_{min}$
considérer M_α de P
$Part \longleftarrow \oslash$
pour tout $m_{i,\alpha} \in M_\alpha$ **faire**
    $C_{i,\alpha} \longleftarrow F(m_{i,\alpha})$
    $C_{i,\alpha_{optimal}} \longleftarrow C_{i,\alpha}$
    $Part \longleftarrow Part \cup m_{i,\alpha}$
fin pour
pour $\alpha \longleftarrow \alpha_{min} + 1$ **jusqu'à** α_{max} **faire**
    considérer M_α de P
    pour tout $m_{i,\alpha} \in M_\alpha$ **faire**
        $C_{i,\alpha} \longleftarrow F(m_{i,\alpha})$
        si $C_{i,\alpha} < C_{i,\alpha_{optimal}}$
            $C_{i,\alpha_{optimal}} \longleftarrow C_{i,\alpha}$
            $Part \longleftarrow Part \cup m_{i,\alpha}$
        fin si
    fin pour
fin pour
retourner $Part$
fin



(a) Initialisation. (b) $\alpha = 6$. (c) $\alpha = 3$. (d) $\alpha = 0$.

FIGURE III.2.5 – Exécution de l'algorithme Near-Far d'estimation des disparités.

III.2.1.2.2 L'algorithme Near-Far

L'algorithme Near-Far marche de manière opposée à l'algorithme Far-Near : la partition est tout d'abord initialisée avec des matchs correspondant à la valeur de disparité maximale, puis ces valeurs de disparité sont parcourues de manière décroissante. Pour chaque valeur de disparité α, on considère tous les matchs associés déjà présents dans la partition et on vérifie leur existence. Si un match est considéré comme non-existant, celui-ci est supprimé de la partition et est substitué (de manière implicite) par un match de disparité $\alpha - 1$ qui devient alors visible. Au cours du temps, la partition g évolue conformément à la figure III.2.5 : le nuage de points est placé sur le premier plan de la scène (correspondant à la valeur de disparité maximale) puis se déplace vers le fond afin de construire la scène.

Cette fois, le sens de parcours des valeurs de disparité ne permet pas à lui tout seul de satisfaire la contrainte d'homogénéité. Pour cela, on utilise un algorithme de type "coupure de graphe" pour la phase de décision d'existence d'un match. Grâce à notre sens de parcours des valeurs de disparité (décroissant), les cartes de disparité ainsi générées contiennent moins d'artefacts et la détection des occlusions est réalisée de manière plus précise qu'avec l'algorithme Far-Near grâce à l'ajout de mécanismes supplémentaires d'affinement des disparités. Le lecteur pourra trouver tous les détails associés à cette méthode dans la section V.1.3 de Niquin (2011). La figure III.2.4(b) présente la carte de disparité ainsi générée et associée à la première vue du jeu de données "Ruinart".

III.2.2 Extraction des différents layers

Dans cette section, nous allons présenter le processus d'extraction des différents layers constituant la LDI (Layered-Depth Image) à partir des vues d'origine et des cartes de disparité générées durant la phase d'estimation des disparités. Nous faisons remarquer au lecteur que notre définition de la LDI est légèrement différente de celle décrite dans la littérature (voir la section "État de l'art"). Dans notre cas, chaque layer est associé à une vue (contrairement à l'approche standard les associant à une vue de référence) et leur nombre est toujours N. De plus, comme on pourra le constater sur la figure III.2.7, on peut constater que la ratio de pixels conservés reste relativement constant (à part pour le premier qui contient toujours 100% des pixels).

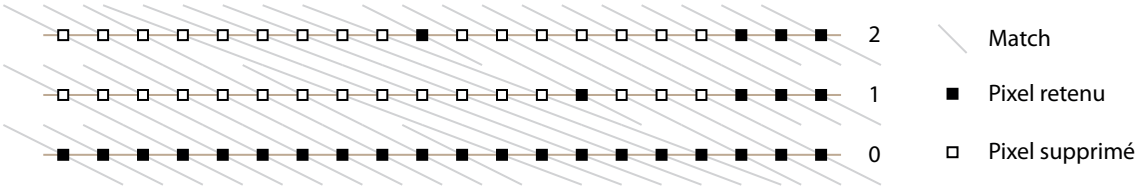


FIGURE III.2.6 – Le graphe de matchs après la phase d'extraction.

(a) Layer 0. (b) Layer 2. (c) Layer 4. (d) Layer 6.

FIGURE III.2.7 – Différents layers résultant du processus d'extraction sur le jeu de données "Ruinart".

La notion de matchs, précédemment définie, nous a permis de mettre en évidence de manière exacte les redondances spatiales au sein de l'image multi-vues en entrée. En effet, tous les pixels appartenant à un même match résultent de la projection d'un unique point 3D M de la scène et sont donc à juste titre considérés comme redondants. Afin de supprimer cette redondance, il suffit de garder un pixel uniquement de chaque match de la partition g (de manière arbitraire, celui issu de la vue 0) et de supprimer les autres. La figure III.2.6 illustre le processus d'extraction sur le graphe de matchs précédemment utilisé dans la figure III.2.2(b), les pixels noirs correspondent au pixel conservé au sein de chaque match et les blancs à ceux qui sont supprimés. Ainsi, une fois la phase d'extraction des layers effectuée, on obtient N layers. Le premier layer contient tous ses pixels, le deuxième contient uniquement les pixels ne pouvant être déduits à partir du premier layer, le troisième contient uniquement les pixels ne pouvant être déduits du premier ni du second layer et ainsi de suite.

Afin de réduire les artefacts exposés dans la section présentant l'algorithme Far-Near (variations lumineuses d'une vue à une autre dues aux réflexions spéculaires, mais aussi le bruit), la couleur du pixel gardé pour chaque match est remplacée par la couleur moyenne de tous les pixels appartenant au match. Ceci expliquant pourquoi le choix du pixel gardé est totalement arbitraire, décider d'en choisir un autre résultera de toute façon à la même moyenne. La figure III.2.7 illustre différents layers obtenus après le processus d'extraction effectué sur le jeu de données "Ruinart".

III.2.3 Reconstruction des vues d'origine

Dans cette section, nous présentons le processus de reconstruction des N vues d'origine à partir des N layers et cartes de disparité associées. Celui-ci consiste principalement à distribuer les pixels constituant chaque layer dans les autres vues et ce, conformément à leur valeur de

(a) $\alpha = 3$. (b) $\alpha = 5$. (c) $\alpha = 7$. (d) $\alpha = 10$.

FIGURE III.2.8 – Reconstruction des vues d'origine à partir des différents layers.

disparité, afin de reconstituer la même partition g que celle générée après la phase d'estimation des disparités. La difficulté majeure durant ce processus consiste à trouver dans quelles vues un match donné est visible et dans celles où il n'y est pas. Toutefois, notre contrainte d'homogénéité (utilisée durant la phase d'estimation des disparités et définie en section III.2.1.1) permet de s'assurer qu'un match n'est pas visible dans une vue seulement si celui est occulté par un match de disparité supérieure. Grâce à cette contrainte, la partition g peut être reconstituée de manière exacte en utilisant l'algorithme du peintre Newell *et al.* (1972).

L'algorithme du peintre (utilisé en synthèse d'images avant l'apparition de techniques telles que le Z-buffer), permet d'effectuer le rendu d'une scène 3D donnée en triant les différents polygones appartenant aux objets constituant la scène selon leur profondeur (du plus éloigné au plus proche). Ainsi le problème des occlusions est pris en compte par superposition lors des projections 2D. De manière analogue, cet algorithme peut être appliqué pour la reconstruction des vues d'origine.

L'idée est donc de distribuer les différents pixels de chaque layer dans les N vues par ordre croissant de disparité. Certains pixels vont donc être dessinés au dessus d'autres, reproduisant ainsi les différentes occlusions de la scène. La figure III.2.8 illustre le processus de reconstruction d'une vue donnée à partir des différents layers de la figure III.2.7, et ceci, pour plusieurs valeurs de disparités.

Nous savons maintenant, à partir d'une image multi-vues donnée, générer un LDI ainsi que reconstruire les vues d'origine à partir de celui-ci lors de la décompression. La prochaine section va maintenant se pencher sur les différents algorithmes possibles pour la compression du LDI.

Chapitre III.3

Compression du LDI

La section précédente présentait le processus de génération du LDI (Layered-Depth Image) à partir des N vues d'origine. Cette opération se passe en deux temps : tout d'abord, une phase d'estimation des disparités qui permet d'une part d'obtenir les cartes associées pour chaque vue et d'autre part d'associer les pixels homologues dans les différentes vues, puis, une phase d'extraction des résidus pendant laquelle l'information considérée comme redondante durant la première phase est supprimée. On dispose maintenant de deux types d'informations : le LDI à proprement parler (une texture RGB contenant N layers) ainsi que les cartes de disparités associées (texture 8-bits composée elle-aussi de N layers). La section suivante va présenter le schéma de compression adopté pour compresser ces deux textures, le LDI dans un premier temps, puis la texture de disparité dans un second temps.

III.3.1 Compression de l'information chromatique du LDI

III.3.1.1 Algorithme basé DCT

Afin de compresser la texture chromatique de la manière la plus homogène possible, nous avons choisi d'adopter un schéma de compression basé DCT pour les N layers. D'un côté, les pixels appartenant au layer de référence présentent une corrélation spatiale qui doit être exploitée ; de l'autre côté, les pixels appartenant aux layers résiduels présentent une corrélation spatiale plus faible, mais on peut constater qu'il existe une certaine redondance d'un layer à l'autre. L'approche qui nous apparaît donc la plus adaptée consiste à compresser le layer de référence en utilisant une DCT bi-dimensionnelle et à exploiter les deux types de redondance présents dans les layers résiduels via l'utilisation d'une DCT 3D (voir figure III.3.1).

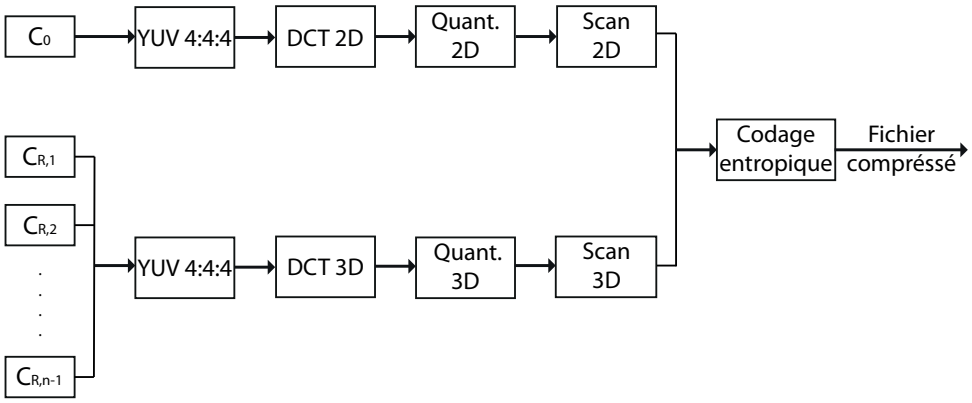


FIGURE III.3.1 – Présentation du système de compression. C_0 représente le layer de référence et $C_{R,i}$ le i^{ieme} layer résiduel.

III.3.1.1.1 Conversion colorimétrique RGB vers YCbCr

Tout d'abord la texture chromatique RGB est convertie dans l'espace de couleur YCbCr en utilisant la transformation décrite dans la section II.1.2.1.1. Nous choisissons de ne pas effectuer de sous-échantillonnage sur les deux composantes Cb et Cr pour les raisons suivantes :

- Si le sous-échantillonnage est effectué en amont du processus de génération de LDI, la mise en correspondance pixel à pixel pourrait être gênée par la distorsion occasionnée sur le signal.
- Si celui-ci est effectué en aval, il va falloir rajouter de l'information autour des pixels appartenant aux layers résiduels afin d'aligner ceux-ci sur la grille de sous échantillonnage (typiquement des blocs de 4×4 pixels dans le cas d'un sous-échantillonnage 4 :2 :0).

III.3.1.1.2 Passage dans le domaine fréquentiel

Comme nous l'avons dit plus haut, notre texture chromatique du LDI est composée de N layers : un layer de référence contenant 100% de ses pixels ainsi que $N-1$ layers résiduels. Nous choisissons de considérer le premier layer comme une image 2D standard (ses pixels présentent une corrélation spatiale et la DCT ne sera pas perturbée par des pixels manquants), en appliquant la DCT bi-dimensionnelle définie par l'équation II.1.14 (section II.1.2.1) sur celui-ci. Afin d'optimiser le temps d'exécution des DCT nous utilisons la méthode optimisée définie par Chen *et al.* (1977).

Pour chaque layer résiduel, nous choisissons dans un premier temps de décaler horizontalement tous les pixels (voir la figure III.3.2) afin de minimiser le nombre de blocs 3D à coder par la suite. Les $N-1$ layers résiduels sont ensuite groupés afin de former un volume 3D contenant 8 tranches (si $N < 8$ le cube est rempli avec des tranches blanches), et une DCT-3D est appliquée sur chaque bloc de taille $8 \times 8 \times 8$. La profondeur du cube a été fixée à 8 afin de permettre l'utilisation du développement de Chen *et al.* (1977) sur la troisième dimension.

(a) Avant le décalage. (b) Après le décalage.

FIGURE III.3.2 – Décalage horizontal vers le bord gauche des pixels d'un layer résiduel de la séquence "Ruinart".

III.3.1.1.3 Quantification

Une fois la DCT effectuée, on doit maintenant quantifier les différents coefficients provenant des blocs 2D (de taille 8×8 et issus du layer de référence) et des blocs 3D (de taille $8 \times 8 \times 8$ et issus, eux, des layers résiduels). Le processus de quantification devra procéder de façon similaire pour les cas 2D et 3D. C'est en effet celui-ci qui génère les distorsions dans le signal d'entrée et l'information devant être ré-injectée dans les différents layers durant la décompression doit être relativement homogène. Pour cela, nous avons choisi d'adopter la méthode proposée par Zamarin *et al.* (2010) pour calculer les différents coefficients de quantification $Q_{i,j}$ et $Q_{i,j,k}$:

$$Q_{i,j} = 0.69 \times 2^{\frac{q}{6}} \times \Delta_{max(i,j)} \tag{III.3.1}$$

$$Q_{i,j} = 0.69 \times 2^{\frac{q}{6}} \times \Delta_{max(i,j,k)} \tag{III.3.2}$$

où $i, j, k \in [0, 8]$, q est le facteur de quantification (de 1 à 50, 1 pour la meilleure qualité) et $\Delta = \{1, 2, \frac{19}{8}, \frac{22}{8}, \frac{26}{8}, \frac{27}{8}, \frac{29}{8}, \frac{34}{8}\}$. Chaque coefficient (2D et 3D) est ensuite divisé par sa valeur de quantification associée puis arrondi à l'entier le plus proche.

III.3.1.1.4 Processus de parcours

Après la phase de quantification, chaque bloc 2D et 3D doit être parcouru afin de former des buffers 1D de façon à maximiser les chaînes de coefficients nuls pour le codage entropique. Ce processus est généralement effectué en parcourant le bloc (2D ou 3D) en commençant par les fréquences les plus faibles (on commence par le coefficient DC) pour finir par les plus hautes. Le standard JPEG ISO (a) définit le parcours en zig-zag afin de traiter le problème des blocs 2D. Celui-ci commence par le coefficient DC et balaye ensuite le bloc entier des basses aux hautes fréquences (voir la figure II.1.12).

Pour le cas des blocs 3D, plusieurs approches sont possibles : Yeo et Liu (1995), Chan et Lee (1997) mais aucun standard n'est réellement établi. De manière générale, on peut affirmer que plus la valeur de quantification est élevée, plus la probabilité que le coefficient DCT soit égal à zéro est grande. C'est pourquoi nous avons choisi d'appliquer un tri multi-critères, conformément à l'approche présentée par Chan et Lee (1997), sur les valeurs de quantification afin d'en déduire l'ordre de parcours d'un bloc 3D. On trie tout d'abord ces valeurs de quantification (ainsi que les localisations associées dans le cube) par ordre croissant, puis, si des valeurs de quantification sont égales pour certaines localisations du cube, on trie alors suivant l'axe des x, puis celui des y et enfin celui des z.

III.3.1.1.5 Codage entropique

La dernière étape du processus de compression est le codage entropique. C'est durant cette étape que va être généré le train de bits constituant le fichier compressé. Pour cela nous utilisons une méthode similaire au codage entropique utilisé dans le standard JPEG : de manière générale pour chaque bloc à coder, le coefficient DC est soustrait à celui du bloc précédent puis est codé en utilisant la table de Huffman adaptée (chrominance ou luminance). Les coefficients AC sont ensuite groupés en couples du type $(zeroRun, size)$ selon la procédure décrite dans le paragraphe dédié au codage entropique de JPEG (voir section II.1.2.1.1).

Cette méthode permet de directement prendre en charge le codage entropique des blocs 2D issus du layer de référence. Par contre des modifications dans les tables de Huffman ont dû être apportées afin de pouvoir aussi coder les blocs 3D. En effet, les quatre tables de Huffman (DC luminance, DC chrominance, AC luminance, AC chrominance) fournies par le standard JPEG ne sont pas adaptées à la DCT-3D. Par exemple, la table DC luminance définit l'intervalle $[-2047, 2047]$ conformément à la valeur maximale possible d'un coefficient DC (2040) qui peut être obtenue en appliquant la DCT sur un bloc 8×8 entièrement blanc. Dans le cas d'une transformée 3D, la valeur maximale du coefficient DC (5770) ne peut pas être prise en compte par les tables standards.

Nous avons décidé de garder la structure principale des arbres de Huffman, mais deux codes additionnels ($0C$ et $0D$) ont été ajoutés afin d'autoriser respectivement les intervalles $[-4095, -2048] \bigcup [2048, 4095]$ et $[-8191, -4096] \bigcup [4096, 8191]$, nécessitant ainsi 12 et 13 bits supplémentaires. De la même manière, pour les tables AC, 48 mots hexadécimaux sont ajoutés (de $0B$ à FD). Après avoir réalisé quelques tests sur nos jeux d'images multi-vues, on se rend vite compte que ces codes sont rarement induits dans le processus de codage entropique. Nous avons donc choisi de les placer à la queue de l'arbre. Le nombre de nœuds libres étant inférieur à 48, nous avons ajouté un niveau de plus sur ces nœuds restants afin de les ajouter tous (en les distribuant de gauche à droite et triés selon le deuxième digit du code hexadécimal de manière croissante).

Une fois la texture chromatique du LDI encodée, il faut maintenant compresser les N cartes de disparité qui sont nécessaires à la reconstruction des vues d'origines lors de la phase de décompression. La section III.3.2 présente les caractéristiques majeures à prendre en compte concernant ces cartes de disparité ainsi que l'algorithme adopté.

III.3.1.1.6 Résultats obtenus

Dans cette section, nous présentons les différents résultats obtenus par notre approche basée DCT et les confrontons au standard actuel H.264/AVC. Ces tests ont été menés en utilisant les cinq jeux d'images multi-vues suivants : "bowling", "Cartes", "ruinart", "dolls" et "Rose".

Le processus d'encodage de H.264/AVC a été réalisé à l'aide du logiciel de référence JM 18.0 en considérant chaque vue comme une frame temporelle. Pour cela nous avons choisi d'utiliser deux profils d'encodage distincts : un profil utilisant des frames P ainsi que d'autres fonctionnalités spécifique à H.264/AVC (profil 1) et un profil "intra" (profil 2) qui nous permet de nous assurer de la bonne homogénéité entre chaque vue. Les paramètres d'encodage relatifs à chaque profil sont détaillés dans la table III.3.1.

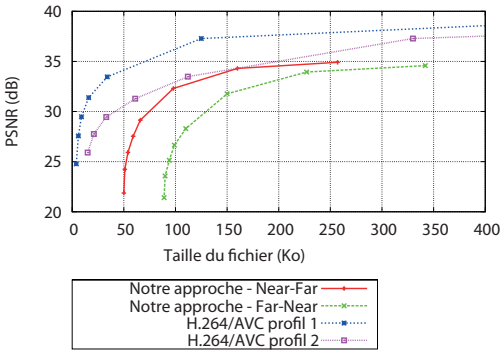
Fonctionnalité	H.264/AVC profil 1	H.264/AVC profil 2
Optimisation Débit/distorsion	Activée	Désactivée
Filtrage anti-blocs	Activé	Désactivé
Codeur entropique	CABAC	CAVLC
Taille des blocs	Adaptatif	8 x 8 fixe
Codage intra uniquement	Désactivé	Activé
Fenêtre de recherche	±32	-
Taille du GOP	4	-
Type du GOP	IPPP	-
Nombre de frames de référence	3	-
Précision de l'estimation de mouvement	Quart de pixel	-

Tableau III.3.1 – Les deux profils utilisés pour l'encodage H.264/AVC.

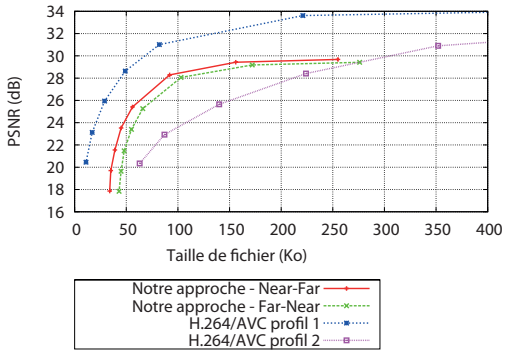
Notre schéma de compression ne sous-échantillonnant pas les composantes de chrominance, on définit le format YUV 4 :4 :4 comme format d'entrée pour H.264/AVC. De manière générale, la littérature rapporte que l'utilisation du sous-échantillonnage sur les composantes de chrominance permet d'obtenir de meilleurs résultats. Toutefois le choix du format YUV 4 :4 :4 est totalement arbitraire et l'utilisation d'autres formats colorimétriques reste toujours possible.

Pour notre approche, les tests ont été conduits en utilisant nos deux méthodes d'estimation des disparités (la méthode Far/Near et la méthode Near/Far décrites respectivement dans les sections III.2.1.2.1 et III.2.1.2.2.

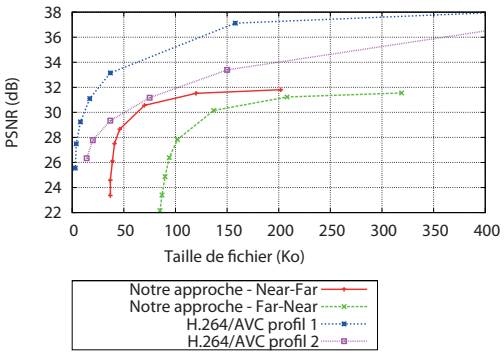
La table III.3.2 présente le pourcentage de pixels résiduels dans chaque layer une fois l'étape d'extraction des layers terminée (toujours en utilisant les deux méthodes Far/Near et Near/Far lors de la phase d'estimation des disparités). La table III.3.3, quant à elle, expose les temps d'encodage (comprenant l'estimation des disparités + l'extraction des layers + la compression) et de décodage (décompression + reconstruction de l'image) pour les cinq jeux de données. Les mesures de temps ont été réalisées sur un ordinateur équipé d'un processeur Intel i7-950, d'un processeur graphique nVidia GTX480 et en fixant le paramètre de quantification q de H.264/AVC à 15. Sur la figure III.3.3, on peut voir les résultats en terme de PSNR pour chacun de nos 5 jeux d'images multi-vues. La taille du fichier comprend à la fois l'information chromatique mais aussi l'information de disparité qui est nécessaire à notre approche pour l'étape de reconstruction.



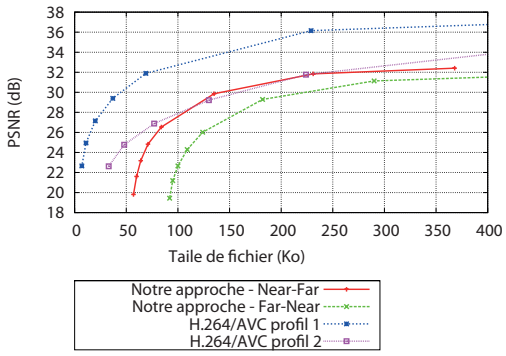
(a) “bowling”



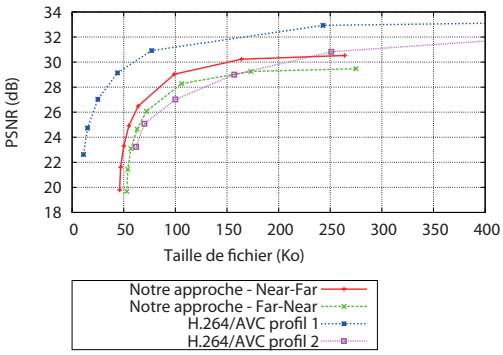
(b) “cartes”



(c) “ruinart”



(d) “dolls”



(e) “rose”

FIGURE III.3.3 – Courbes taille de fichier/distortion pour les cinq datasets utilisés.

	Algorithme Near-Far					Algorithme Far-Near				
	bowling	cartes	ruinart	dolls	rose	bowling	cartes	ruinart	dolls	rose
0	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
1	4.07	1.35	1.69	4.00	1.42	5.19	1.55	3.11	4.84	1.50
2	4.02	1.35	1.68	3.98	1.41	4.81	1.54	2.89	4.51	1.49
3	4.01	1.34	1.67	3.94	1.39	4.71	1.53	2.76	4.41	1.48
4	3.98	1.34	1.68	3.94	1.38	4.60	1.53	2.61	4.37	1.47
5	3.95	1.33	1.67	3.93	1.36	4.52	1.53	2.49	4.32	1.46
6	3.90	1.33	1.67	3.94	1.33	4.41	1.52	2.44	4.26	1.45
7		1.33	1.67		1.30		1.51	2.42		1.42

TABLEAU III.3.2 – Pourcentage de pixels résiduels après l'étape d'extraction des layers pour les cinq jeux de données.

	Processus d'encodage (Near-Far)	Processus d'encodage (Far-Near)	Processus de décodage
bowling	1316.3	60.35	42.35
cartes	492.8	64.45	55.8
ruinart	1010.5	58.9	42.2
dolls	840.6	65.4	56.1
rose	557.9	63.32	54.3

TABLEAU III.3.3 – Temps d'encodage et de décodage (ms) pour chaque jeu de données.

Conformément à la table III.3.2, l'étape d'extraction des layers réduit considérablement le volume des données à compresser pendant l'étape suivante, et comme nous l'avons précisé dans la section III.2.2, chaque layer résiduel contient approximativement la même quantité d'information. A partir de la figure III.3.3, on peut tout d'abord remarquer que notre approche n'est pas capable d'atteindre des valeurs élevées de PSNR. En effet, les approches basées LDI réalisent une association globale de tous les pixels de la scène et supprime ensuite l'information redondante. Contrairement à d'autres méthodes qui utilisent un facteur de quantification, nous ne pouvons pas choisir de garder ou de négliger telle ou telle information, ceci permettant d'obtenir un volume d'information très faible à compresser, mais nous empêchant par la même occasion d'atteindre des valeurs de PSNR élevées. De plus, la méthode d'estimation des disparités quantifie la profondeur et produit de artefacts de coupure sur les objets orientés le long de l'axe des profondeurs (voir figure III.3.4). Ces artefacts impactent de manière forte sur le MSE et consécutivement sur le PSNR.

Notre approche produit toutefois de meilleurs résultats que H.264/AVC (pour le profil 2) sur les deux séquences "Cartes" et "Rose" à faible débit. L'amélioration en terme de PSNR se situe entre +1.0 et +5.0dB pour la séquence "Cartes" et entre +1.0 et +2.0db pour la séquence "Rose". Pour les séquences "Bowling" et "Dolls", l'algorithme Near/Far fournit des performances relativement équivalentes au profil 2 d'H.264/AVC à faible débit, alors que l'algorithme Far/Near est moins qualitatif mais le coût algorithmique de celui-ci est moindre.

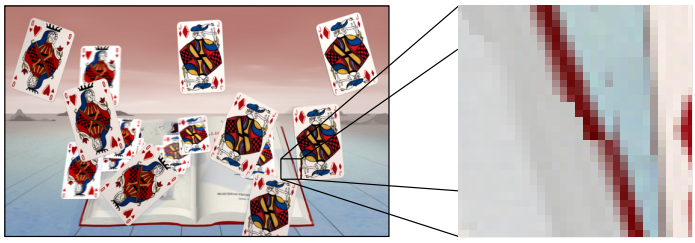


FIGURE III.3.4 – Artefacts de coupure générés par notre méthode d'estimation des disparités.

III.3.1.2 Algorithme basé DWT

Dans cette section, nous présentons la deuxième version de notre algorithme dédié à la compression d'images multi-vues et basé sur l'approche LDI. Comme nous l'avons indiqué à la fin du chapitre précédent, l'approche précédente présentait certains défauts, notamment pour la compression des layers résiduels. Nous avons donc développé une nouvelle version de l'algorithme cette fois-ci basée sur la DWT (Discrete Wavelet Transform) tout en gardant le concept du LDI en amont afin de supprimer l'information redondante d'une vue à l'autre. Pour cela, nous décrirons dans un premier temps la nouvelle approche utilisée pour la compression des N layers chromatiques, puis nous présenterons les différents résultats obtenus avec la présente méthode.

III.3.1.2.1 Conversion colorimétrique

La première étape consiste en une conversion colorimétrique de l'espace RGB vers l'espace YCbCr à l'aide de la formule définie par l'équation II.1.16 (section II.1.2.1.1). Pour les mêmes raisons que pour l'algorithme basé DCT, aucun sous-échantillonnage n'est effectué. Une fois cette conversion effectuée, on peut passer à l'application de la DWT.

III.3.1.2.2 Application de la DWT

Comme cela est indiqué dans la section II.1.2.2, l'utilisation de la DWT à la place de la DCT permet d'atteindre des ratios de compression meilleurs pour une qualité d'image similaire à la décompression. De plus, la transformée en ondelettes travaille de manière globale sur l'image (contrairement à la DCT qui travaille de manière locale sur des blocs de taille 8×8) et ne va pas générer d'artefacts de type de "blocking" à fort taux de compression.

Afin d'appliquer la DWT nous avons besoin de choisir le type d'ondelettes à appliquer sur le signal d'entrée. Il existe une multitude d'ondelettes disponibles pour différentes applications telles que la compression d'images, le débruitage etc ... Pour notre algorithme nous avons choisi d'utiliser l'ondelette Daubechies 9/7 (utilisée pour la compression avec pertes de l'algorithme JPEG 2000 et dont les coefficients sont présentés dans la section II.1.2.2.1).

Le problème majeur de notre algorithme basé DCT (présenté dans la section III.3.1) résidait dans la compression des $N-1$ layers résiduels : en effet, ceux-ci contiennent un faible nombre de pixels utilisés et cette caractéristique doit être exploitée afin de tirer parti au maximum de l'approche LDI. La solution qui avait été adoptée dans notre précédente approche consistait à agréger horizontalement les pixels sur la gauche afin de minimiser le nombre de blocs non vides et ainsi réduire au maximum le nombre de coefficients à coder. Toutefois cette approche nous a vite paru non satisfaisante, les différents pixels résiduels n'étant pas localisés dans des zones voisines de l'image, ceux-ci ne présentaient pas de corrélation spatiale. Cette absence de corrélation spatiale génère donc des hautes fréquences après application de la DCT 3D. Lorsque le paramètre de quantification est trop élevé, celui-ci va supprimer ces hautes fréquences ce qui va se répercuter lors de la décompression par l'apparition d'artefacts dans les zones où les pixels résiduels sont localisés. Ce phénomène est ensuite amplifié lors de la projection de ceux-ci dans l'étape de reconstruction des N vues à partir des N layers résiduels.

Il fallait donc trouver un moyen de compresser ces pixels résiduels localisés dans des zones dispersées de chaque layer résiduel. Pour cela, nous avons choisi d'utiliser la SA-DWT (Shape-Adaptative Discrete Wavelet Transform) mise au point par Li et Li (2000). En effet, celle-ci répond parfaitement à notre besoin, à savoir : compresser certaines zones d'une image tout en omettant le reste de l'information. Afin de pouvoir omettre les zones ne contenant pas d'information, l'encodeur a besoin d'un masque binaire lui indiquant si ce pixel est à prendre en considération pour la compression ou pas. Dans notre cas, nous n'avons pas besoin de stocker ce masque pour la décompression, celui-ci peut être calculé à partir des N layers de disparité. Pour cela il suffit d'appliquer l'algorithme III.3.1

Algorithme III.3.1 Algorithme de génération des N masques pour la SA-DWT.

Entrées:
D_n : Les N layers de disparité
Sorties:
M_n : Les N masques pour la SA-DWT
Données:
N : Le nombre de layers
hauteur : La hauteur d'un layer
largeur : La largeur d'un layer
début
pour $i = 0$ jusqu'à N **faire**
  pour $j = 0$ jusqu'à *hauteur* **faire**
    pour $k = 0$ jusqu'à *largeur* **faire**
      si $D_i(j,k) \neq 255$ $M_i(j,k) \longleftarrow 255$
      sinon $M_i(j,k) \longleftarrow 0$
      fin si
    fin pour
  fin pour
fin pour
retourner M_n
fin

Une fois le masque calculé, on connaît les zones de l'image à prendre à considération et celles

à omettre. Toutefois, il reste un problème à régler : lors de l'application de la SA-DWT par convolution, les pixels résiduels étant très dispersés, il va nous manquer de l'information autour de ceux-ci afin d'appliquer la convolution. On se retrouve alors dans le même cas de figure que l'algorithme JPEG 2000 lorsque celui-ci compresse une tuile donnée par convolution et se trouve sur le bord de celle-ci. L'algorithme procédait alors par extension symétrique des coefficients afin de générer l'information qui lui manquait. Dans notre cas, nous procédons de la même manière : l'information manquante va donc être extrapolée par extension des coefficients présents. Une fois la SA-DWT appliquée, le codage des différents coefficients d'ondelettes va être réalisée à l'aide de l'algorithme SPIHT (décrit en section II.1.2.2.2) associé à un codeur arithmétique binaire.

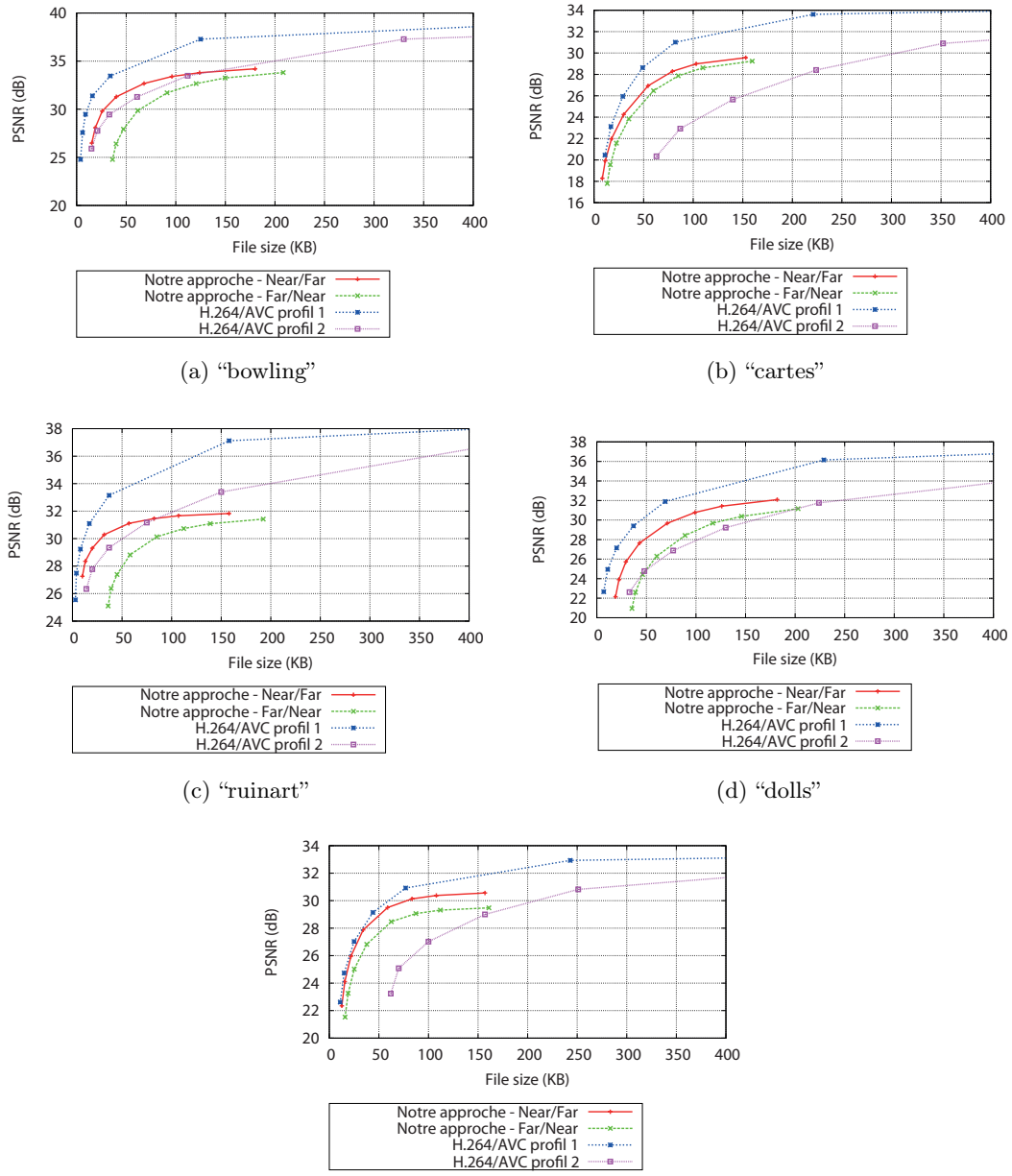
La section suivante présente une évaluation objective et subjective des résultats obtenus par cette seconde approche. Nous mettrons en avant les améliorations apportées par rapport à notre approche basée DCT et discuterons des différents problèmes soulevés par cette seconde approche.

III.3.1.2.3 Résultats obtenus

Cette section présente les résultats obtenus en termes de compression par notre approche basée sur la DWT. Afin de permettre une comparaison judicieuse vis-à-vis de notre précédent algorithme de codage, nous avons choisi de conserver le même protocole de test : nous gardons les mêmes jeux de données, nous conservons les deux profils associés au codage H.264/AVC utilisant les mêmes paramètres de codage et pour notre approche, nous faisons apparaître les résulats obtenus à l'aide de nos deux algorithmes d'estimation de disparités.

Sur la figure III.3.5, on peut voir les résultats en terme de PSNR pour chacun de nos 5 jeux d'images multi-vues. Comme dans la section précédente, la taille du fichier comprend l'information chromatique mais aussi l'information de disparité.

La première observation possible concernant notre approche basée DWT est que celle-ci est plus performante que l'approche basée DCT : pour toutes les séquences on constate un gain de compression mais aussi de qualité sur l'ensemble des jeux de données. On constate que pour la séquence "Cards", "ruinart" et "rose", les performances de notre seconde approche sont très proches du profil 1 d'encodage de H.264/AVC. De plus, on peut aussi remarquer que contrairement à l'approche basée DCT, notre approché basée DWT offre presque dans tous les cas (sauf pour la sequence "ruinart") de meilleurs résultats que le profil 2 d' H.264/AVC.



(a) "bowling"

(b) "cartes"

(c) "ruinart"

(d) "dolls"

(e) "rose"

FIGURE III.3.5 – Courbes taille de fichier/distortion pour les cinq datasets utilisés.

III.3.2 Compression des cartes de disparité

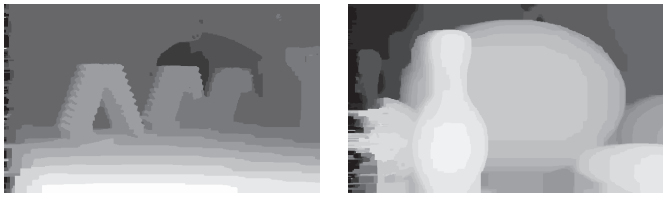
Dans cette section, nous décrivons la méthode utilisée pour compresser les N cartes de disparité générées durant la phase d'estimation des disparités (voir section III.2.1.2). De manière analogue à la texture chromatique, la texture de disparité est constituée d'un layer de référence et de $N-1$ layers résiduels. Le point important concernant la texture de disparité est que c'est elle qui va nous permettre de retrouver l'emplacement exact des pixels résiduels de la texture chromatique (qui ont été décalés vers la gauche afin d'optimiser la compression), mais aussi de les re-projeter dans les différentes vues. Or la moindre perturbation du signal associé aux cartes de disparité va forcément entraîner des erreurs de projection et donc des trous dans l'image finale reconstruite. Nous devons donc traiter cette texture à l'aide de méthodes de compression sans perte.

Pour cela, nous allons tout d'abord étudier les différentes cartes de disparité générées lors de nos tests sur des jeux multi-vues hétérogènes (image réelle et de synthèse) afin d'extraire les différentes caractéristiques de ces images. Puis, on comparera les résultats obtenus en utilisant divers algorithmes de compression sans perte.

III.3.2.1 Etude préliminaire des cartes de disparités

Les cartes de disparité sont générées durant la phase d'estimation des disparités en utilisant deux algorithmes différents (l'algorithme Near-Far et l'algorithme Far-Near) puis subissent ensuite la phase d'extraction des résidus durant laquelle sont constitués N layers (un de référence et $N-1$ résiduels). De manière analogue à la texture chromatique, les N layers sont constitués d'un layer de référence ainsi que de $N-1$ layers résiduels et la majorité de l'information sera donc stockée dans le layer de référence. La figure III.3.6 présente les layers de référence obtenus sur les différentes séquences multi-vues utilisées lors de nos tests.

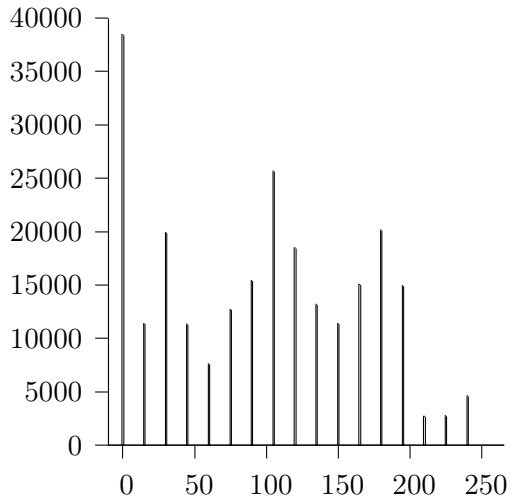
(a) Séquence "Art" (b) Séquence "Dolls" (c) Séquence "Cards"



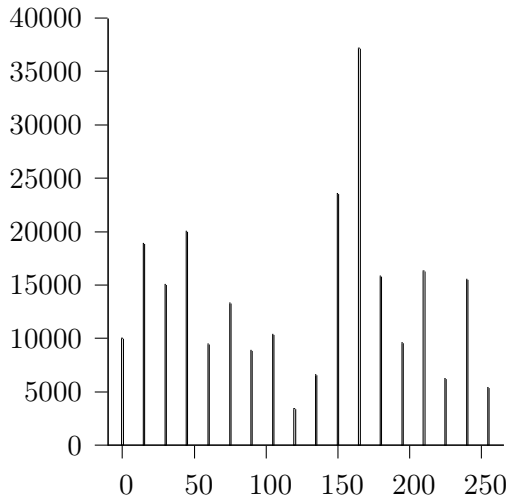
(d) Séquence "Caves" (e) Séquence "Bowling"

FIGURE III.3.6 – Différents layers de référence générés à partir de nos séquences multi-vues.

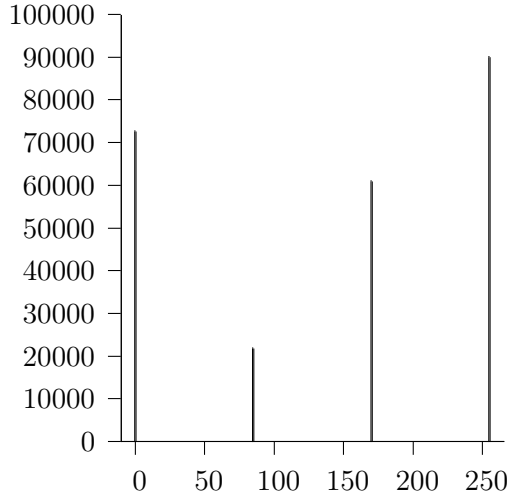
Sur la figure III.3.7, on peut trouver les différents histogrammes associés aux cartes de dispa-



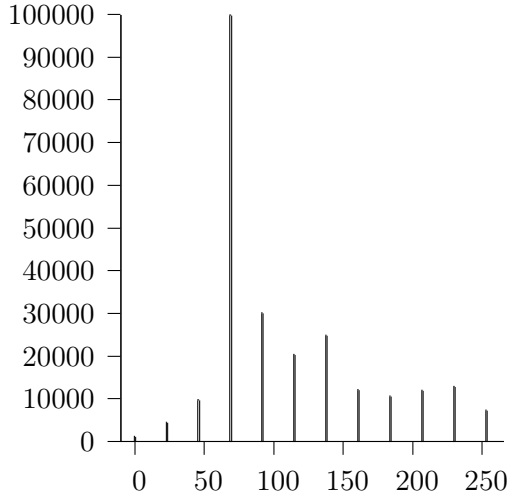
(a) Séquence “Art”



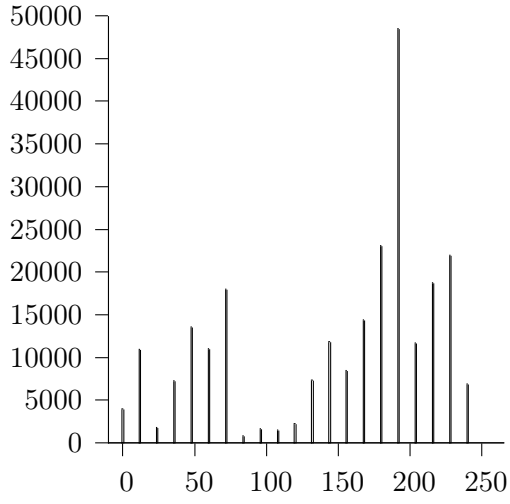
(b) Séquence “Dolls”



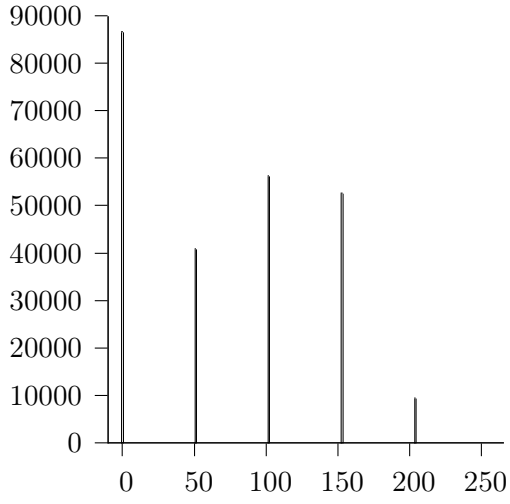
(c) Séquence “Cards”



(d) Séquence “Caves”



(e) Séquence “Bowling”



(f) Séquence “Rose”

FIGURE III.3.7 – Histogrammes associés aux layers de disparité de la figure III.3.6.

rités de la figure III.3.6. D'une manière générale, on peut s'apercevoir que seulement un nombre très limité de niveaux de gris (i.e. de valeurs de disparités) sont utilisés allant de 4 pour la séquence "Cards" à 21 pour la séquence "Bowling". Ce faible nombre est expliqué par nos valeurs de disparité entière, correspondant à une discrétisation de la profondeur. De plus d'après la figure III.3.6, on peut voir que les images de disparités contiennent de larges aplats d'intensité lumineuse.

L'algorithme devra répondre à une autre contrainte : en effet, aucune déformation du signal n'est autorisée afin de préserver les valeurs de disparité intactes mais en plus, nous n'avons même pas la possibilité de concaténer toutes les valeurs de disparité des layers résiduels (afin d'optimiser, dès le départ, le ratio de compression) car nous ne connaissons pas les localisations de ces pixels durant la décompression. A cause de cela, l'image devra être compressée telle quelle et l'algorithme devra pouvoir compresser de larges zones contenant la même valeur (correspondant à des zones vides) et ce, de manière efficace.

III.3.2.2 Tests effectués

Dans cette partie, nous présentons les résultats obtenus après avoir compressé nos cartes de disparités avec différents algorithmes de compression sans perte. Les tests ont été effectués en utilisant les algorithmes de compression sans perte généralement utilisés dans la littérature et appartenant à des familles distinctes d'algorithmes. Toutes ces méthodes sont présentées plus en détails dans les différentes sections indiquées entre parenthèses.

Pour la procédure de test nous avons donc décidé d'utiliser les algorithmes suivants :

Algorithmes basés dictionnaires et orientés image :
- GIF (Graphics Interchange Format), basé sur l'algorithme LZW (voir section II.1.1.2.1).
- PNG (Portable Network Graphics), basé sur l'algorithme LZ77 (section II.1.1.2.2).

Algorithmes basés dictionnaires et non orientés image :
- gzip, basé sur Deflate (LZ77 + tables de Huffman, voir section II.1.1.2 et II.1.1.1.2).
- LZMA2 (Lempel-Ziv-Markov chain-Algorithm 2), basé sur une variante de LZ77 (section II.1.1.2).

Algorithmes basés sur la modélisation de contextes :
- JPEG-LS, basé sur l'algorithme LOCO-I et utilisant une prédiction à 180° (section II.3.1).
- CALIC, utilisant une prédiction à 360° (section II.1.1.3.2).

La table III.3.4 présente les résultats obtenus après compression des textures de disparité. On peut tout d'abord s'apercevoir que l'algorithme JPEG-2000 dans sa version sans perte n'arrive pas à atteindre de bons résultats. Les approches par transformation nécessitent généralement une étape de quantification pour se révéler vraiment efficaces. De plus, même si JPEG-2000 opère de manière globale via l'utilisation de la DWT, le nombre d'informations à coder durant le processus de codage entropique est égal au nombre de pixels.

Séquence	GIF	PNG	JPEG-2000	JPEG-LS	CALIC	gzip	LZMA2
Art	0.45	0.28	1.39	0.71	0.35	0.26	0.22
Dolls	0.29	0.19	0.97	0.44	0.18	0.22	0.16
Cards	0.14	0.08	0.46	0.26	0.08	0.11	0.06
Caves	0.14	0.10	0.51	0.23	0.08	0.11	0.07
Bowling	0.28	0.17	0.77	0.42	0.17	0.17	0.13
Rose	0.18	0.13	0.74	0.30	0.10	0.16	0.10

TABLEAU III.3.4 – Résultats obtenus (bits/pixel) en utilisant différents algorithmes de compression sans perte sur nos cartes de disparité

Conclusion

Dans ce dernier chapitre, nous avons décrit les différents formats et techniques de codage spécifiques à la vision multi-scopique ainsi que nos contributions associées à celle-ci.

Dans la première partie avons dressé un état de l'art des différents formats 3D et techniques de codage associés aux séquences multi-vues (stéréoscopiques et auto-stéréoscopiques). En effet, l'utilisation de formats 3D spécifiques (tels que 2D+D, LDI par exemple) permet de réduire considérablement le volume de données à traiter par le système de codage en aval et ainsi de réduire la corrélation existante entre le nombre de vues en entrée et la taille du flux compressé.

Ensuite nous avons présenté notre approche LDI développée dans le cadre du projet CamRelief et spécifique à notre géométrie de capture. Cette approche repose sur une méthode d'estimation des disparités innovante, générant des cartes de disparités à valeurs entières et permettant d'éviter des imprécisions liées aux valeurs flottantes lors de la reconstruction des différentes vues du LDI ou de la synthèse de points de vue intermédiaires (via des méthodes DIBR). Notre approche, via ses concepts de base, permet ainsi la génération implicite du LDI sans traitements supplémentaire.

Nos contributions, basées sur l'approche précédemment décrite, ont été exposées dans la troisième partie de ce chapitre. Dans un premier temps nous avons proposé un algorithme de compression multi-vues utilisant la DCT-3D pour la compression des layers résiduels de la LDI. Cette première approche ne nous paraissant pas satisfaisante, nous avons ensuite présenté notre deuxième approche, destinée à améliorer la première et basée sur l'utilisation de la SA-DWT et de l'algorithme SPIHT. Nous avons pu constater que celle-ci remplit son objectif, à savoir produire de meilleurs résultats que notre première approche. Toutefois cette seconde approche ne permet pas d'obtenir des ratios de compression comparables au standard H.264/AVC sur toutes les séquences de test et nécessitera donc certaines améliorations futures. La dernière partie de ce chapitre a permis de présenter les différents schémas de codage possibles pour la compression de nos cartes de disparités (qui doivent être compressées sans perte). De cette étude, nous avons pu nous rendre compte que les algorithmes de compression de données basés sur la notion de dictionnaire (comme LZMA2) permettent d'obtenir les meilleurs taux de compression.

Conclusion

Les travaux de cette thèse traitent de la compression multi-vues des flux auto-stéréoscopiques. Ces flux, produits de l'acquisition simultanée d'une même scène par un dispositif de capture spécifique, représentent un volume de données conséquent qui doit être compressé afin de faciliter son stockage ou sa transmission (dans le cadre de la visio-conférence par exemple). Ce besoin de compression est d'autant plus fort que les progrès technologiques en termes d'écran permettent l'affichage d'images utilisant des résolutions de plus en plus élevées et ce, à des fréquences qui augmentent constamment elles aussi. Cependant, on constate généralement que les flux auto-stéréoscopiques présentent une forte corrélation qui doit être prise en compte par le système de compression afin d'atténuer la relation existante entre le nombre de vues captées et le volume de données compressé en sortie. Nous proposons dans cette thèse diverses solutions destinées à la compression des flux auto-stéréoscopiques, chacune répondant à un besoin spécifique.

Pour cela, nous avons tout d'abord étudié le contexte spécifique à la vision multiscopique à travers ses aspects physiologiques (qui permettent à l'être humain de percevoir le relief) et historiques. Nous nous sommes aussi penchés sur les multiples techniques de restitution stéréoscopiques et auto-stéréoscopiques qui exploitent les principes de la "stéréopsie" et qui permettent à un observateur de percevoir le relief d'une scène donnée. Enfin, nous avons présenté nos différents dispositifs d'acquisition auto-stéréoscopiques permettant de générer du contenu relief à partir d'une prise de vue réelle. Ces dispositifs, qui ont tous été réalisés dans le cadre du projet Cam-Relief en collaboration avec la société 3DTVSolutions, présentent certaines spécificités (telle que la géométrie de capture ou le type de capteur optique utilisé) qui doivent être prises en compte lors de la conception de nos algorithmes de compression.

Nous avons ensuite exploré l'adaptation possible de méthodes de compression monoscopiques au cas multi-vues. On constate, dans la majorité des cas, que les algorithmes de compression multi-vues reposent sur des concepts issus des techniques de compression associés à la vision 2D. Il nous semblait donc essentiel, afin de permettre une meilleure compréhension pour le lecteur, de décrire ces méthodes de compression (qu'elles soient avec pertes ou sans perte) à travers un état de l'art associé à celles-ci. Nous avons ensuite présenté notre première contribution : MICA (Multiview Image Compression Algorithm). MICA est un algorithme à faible complexité permettant un encodage temps réel de séquences multi-vues. En effet, un des différents besoins évoqués dans le cadre du projet Cam-Relief est le développement d'un schéma de compression temps réel dédié à notre périphérique de capture afin de rendre possible des applications de type visio-conférence. Notre algorithme MICA est destiné à cet usage. Il se base sur une approche alternative permettant d'exploiter la redondance existante entre les différentes vues tout en gardant une faible complexité via les concepts d'images de différence et d'images de référence. Les images de différences, générées à partir de deux vues adjacentes d'une image multi-vues, mettent en évidence les zones de l'image soumises à l'effet parallaxe (pixels ayant une forte intensité lumineuse) et celles appartenant au plan de disparité zéro (pixels ayant une faible intensité lumineuse) qui constituent approximativement 90% de l'information du fait de notre géométrie de capture. Nous avons ensuite développé un système de codage adaptatif qui permet de coder de manière différente les pixels à faible intensité lumineuse et ceux à forte intensité lumineuse. Ces derniers qui correspondent aux zones soumises à l'effet parallaxe et qui sont cruciaux lors de la restitution relief, sont compressés de manière presque sans perte. Pour les autres, on utilise des codes à longueur variable modélisés en accord avec la distribution de leurs intensités lumineuses quantifiées. Les images de référence (au nombre de deux dans le cas 8 vues), qui nous permettent de limiter la propagation d'erreurs lors de la reconstruction des différentes vues pendant le processus de décompression, sont compressées sans pertes. L'algorithme a ensuite

été adapté au temps afin de permettre la compression de séquences multi-vues. La corrélation temporelle est exploitée au niveau des vues de référence, grâce encore une fois, au principe des images de différence. Lors de la comparaison des résultats face au format JPEG (à taille de fichier égale), on se rend compte que même si MICA ne permet pas d'obtenir un gain en terme de qualité objective, ce dernier permet une meilleure préservation des zones soumises à l'effet parallaxe vis-à-vis de la distorsion du signal. Ces travaux ont fait l'objet d'une publication dans une conférence internationale avec comité de lecture (voir la partie "Publications").

Nous avons ensuite présenté notre deuxième contribution, Multiview-LS, qui est un algorithme de compression multi-vues sans perte. Bien que la compression sans perte puisse sembler anecdotique dans le cas multi-vues, il nous paraissait tout de même intéressant de proposer une telle solution. En effet, lors de la production de séquences multi-vues, une étape de "post-processing" est souvent nécessaire afin d'opérer des corrections sur les images ou d'ajouter différents effets. Cette étape peut être perturbée par la présence d'artefacts et devenir fastidieuse pour l'utilisateur. L'utilisation d'un algorithme de compression sans perte nous semble donc une manière judicieuse d'éviter tous ces désagréments. On constate en outre qu'il existe très peu (ou pas) de travaux dédiés à la compression multi-vues sans perte. Multiview-LS, basé sur l'algorithme JPEG-LS, utilise les concepts associés à la modélisation de contexte. Les méthodes utilisant la modélisation de contexte sont hautement adaptatives et permettent généralement d'obtenir les meilleurs résultats en terme de ratio de compression. JPEG-LS utilise le voisinage spatial d'un pixel (appelé "template") pour la modélisation de contexte. Notre algorithme Multiview-LS propose d'exploiter la corrélation inter-vues et temporelle en considérant un template, non plus constitué de pixels appartenant au voisinage du pixel courant, mais de pixels associés à la même localisation dans les frames voisines de la grille 2D d'images. Une étude préliminaire a été menée afin de vérifier que l'utilisation d'un tel template est pertinente vis à vis des étapes de modélisation de contexte et de codage en aval. Lors des phases de tests, on a pu constater que Multiview-LS permettait d'obtenir de meilleurs ratios de compression que les méthodes de compression sans perte généralement citées dans l'état de l'art.

Nous nous sommes ensuite intéressés aux méthodes de compression spécifiques aux flux auto-stéréoscopiques. Pour cela, nous avons tout d'abord dressé un état de l'art exposant les différents formats 3D et techniques de codages associées dans le cadre de la vision stéréoscopique, puis de la vision auto-stéréoscopique. Ces formats 3D permettent une première exploitation de la redondance inter-vues et de réduire ainsi le volume de données transmis au processus de codage en aval. Nous avons ensuite présenté notre approche LDI, développée dans le cadre du projet Cam-Relief et exploitant les différentes spécificités de nos dispositifs d'acquisition. Notre LDI est une représentation constituée de N couches (ou "layers") associées à chacune des N vues. La première couche contient tous les pixels de la première vue, alors que les $N-1$ couches suivantes (désignées par le terme "couches résiduelles") contiennent uniquement l'information n'appartenant pas à la première couche (à cause des zones d'occultations). Notre approche LDI comprend une étape d'estimation des disparités permettant la génération des cartes de disparités et une étape de génération de la LDI. L'estimation des disparités est réalisée à l'aide de deux algorithmes (Near/Far et Far/Near) reposant sur des concepts innovants. Le premier permet une bonne détection des occultations et produit des cartes de disparités précises, alors que le deuxième, qui opère de manière locale, génère des cartes de disparités moins précises mais en temps réel. Grâce aux concepts novateurs associés à l'étape d'estimation des disparités, la génération du LDI est automatique.

Enfin, nous avons présenté nos deux dernières contributions destinées à la compression de ce LDI. La première approche est basée sur l'utilisation conjointe de la DCT 2D pour la compression de la couche de base et de la DCT 3D pour les couches résiduelles. La compression de la couche de base utilise, elle, un pipeline de traitements standard (conversion colorimétrique/DCT 2D/quantification/parcours/codage entropique), alors que pour la compression des couches résiduelles, nous avons été amenés à développer notre propre méthode. Les pixels contenus dans chacun de nos $N-1$ couches résiduelles sont tout d'abord concaténés vers le bord gauche de chaque couche, puis celles-ci sont rassemblées dans un volume 3D sur lequel on applique la DCT 3D. Les méthodes usuelles utilisées lors des étapes de quantification, parcours et de codage entropique sont conçues pour le cas 2D. Il a donc fallu adapter ces différentes méthodes afin de permettre leur application au cas 3D. Nous avons ensuite présenté les résultats obtenus par cette première approche et les avons confronté au standard H.264/AVC en utilisant deux profils d'encodage (un profil utilisant certaines des fonctionnalités avancées d'H.264/AVC, et un autre utilisant uniquement le codage intra-frame). On a pu constater, à travers l'exploitation des résultats, que notre algorithme proposait de meilleurs résultats que le profil "intra" de H.264/AVC sur certaines séquences mais que dans la majorité des cas, celui-ci produisait des résultats relativement équivalents. C'est pourquoi nous avons présenté dans la partie suivante notre deuxième approche qui vise à améliorer le schéma de compression.

Notre deuxième approche est, elle, basée sur l'utilisation de la SA-DWT associée à l'algorithme SPIHT. Une des observations que nous avons pu faire concernant l'approche basée DCT 3D est que l'étape de concaténation opérée sur les pixels appartenant aux $N-1$ couches résiduelles générait des hautes fréquences après application de la DCT 3D rendant le processus de codage entropique moins performant. L'utilisation de la SA-DWT couplée à l'algorithme SPIHT permet de pallier à ce problème. En effet, grâce à un masque binaire permettant de dissocier l'information pertinente (pixels résiduels) de l'information non pertinente (pixels vides), on ne code que l'information nécessaire sans introduire de hautes fréquences dans le signal. Nous avons pu ainsi constater que cette seconde approche permettait d'obtenir de meilleurs résultats que celle basée sur la DCT 3D. Celle-ci permet notamment d'avoisiner, à fort taux de compression, les performances en terme de débit/distorsion du profil utilisant les fonctionnalités avancées de H.264/AVC sur les séquences multi-vues respectant notre géométrie de capture. Dans la dernière partie de ce document, nous avons mené une étude visant à déterminer l'algorithme de compression à utiliser pour l'information de disparité contenue dans le LDI. Celle-ci, utilisée lors de la reconstruction des N vues à partir du LDI décompressé, doit être compressée sans perte. Nous avons donc comparé les différents résultats obtenus en utilisant différents algorithmes généralement cités dans l'état de l'art. Il semblerait aux vues de ces résultats que la meilleure solution consiste à utiliser l'algorithme LZMA2. En effet, l'information de disparité contenue dans le LDI utilise un nombre restreint de niveaux de gris et présente de larges aplats d'intensité lumineuse (et donc un nombre important de répétitions). Il semble donc normal que ce type d'algorithme (basé sur la notion de dictionnaire) permette d'obtenir de meilleurs résultats.

Au final, même s'il apparaît qu'un travail d'amélioration de nos différentes contributions reste à faire, nous pensons que les différentes structures visant à exploiter la corrélation inter-vues sont pertinentes. Concernant notre algorithme MICA, nous devons maintenant trouver un moyen d'améliorer les taux de compression. Cela pourrait être rendu possible notamment grâce à la mise en place d'un système de quantification plus élaboré. Notre système de codage actuel (utilisant des codes à longueur variable) pourra être remplacé par un codeur de type Rice-Golomb ou par un codeur arithmétique binaire afin d'optimiser le processus de codage entropique. Notre

approche basée LDI, quant à elle, devra elle aussi être améliorée, peut être en adaptant la méthode décrite par Jantet *et al.* (2010) à notre structure de LDI ?

Annexes

Annexe A

Jeux de données utilisés

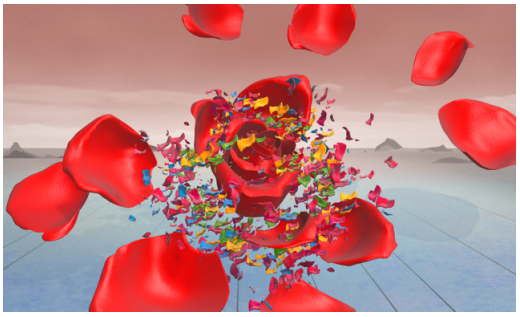
Dans cette annexe, le lecteur pourra trouver la première vue des différents jeux de données utilisés dans ce document.

Les séquences (a) à (f) ont été réalisées par la société 3DTVSolutions et respectent la géométrie de capture propre à l'Octocam. Les séquences (g) et (h) sont disponibles sur le site "Middlebury Stereo Vision Page"[1].

(a) Séquence "Cartes"

(b) Séquence "Pinceau"

(c) Séquence "Poupees"



(d) Séquence "Rose"

1. http ://vision.middlebury.edu/stereo/

(e) Séquence “Statue”

(f) Séquence “Caves”

(g) Séquence “Bowling”

(h) Séquence “Dolls”

Annexe B

Glossaire

- APS (Auxiliary Picture Syntax) : Mécanisme du standard H.264 permettant l'ajout d'une composante auxiliaire au flux vidéo 2D.
- AVC (Advanced Video Coding) : Partie 10 du standard MPEG-4 permettant l'utilisation de fonctionnalités avancées de codage.
- CABAC (Context Adaptative Binary Arithmetic Coding) : Méthode de codage entropique basée sur un codage arithmétique binaire et utilisée par le standard H.264.
- CALIC (Context Adaptative Lossless Image Compression) : Algorithme de compression d'images sans perte basé sur la modélisation de contexte.
- CAVLC (Context Adaptative Variable Length Codes) : Méthode de codage entropique basée sur des codes binaires à longueur variable et utilisée par le standard H.264.
- CSV (Conventional Stereo Video) : Format 3D associé à la stéréoscopie et contenant les deux vues du couple stéréo.
- DCT (Discrete Cosine Transform) : Transformée en cosinus discrète.
- DES (Depth Enhanced Stereo) : Format 3D associé à l'auto-stéréoscopie et basé sur l'utilisation conjointe de 2 LDI par image multi-vues.
- DIBR (Depth Image Based Rendering) : Technique de rendu image basée sur l'utilisation des profondeurs.
- DWT (Discrete Wavelet Transform) : Transformée en ondelettes discrète.
- EBCOT (Embedded Block Coding with Optimal Truncation points) : Méthode de codage entropique utilisée par le format JPEG2000.
- EZW (Embedded Zerotree Wavelet) : Méthode destinée au codage des coefficients d'ondelettes et utilisant une structure d'arbre.
- FELICS (Fast Efficient and Lossless Image Compression System) : Algorithme de compression d'images sans perte basé sur la modélisation de contexte.
- FPA (Frame Packing Arrangement) : Message propre à la syntaxe du standard H.264 permettant de spécifier le type d'arrangement des deux vues d'un couple stéréoscopique au sein d'un flux vidéo unique.
- FSS (Frame Sequential Stereo) : Format 3D correspondant à l'entrelacement temporel des deux vues constituant le couple stéréoscopique.
- GIF (Graphics Interchange Format) : Format d'image utilisant une compression sans perte

maintenant remplacé par le format PNG.
- GOP (Group Of Pictures) : Groupe d'images pouvant être codé/décodé de manière indépendante, défini par le standard de codage vidéo MPEG.
- HD (High Definition) : Norme d'affichage associée à une résolution de 1920×1080 pixels.
- ISO (International Organization for Standardization) : Organisme de normalisation international composé de représentants d'organisations nationales de normalisation de 157 pays.
- JMVM (Joint Multiview Video Model) : Implémentation du standard H.264/MVC conduite par le groupe JVT.
- JPEG (Joint Photographic Expert Group) : Comité d'experts qui édite des normes de compression pour l'image fixe. Correspond aussi à un standard de compression d'images avec pertes basé sur la DCT.
- JVT (Joint Video Team) : Partenariat entre les experts appartenant aux groupes VCEG et MPEG et étant responsable du développement des standards H.26x.
- LDI (Layered Depth Image) : Format 3D associé à l'auto-stéréoscopie (N vues) et basé sur l'utilisation des N vues et de leur carte de profondeur/disparité.
- LDV (Layered Depth Video) : Format 3D associé à l'auto-stéréoscopie et correspondant à une succession temporelle de LDI.
- LUT (Look-Up Table) : Table de correspondance.
- MED (Median Edge Detector) : Fonction de prédiction fixe utilisée par l'algorithme JPEG-LS.
- MPEG (Motion Photographic Expert Group) : Groupe d'experts chargé du développement de normes internationales pour la compression, la décompression, le traitement et le codage de la vidéo, de l'audio et de leur combinaison.
- MRS (Mixed Resolution Stereo) : Format 3D associé à la stéréoscopie et sous-échantillonnant la résolution d'une des vues du couple stéréo.
- MVD (MultiView plus Depth) : Format 3D associé à l'auto-stéréoscopie et contenant N flux vidéos d'information chromatique et N flux vidéos d'information de profondeur/disparité.
- MVP (MultiView Profile) : Profil de codage du standard MPEG-2 destiné à la compression de flux stéréoscopiques.
- MVC (MultiView Coding) : Extension du standard H.264 permettant la compression de séquences stéréoscopiques ou auto-stéréoscopiques.
- MVV (MultiView Video) : Format 3D associé à l'auto-stéréoscopie et contenant N flux vidéos d'information chromatique.
- PNG (Portable Network Graphics) : Format d'image utilisant une compression sans perte et successeur du format GIF.
- PSNR (Peek Signal to Noise Ratio) : Mesure liée au rapport signal sur bruit entre deux images (en décibels).
- QFHD (Quad Full High Definition) : Norme d'affichage associée à une résolution de 3840×2160 pixels.
- RCT (Reversible Color Transform) : Transformation colorimétrique réversible.
- RLE (Run Length Encoding) : Algorithme de compression de données sans perte basé sur la répétition de symboles.
- SA-DWT (Shape Adaptative Discrete Wavelet Transform) : Transformée en ondelettes discrète effectuée sur des zones particulières d'une image repérées par un masque binaire.
- SAD (Sum of the Absolute Differences) : Formule mathématique permettant de mettre en évidence la similarité ou non entre deux données du même type.
- SEI (Supplemental Enhanced Information) : Messages propres à la syntaxe du standard

H.264 permettant d'apporter des informations spécifiques au codeur ou au décodeur.
- SPIHT (Set Partitionning In Hierarchical Trees) : Méthode destinée au codage des coefficients d'ondelettes et utilisant une structure d'arbre.
- UHD (Ultra High Definition) : Norme d'affichage associée à une résolution de 7680×4320 pixels.
- VCEG (Video Coding Experts Group) : Groupe d'experts responsable du processus de standardisation des standards H.26x

Bibliographie

Information technology - digital compression and coding of continuous-tone still images - requirements and guidelines, a. [p. 49, 53 et 145]

Information technology - jpeg 2000 image coding system - part 1 : Core coding system, b. [p. 54 et 100]

Information technology - lossless and near-lossless compression of continuous-tone still images, c. [p. 83, 91, 92 et 100]

Dolby open specification for frame-compatible 3d systems. Rap. tech., Dolby Laboratories, 2010. [p. 107]

N. Abramson : *Information Theory and Coding*. Mc Grawn - Hill, 1963. [p. 35]

N. Ahmed, T. Natarajan et R. K. Rao : Discrete cosine transform. *IEEE Transactions on Computers*, 23:90 – 93, 1974. [p. 47]

M. Antonini, M. Barlaud, P. Mathieu et I. Daubechies : Image coding using wavelet transform. *IEEE Transactions on Image Processing*, 1:205 – 220, 1992. [p. 56]

B. Battin, P. Vautrot, D. Debons et L. Lucas : A new near-lossless scheme for multi-view image compression. *In IS&T/SPIE Electronic Imaging, Conference EI101 : Stereoscopic Displays and Application XXI (SDA XXI), SPIE*, 2010. [p. 79]

B. BE et E. K. Company : Color imaging array. Rap. tech., 1975. [p. 26 et 82]

A. Berthier : Images stéréoscopiques de grand format. *Le Cosmos*, 34:205–210, Mai 1896. [p. 20]

S. Blackstock : Lzw and gif explained, 1987. URL http://www.martinreddy.net/gfx/2d/GIF-comp.txt. [p. 40 et 100]

D. Broberg : Infrastructures for home delivery, interfacing, captioning, and viewing of 3d content. *Proceedings of the IEEE - Special Issue : 3D Media and Displays*, 99:684 – 693, 2011. [p. 107]

H. Brust, A. Smolic, K. Mueller, G. Tech et T. Wiegand : Mixed resolution coding of stereoscopic video for mobile devices. *In 3DTV-Conference*, 2009. [p. 107]

R. C. Calderbank, I. Daubechies, W. Sweldens et B.-L. Yeo : Wavelet transforms that map integers to integers. *Applied and Computational Harmonic Analysis*, 5:332 – 369, 1998. [p. 57]

R. Chan et M. Lee : 3d-dct quantization as a compression technique for video sequences. *In International Conference on Virtual Systems and Multimedia*, p. 188 – 196, 1997. [p. 131 et 146]

W.-H. Chen, C. Smith et S. Fralick : A fast computationnal algorithm for the discrete cosine transform. *IEEE Transactions on Communications*, 25(9):1004 – 1009, Septembre 1977. [p. 144]

R. Crochiere, S. Webber et J. Flanagan : Digital coding of speech in sub-bands. *In IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1976. [p. 53]

A. Croiser, D. Esteban et C. Galand : Perfect channel splitting by use of interpolation/decimation/tree decomposition techniques. *In International Symposium on Information, Circuits, Systems*, Patras, Grece, 1976. [p. 53]

I. Daubechies : *Ten lectures on wavelets*. Society for Industrial and Applied Mathematics, 1992. [p. 56]

N. A. Dodgson : Autostereoscopic 3d displays. *Computer*, 38(8):31–36, août 2005. ISSN 0018-9162. URL http://dx.doi.org/10.1109/MC.2005.252. [p. 20]

N. Faller : An adaptative system for data compression. *In Record of the 7th Asilomar Conference on Circuits, Systems, and Computers*, p. 593 – 597, 1973. [p. 34]

R. Fano : *Transmission of Information*. MIT Press, 1961. [p. 32]

R. Gallager et D. V. Voorhis : Optimal source codes for geometrically distributed integer alphabets. *IEEE Transactions on Information Theory*, 21:228 – 230, 1975. [p. 95 et 96]

R. G. Gallager : Variations on a theme by huffman. *IEEE Transactions on Information Theory*, 24:668 – 674, 1978. [p. 34]

J.-U. Garbas, U. Fecker et A. Kaup : Wavelet-based multi-view video coding with full scalability and illumination process. *Proceedings of the 15th international conference on Multimedia*, 2007. [p. 132]

E. Gershikov, E. Lavi-Burlak et M. Porat : Correlation-based approach to color image compression. *Elsevier Signal Processing : Image communication*, 22:719 – 733, 2007. [p. 46]

S. Golomb : Run-length encodings. *IEEE Transactions on Information Theory*, 12:399 – 401, 1966. [p. 43]

X. Guo, Y. Lu, F. Wu, W. Gao et S. Li : Distributed multi-view video coding. *In Visual Communications and Image Processing*, 2006. [p. 132]

M. Halle : Autostereoscopic displays and computer graphics. *In ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, New York, NY, USA, 2005. ACM. URL http://doi.acm.org/10.1145/1198555.1198736. [p. 20]

M. Hirsch et D. Lanman : Build your own 3d display. *In ACM SIGGRAPH 2010 Courses*, SIGGRAPH '10, p. 4 :1–4 :106, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0395-8. URL http://doi.acm.org/10.1145/1837101.1837105. [p. 20]

Holografika : Holografika white paper. http://www.holografika.com/Technology/Technology-Principles.html. [p. 22]

P. G. Howard et J. S. Vitter : Fast and efficient lossless image compression. *In* J. A. Storer, éd. : *Proceedings of the 1993 Data Compression Conference*, p. 351–360, Los Alamitos, CA, 1993. IEEE Computer Society Press. [p. 41]

D. Huffman : A method for the construction of minimum redundancy codes. *In Proceedings of the IRE*, vol. 40, p. 1098 – 1101, 1952. [p. 32]

F. E. Ives : A novel stereogram. *Journal of the Franklin Institute*, 153(1):51–52, January 1902. [p. 20]

V. Jantet, L. Morin et C. Guillemot : Incremental-ldi for multi-view coding. *In 3DTV Conference : The True Vision - Capture, Transmission and Display of 3D Video*, 2009. [p. 79, 122 et 123]

V. Jantet, L. Morin et C. Guillemot : Génération, compression et rendu de ldi. *In COmpression et REpresentation des signaux AUdiovisuels (CORESA)*, 2010. [p. 127, 128 et 166]

P. Katz : Rfc 1951 : Deflate compressed data format specifications version 1.3, 1996. [p. 39 et 40]

Y. Kim, K. Hong et B. Lee : Recent researches based on integral imaging display method. *3D Research*, 1:17–27, 2010. ISSN 2092-6731. URL http://dx.doi.org/10.1007/3DRes.01(2010)2. 10.1007/3DRes.01(2010)2. [p. 22]

D. E. Knuth : Dynamic huffman coding. *Journal of Algorithms*, 6:163 – 180, 1985. [p. 34]

D. Lanman, M. Hirsch, Y. Kim et R. Raskar : Content-adaptive parallax barriers : optimizing dual-layer 3d displays using low-rank light field factorization. *In ACM SIGGRAPH Asia 2010 papers*, SIGGRAPH ASIA '10, p. 163 :1–163 :10, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0439-9. URL http://doi.acm.org/10.1145/1866158.1866164. [p. 22]

S. Li et W. Li : Shape-adaptative discrete wavelet transforms for arbitrarily shaped visual object coding. *IEEE Transactions on Circuits and Systems for Video Coding*, 10:725 – 743, 2000. [p. 151]

D. Marpe, G. Blattermann et T. Wiegand : Adaptive codes for h.26l. Rap. tech., JVT. [p. 75]

D. Marpe, H. Schwarz et T. Wiegand : Context-based adaptative binary arithmetic coding in the h.264/avc video compression standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13:620 – 636, 2003. [p. 75]

W. Matusik et H. Pfister : 3d tv : a scalable system for real-time acquisition, transmission, and autostereoscopic display of dynamic scenes. *In ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, p. 814–824, New York, NY, USA, 2004. ACM. URL http://doi.acm.org/10.1145/1186562.1015805. [p. 20]

P. Merkle, K. Mueller, A. Smolic et T. Wiegand : Efficient compression of multi-view video exploiting inter-view dependencies based on h.264/mpeg-4 avc. *In IEEE International Conference on Multimedia and Expo*, 2006. [p. 79]

P. Merkle, A. Smolic, K. Mueller et T. Wiegand : Efficient prediction structures for multi-view video coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 17:1461 – 1473, 2007. [p. 79, 125 et 127]

P. Merkle, Y. Wang, K. Muller, A. Smolic et T. Wiegand : Video plus depth compression for mobile 3d services. *In 3DTV Conference : The True Vision - Capture, Transmission and Display of 3D Video*, 2009. [p. 115]

B. Michel : *La stéréoscopie numérique*. Eyrolles, 2012. [p. 14]

A. Moffat, R. M. Neal et I. H. Witten : Arithmetic coding revisited. *ACM Transactions on Information Systems*, 16(3):256 – 294, Juillet 1998. [p. 35]

A. Netravali et J. O. Limb : Picture coding : A review. *In Proceedings of the IEEE*, 1980. [p. 93]

M. E. Newell, R. G. Newell et T. L. Sancha : A new approach on the shaded picture problem. *Proceedings of the ACM National Conference*, p. 443 – 450, 1972. [p. 141]

C. Niquin, S. Prévost et Y. Remion : Accurate multi-view depth reconstruction with occlusions handling. *In 3DTV-Conference 2009 : The True Vision - Capture, Transmission and Display of 3D Video (3DTV-CON 2009)*, 2009. [p. 133]

C. Niquin, S. Prévost et Y. Remion : Depth extraction for auto-stereoscopic sets by means of a mesh reconstruction algorithm. *In IS&T/SPIE Electronic Imaging, Conference EI101 : Stereoscopic Displays and Application XXI (SDA XXI)*, 2010a. [p. 133]

C. Niquin, S. Prévost et Y. Remion : An occlusion approach with consistency constraint for multiscopic depth extraction. *International Journal of Digital Multimedia Broadcasting (IJDMB), special issue Advances in 3DTV : Theory and Practice*, p. 1 – 8, 2010b. [p. 133 et 136]

C. Niquin : *Reconstruction du relief et mixage réel virtuel par caméras relief multi-points de vues*. Thèse de doctorat, Université de Reims Champagne-Ardenne, 2011. [p. 133, 134, 136 et 139]

R. Ohnishi, Y. Ueno et F. Ono : The efficient coding scheme for binary sources. *IECE of Japan*, 60-A:1114 – 1121, 1977. [p. 92]

M. Ouaret, F. Dufaux et T. Ebrahimi : Fusion-based multiview distributed video coding. *In 4th ACM international workshop on video surveillance and sensor networks*, 2006. [p. 132]

J. Prévoteau, S. Chalençon-Piotin, D. Debons, L. Lucas et Y. Remion : Multi-view shooting geometry for multiscopic redering with controlled distortion. *International of Digital Multimedia Broadcasting (IJDMB), special issue Advances in 3DTV : Theory and Practice*, p. 1 – 11, 2010. [p. 23]

R. Rice : Some practical universal noiseless coding techniques. *JPL Publication 79*, 22, 1979. [p. 43]

I. E. Richardson : *The H.264 Advanced Video Compression Standard, Second Edition*. Wiley, 2010. [p. 69 et 75]

A. Said et W. A. Pearlman : A new fast and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(6):243 – 250, Juin 1996. [p. 60 et 63]

M. Santhi et R. S. D. W. Banu : Inter color correlation based enhanced color spiht coder. *European Journal of Scientific Research*, 57:592 – 600, 2011. [p. 46]

J. Shade, S. Gortler, L. He, et R. Szeliski : Layered depth images. *Proc. ACM SIGGRAPH*, p. 231 – 242, 1998. [p. 120]

J. M. Shapiro : Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing*, 41:3445 – 3462, 1993. [p. 60]

Y. Shishikui, Y. Fujita et K. Kubota : Super hi-vision : the star of the show ! Rap. tech., EBU Technical Review, 2009. [p. 3]

A. Smolic, K. Mueller, K. Dix, P. Merkle, P. Kauff et T. Wiegand : Intermediate view interpolation based on multiview video plus depth for advanced 3d video systems. *In IEEE International Conference on Image Processing*, 2008. [p. 107 et 119]

A. Smolic, K. Mueller, P. Merkle, P. Kauff et T. Wiegand : An overview of available and emerging 3d video formats and depth enhanced stereo as efficient generic solution. *In Picture Coding Symposium*, 2009. [p. ix, 123 et 124]

L. Stelmach, W. J. Tam, D. Meegan et A. Vincent : Stereo image quality : Effects of mixed spatio-temporal resolution. *IEEE Transactions on Circuits and Systems for Video Technology*, 10, 2000. [p. 108]

E. J. Stollnitz, T. D. DeRose et D. H. Salesin : *Wavelets for Computer Graphics : Theory and Applications*. Morgan Kauffmann, 1996. [p. 56]

M. Sugawara : Super hi-vision : research on a future ultra-hdtv system. Rap. tech., EBU Technical Review, 2008. [p. 3]

W. Sweldens et P. Schröder : Building your own wavelets at home. SIGGRAPH 96 Course Notes, 1996. [p. 56]

D. Taubman : High performance scalable image compression with ebcot. *IEEE Transactions on Image Processing*, 9:1158 – 1170, 2000. [p. 59 et 60]

D. Taubman, E. Ordentlich, M. Weinberger et G. Seroussi : Embedded block coding in jpeg2000. Rap. tech., Hewlett-Packard Company, 2001. [p. 59]

D. S. Taubman et M. W. Marcellin : *JPEG 2000, Image Compression Fundamentals, Standards and Practice*. Kluwer Academic, 2002. [p. 54, 59 et 100]

A. Vetro, A. M. Tourapis, K. Müller et T. Chen : 3d-tv content storage and transmission. *Proceedings of the IEEE - Special Issue : 3D Media and Displays*, 99:626 – 642, 2011. [p. 107 et 127]

G. K. Wallace : The jpeg still image compression standard. *Communications of the ACM*, 34 (4):30 – 44, Avril 1991. [p. 49]

M. J. Weinberger, G. Seroussi et G. Sapiro : Loco-i : A low complexity, context-based, lossless image compression algorithm. *In Data Compression Conference*, 1996. [p. 83, 91 et 100]

T. Welch : A technique for high-performance data compression. *IEEE Computer*, 17(6):8–19, 1984. [p. 39]

G. Wetzstein, D. Lanman, W. Heidrich et R. Raskar : Layered 3d : tomographic image synthesis for attenuation-based light field and high dynamic range displays. *In ACM SIGGRAPH 2011 papers*, SIGGRAPH '11, p. 95 :1–95 :12, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0943-1. URL http://doi.acm.org/10.1145/1964921.1964990. [p. 22]

I. H. Witten, R. M. Neal et J. G. Cleary : Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520 – 540, June 1987. [p. 35]

X. Wu : Context selection and quantization for lossless image coding. *In* J. A. Storer et M. Cohn, éds : *DCC '95, Data Compression Conference*, p. 453, Los Alamitos, CA, 1995. IEEE Computer Society Press. [p. 43 et 100]

X. Wu : An algorithmic study on lossless image compression. *In* J. A. Storer et M. Cohn, éds : *DCC '96, Data Compression Conference*, Los Alamitos, CA, 1996. IEEE Computer Society Press. [p. 43 et 100]

H. Yamanoue : The differences between toed-in camera configurations and parallel camera configurations in shoting stereoscopic images. *In IEEE Conference on Multimedia and Expo*, 2006. [p. 23 et 24]

B. Yeo et B. Liu : Volume rendering of dct-based compressed 3d scalar data. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):29 – 43, Mars 1995. [p. 146]

S. Yoon, E. Lee, S. Kim et Y. Ho : A framework for representation and processing of multi-view video using the concept of layer depth image. *Journal of VLSI Signal Processing*, 46:87 – 102, 2007. [p. ix, 79, 120, 121, 122, 123 et 128]

M. Zamarin, S. Milani, P. Zanuttigh et G. Cortelazzo : A novel multi-view image coding scheme based on view-warping and 3d-dct. *Journal of Visual Communication and Image Representation*, 21:462 – 473, 2010. [p. 128 et 145]

L. Zhang, X. Wu et P. Bao : Real-time lossless compression of mosaic video sequences. *Real-Time Imaging, Elsevier*, 11:370 – 377, 2005. [p. 82]

M. Zhang : The jpeg and image data compression algorithms, 1990. [p. 49]

J. Ziv et A. Lempel : A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977. [p. 38]

J. Ziv et A. Lempel : Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978. [p. 39]

Publications

B. Battin, P. Vautrot, D. Debons et L. Lucas : A new near-lossless scheme for multi-view image compression. *Dans IS&T/SPIE Electronic Imaging, Conference EI101 : Stereoscopic Displays and Application XXI*, San Jose, CA, jan. 2010. SPIE.

L. Lucas, B. Batin, C. Niquin, S. Prévost et Y. Remion : Acquisition and compression of 3D multiview images in real time. 3D STEREO MEDIA, Liège, Belgique, déc. 2010.

B. Battin, C. Niquin, P. Vautrot et L. Lucas : Multiview image compression based on LDV scheme. *Dans IS&T/SPIE Electronic Imaging, Conference EI101 : Stereoscopic Displays and Application XXII*, San Francisco, CA, jan. 2011. SPIE.

Résumé

Le développement de la "3D" dans l'industrie cinématographique ainsi que la mise sur le marché de consoles telles que la Nintendo 3DS et d'écrans stéréoscopiques utilisant des lunettes (actives ou passives) accrédite la démocratisation du relief dans notre rapport à l'image. L'une des dernières technologies en matière de restitution relief est l'écran auto-stéréoscopique : il permet l'affichage simultané de plusieurs vues d'une même scène (généralement entre 2 et 9) et ne nécessite pas le port de lunettes pour percevoir le relief. Les séquences auto-stéréoscopiques représentent un volume conséquent de données (lié au nombre de points de vue) qui est encore accentué par le fait que la technologie associée aux écrans est en constante évolution. En effet, les écrans actuels proposent des résolutions et des fréquences d'affichage de plus en plus élevées avec, notamment, l'arrivée prochaine sur le marché du standard UHDTV.

Ces différents facteurs tendent à produire des volumes de données toujours plus conséquents qui doivent être compressés pour permettre leur transmission sur des réseaux ou pour faciliter leur stockage. Le travail de thèse, qui a été mené dans le cadre du projet "Cam-Relief", a pour objectif le développement de solutions logicielles dédiées à la compression multi-vues de séquences auto-stéréoscopiques. Chacune de nos contributions vise à répondre à un des besoins spécifiques suivant : la compression temps réel des séquences issues de nos systèmes d'acquisition, la compression sans perte de séquences destinées à la post-production ainsi qu'une alternative au standard de compression multi-vues actuel : H.264/MultiView-Coding.

Nous présentons dans ce mémoire trois méthodes de compression multi-vues, chacune répondant aux trois besoins respectifs énoncés ci-dessus. La première, appelée MICA (Multiview Image Compression Algorithm), est un algorithme de compression multi-vues temps réel qui exploite la corrélation inter-vues présente au sein des séquences auto-stéréoscopiques en utilisant le principe d'images de différence. Ces images de différence vont aussi nous permettre de mettre en avant les zones de l'image soumises à la parallaxe et ainsi permettre de les préserver le plus possible d'importantes distorsions. La deuxième contribution, nommée Multiview-LS (Lossless), est une adaptation au cas multi-vues de l'algorithme JPEG-LS. En modifiant la structure du schéma de prédiction de JPEG-LS, notre algorithme permet l'exploitation des corrélations temporelles et inter-vues spécifiques aux séquences auto-stéréoscopiques. Le troisième schéma de compression, enfin, propose un algorithme de compression basé sur l'approche LDI (Layered-Depth Image). La génération de celui-ci est basée sur une approche innovante utilisant des cartes de disparité entières. Nous proposons deux schémas dédiés à la compression de l'information chromatique du LDI : un basé sur l'utilisation de la DCT et l'autre basé sur l'utilisation de la DWT. L'information de disparité est quant à elle codée à l'aide d'un algorithme de compression sans perte.