Transmodalité de flux d'images de synthèse

Pierre-Olivier Rocher

Thèse

Pour obtenir le grade de

Docteur de l'Université Jean Monnet de Saint-Étienne

Discipline : génie informatique, informatique

Présentée et soutenue publiquement le 31 octobre 2014
par

Pierre-Olivier Rocher

Ingénieur Télécom Saint-Étienne

Transmodalité de flux d'images de synthèse

Directeur de thèse :
Jacques Fayolle

Co-encadrants de thèse :
Christophe Gravier & Marius Preda

Composition du Jury :

Jean-Claude Dufourd	Professeur à Télécom ParisTech	*Rapporteur*
Valeriu Vrabie	Maître de conférences à l'Université de Reims	*Rapporteur*
Frédérique Laforest	Professeur à l'Université Jean Monnet	*Examinateur*
Jacques Fayolle	Professeur à l'Université Jean Monnet	*Directeur*
Christophe Gravier	Maître de conférences à l'Université Jean Monnet	*Co-encadrant*
Marius Preda	Maître de conférences à Télécom SudParis	*Co-encadrant*

Numéro d'ordre : XXXXXX – Version [206EE35]
Ce document est signé numériquement.

Notes

Remerciements

Cette thèse (projet GILMOUR) est l'aboutissement de quatre années de travail au sein de l'équipe SATIn (qui fait partie du LT2C) de Télécom Saint-Étienne et du département ARTEMIS/GRIN de Télécom SudParis. Elle a été financée par l'Institut Mines-Télécom, Télécom Saint-Étienne et par Saint-Étienne Métropole.

Je tiens tout d'abord à remercier Jacques Fayolle de m'avoir accueilli au sein de son équipe de recherche et de son école. Je remercie aussi Christophe Gravier et Marius Preda de m'avoir accompagné durant ces quatre années.

Un grand merci à Julien Subercaze et Frédérique Laforest, qui depuis leur arrivée ont participé d'une manière ou d'une autre à la réalisation de cette thèse.

Je tiens à remercier tout particulièrement Jean-Jacques Rousseau pour son aide durant les derniers mois de thèse, et de manière plus générale les personnels du LT2C et d'ARTEMIS.

Mes remerciements vont aussi à Jean-Claude Dufourd et Valeriu Vrabie pour l'intérêt qu'ils ont porté à mon travail en acceptant d'être rapporteurs.

Merci aux anciens de l'équipe qui ont brillamment terminé leur thèse et qui se sont depuis envolés vers leur nouvelle carrière : Benjamin, Yves-Gaël, Ivica et Antoine.

Merci bien sûr aux membres actuels des deux équipes de recherche que j'ai cotoyés durant plus de quatre ans : Nicolas, Mérième, Jules, Abderrahmen, Syd, Nicolas, Christian, Christina, Christophe ainsi qu'à la société Craftsmen représentée par Mehdi et Kevin, pour ces heures de débats philosophiques sur des sujets informatiques et sociétaux passionnants. Certains d'entre eux ont d'ailleurs contribué à la partie expérimentale de ce travail. Je tiens finalement à citer dans ces lignes Pierre-Yves avec qui les débats (scientifiques ou de tout autre nature) sont toujours fort intéressants.

Je remercie également l'ensemble des personnes avec qui j'ai pu travailler tant à Télécom Saint-Étienne qu'à Télécom SudParis et tout particulièrement le personnel administratif.

Signature numérique.

Table des matières

Liste des figures

Liste des tableaux

Liste des Algorithmes

Liste des équations

Liste des acronymes

AAM	*Active Appearance Model*
ADSL	*Asymmetric Digital Subscriber Line*
AFX	*Animation Framework eXtension*
API	*Application Programming Interface*, glossaire : API
ARDECO	*Automatic Region DEtection and COnversion*
ARTEMIS	*Advanced Research and TEchniques for Multidimensional Imaging Systems*
AVU	Attention Visuelle de l'Utilisateur
BGD	*Bipartite Graph-based Decomposition*
BIFS	*Binary Imaging For Scene*
BMP	*Bitmap*
BSD	*Berkeley software distribution license*
CABAC	*Context-Adaptive Binary Arithmetic Coding*
CAO	*Computer-Aided Design*
CAVLC	*Context-Adaptive Variable-Length Coding*
CCIR 601	Format vidéo en 525×625
CODEC	*COde DECode*
CPU	*Central Processing Unit*
CVMP	*Conference on Visual Media Production*
DASH	*Dynamic Adaptive Streaming over HTTP*
DBMR	*Decomposing Binary Matrices into Rectangles*
DCT	*Discrete Cosine Transform*
DTD	*Distance-Transform Decomposition*
DVD	*Digital Versatile Disc*
DWT	*Discrete Wavelet Transform*
EPS	*Encapsulated PostScript*
FMO	*Flexible Macroblock Ordering*
FPS	*First-Person Shooter*
GAM	*Game Attention Model*
GNU	*GNU's Not Unix !*
GNU GPL	*GNU General Public License*
GNU LGPL	*GNU Lesser General Public License*
GOP	*Groupe Of Pictures*
GPAC	*GPAC Project on Advanced Content*
GPU	*Graphics Processing Unit*
GRIN	*GRaphics and INteractive Media*
GSL	*Gnu Scientific Library*
HDTV	*High Definition TeleVision*

HEVC	*High Efficiency Video Coding*
HLS	*HTTP Live Streaming*
HPC	*High-Performance Computing*
HTTP	*Hyper Text Transfert Protocole*
IaaS	*Infrastructure-as-a-Service*
IBM	*International Business Machines*
IBR	*Image Block Representation*
IDCT	*Inverse Discrete Cosine Transform*
IDWT	*Inverse Discrete Wavelet Transform*
IEC	*International Electrotechnical Commission*
IFS	*Iterated Function System*
INSA	*Institut National des Sciences Appliquées*
IP	*Internet Protocol*
ISO	*International Standard Organisation*
ISOBMFF	*ISO Base Media File Format*
ITU	*International Telecommunication Union*
JPEG	*Joint Photographic Experts Group*
KLT	*Karhunen–Loève theorem*
LOD	*Level Of Detail*
LT2C	Laboratoire Télécom Claude Chappe
MED	*Morphological Decomposition*
MOS	*Mean Opinion Score*
MPD	*Media Presentation Description*
MPEG	*Motion Picture Expert Group*
MPEG-2 (M2TS)	MPEG-2 *Transport Stream*
MPEG-TS	MPEG *Transport Stream*
MSSIM	*Multi-scale Structural Similarity*
NURBS	*Non Uniform Rational B-Splines*
PC	*Personal Computer*
PCCR	*Pupil Centre Corneal Reflection*
PIFS	*Partitioned Iterated Function System*
PNG	*Portable Network Graphics*
PSNR	*Peak Signal to Noise Ratio*
QoE	*Quality of Experience*
QoS	*Quality of Service*
QTD	*Quad Tree Decomposition*
RFC	*Requests For Comments*
RGB	*Red Green Blue*
ROI	*Region Of Interest*
RS	*Row Segmentation*
RTCP	*Real-time Transport Control Protocol*
RTP	*Real-time Transport Protocol*
RTS	*Real-Time Strategy*
RTSP	*Real-Time Streaming Protocol*
SATIn	Sécurité Algorithmes Télécoms Intégration
SDK	*Software Development Kit*
SDP	*Session Description Protocol*
SDTV	*Standard-Definition TeleVision*
SIF	*Source Input Format*
SVC	*Scalable Video Coding*

SVD	*Singular Value Decomposition*
SVG	*Scalable Vector Graphics*
TCLAP	*Templatized C++ Command Line Parser*
TCP	*Transport Control Protocol*
UDP	*User Datagram Protocole*
UIT-T	*International Telecommunication Union*
VIM	*Vector Imaging Model*
VLC	*Variable-Length Code*
VLSI	*Very-Large-Scale Integration*
VO	*Video Objects*
VPI	*Vectorized Photographic Images*
VSV	*Vectorized Streaming Video*
Wi-Fi	*Wireless Fidelity*
WSRM	*Walk Stack Remove Merge*, glossaire : WSRM
XML	*eXtended Markup Language*

Glossaire

1080p	Résolution spatiale de 1920×1080 pixels.
2K	Résolution spatiale de 2880×1620 pixels.
480p	Résolution spatiale de 720×480 pixels.
4K	Résolution spatiale de 3840×2160 pixels.
720p	Résolution spatiale de 1280×720 pixels.
API	Ensemble normalisé de classes, de méthodes et/ou de fonctions servant d'interface pour la délégation de services entre logiciels.
DirectX	Collection propriétaire de bibliothèques destinées à la programmation d'applications multimédia.
H.262	Standard d'encodage vidéo de l'UIT-T, équivalent de MPEG-2.
H.264	Standard d'encodage vidéo de l'UIT-T, équivalent de MPEG-4 AVC.
H.265	Standard d'encodage vidéo de l'UIT-T, équivalent de HEVC.
JPEG2000	Second système de compression d'images proposé par le JPEG.
mp4	Conteneur pour encapsuler des données de type multimédia (audio ou vidéo essentiellement).
MPEG-1	Premier standard d'encodage vidéo du MPEG.
MPEG-2	Deuxième standard d'encodage vidéo du MPEG, il s'agit d'une mise à jour de MPEG-1.
MPEG-4	Nouveau standard d'encodage vidéo du MPEG.
MPEG-4 AVC	Partie 10 du standard MPEG-4, publié en 2004.
OpenGL	Pendant libre de DirectX.
SVGZ	Format SVG compressé avec le standard gzip.
WSRM	Algorithme de recherche de la liste des plus grands rectangles dans une image binaire. Se reporter à l'annexe D pour plus de détails.
x264	Implémentation *open source* de la norme MPEG-4 AVC.

Première partie

Introduction

Chapitre 1

Contexte & objectifs

Durant ces dernières années, le développement du monde de l'informatique a été fulgurant. Les ordinateurs personnels, les réseaux et leurs interconnexions qui ont constitué Internet, sont devenus une brique essentielle du monde d'aujourd'hui. Au fil du temps, les différentes technologies entrant en jeu ont évolué. Les performances sans cesse améliorées des matériels comme des logiciels ont permis de gagner en productivité. L'architecture client/serveur, qui est un socle important de l'Internet actuel a elle aussi subi de nombreuses modifications. On peut considérer que son évolution s'est déroulée en trois étapes : l'architecture *mainframe*, l'apparition du PC [1] et enfin l'avènement du *cloud*. La première version de cette architecture était basée sur l'utilisation d'un ordinateur central auquel se connectaient un grand nombre de clients. À cette époque, le coût prohibitif d'un ordinateur ne permettait pas à chaque employé d'en posséder un. Les utilisateurs travaillaient donc sur cet ordinateur central grâce à une connexion limitée. L'apparition de l'ordinateur personnel, inventé et proposé par IBM [2] en 1981 modifie profondément le modèle. L'ordinateur personnel, plus communément appelé PC devient abordable. La majorité du travail est alors réalisé sur l'ordinateur de chaque employé. C'est la fin des architectures *mainframe*, les serveurs ont alors comme rôle principal des tâches de synchronisation. Le réseau reliant clients et serveur évolue en offrant de meilleures performances globales. De nos jours, l'utilisation massive du *cloud computing* remet en avant le paradigme du *remote computing*. Les tâches requérant une puissance de calcul importante sont massivement déportées dans le *cloud* grâce à un réseau efficace. Le terminal client, c'est-à-dire le matériel permettant à l'utilisateur d'interagir avec le système distant est simplement chargé de l'affichage du résultat.

Actuellement, la bande passante réseau disponible est largement consommée par la diffusion de vidéos. Selon [75], 90% de la capacité sera dévolue à cet usage aux environs de 2017. Selon les prévisions, cette proportion va continuer à augmenter dans les prochaines années. Dans le même temps, le nombre d'acteurs faisant appel à la vidéo comme support de communication ne cesse d'augmenter. Aujourd'hui la vidéo est omniprésente, que ce soit sur les sites spécialisés, sur les réseaux sociaux, sur les plates-formes de télévision de rattrapage (aussi

1. *Personal Computer*
2. *International Business Machines*

connue sous les noms de *replay TV* ou *catch-up TV*), ou encore dans les versions électroniques de la presse écrite. Pour permettre à un nombre toujours plus grand de personnes d'accéder à ces contenus, deux solutions viennent à l'esprit : renforcer les capacités des connexions réseau ou travailler sur la compression de ce type de contenus. Actuellement, les connexions basées sur la paire de cuivre (historique) ne permettent plus de satisfaire des besoins toujours plus importants. Pour apporter une solution viable à ce problème, la solution de la fibre optique semble s'imposer partout dans le monde. Ce nouveau type de vecteur va permettre à chaque utilisateur d'avoir accès à de nombreux services inédits, tels que la possibilité de visionner plusieurs flux vidéo haute définition voire en ultra haute définition en même temps. La diversité des terminaux (mobiles ou non) permettant de visionner ces contenus est devenue telle que chaque vidéo disponible sur Internet l'est en différents formats (relatifs aux différentes méthodes de compression), mais aussi en diverses qualités d'encodage (en fonction de la résolution spatiale notamment). Ainsi un utilisateur regardant une vidéo sur son téléphone mobile ne reçoit pas le même flux vidéo de la part du pourvoyeur de service que s'il avait visionné cette même vidéo avec un ordinateur. Le but est d'adapter le service en fonction des besoins du client, mais aussi en fonction du matériel qui accède à ce contenu. Il s'agit d'un problème d'optimisation : nul besoin d'offrir un flux vidéo en haute définition à un terminal mobile dont la résolution native de l'écran serait inférieure à celle utilisée durant l'encodage vidéo. Ceci met en avant d'autres problèmes : doit-on pouvoir fournir autant de flux vidéo qu'il existe de résolutions spatiales différentes ? Comment prendre en compte les aléas de performance des réseaux empruntés ? La réponse est assez ardue, puisque si le premier point peut être résolu assez rapidement, le deuxième apporte un degré d'incertitude. Certains grands acteurs comme Google, à travers son service de vidéo YouTube proposent déjà des solutions. Il en est d'ailleurs de même pour tous les autres fournisseurs de contenus vidéo. La solution technique n'est pas forcément la même, mais une capacité d'adaptation est quasiment toujours présente. Une autre solution est en passe d'être mise en œuvre par les poids lourds du secteur tels qu'Amazon ou Microsoft, une solution qui permettrait d'adapter encore plus finement les besoins. Il s'agit d'utiliser la puissance fournie par le *cloud* dans le but de réaliser un transcodage en temps réel et d'envoyer à tous les clients un flux vidéo adapté.

Dans le même temps, les solutions de *remote computing* et de *remote rendering* sont de plus en plus utilisées. Ces applications requièrent des besoins particuliers, notamment dans l'établissement et le maintien de boucles interactives en temps réel qui nécessitent des latences faibles et des bandes passantes parfois conséquentes (cas du *cloud gaming*). Pour pouvoir utiliser toutes ces applications de manière simultanée, plusieurs parties constituant la chaîne de rendu distant doivent être optimisées, ce qui inclut l'encodage vidéo. De manière traditionnelle, il existe trois axes majeurs de recherche pour diminuer la quantité d'informations à transmettre : la résolution spatiale, la résolution temporelle, la qualité. Ces approches considèrent toujours la vidéo comme un ensemble de pixels (le pixel est l'unité de base) et il existe un seuil d'adaptation du débit au-delà duquel l'utilisation d'un encodeur classique (exclusivement à base de pixels donc) n'est plus possible. Il est alors nécessaire d'utiliser une nouvelle façon de représenter l'information. Il est démontré dans la littérature que pour certains types d'images (plutôt uniformes) une représentation paramétrique (via l'uti-

lisation de primitives graphiques) est plus compacte. Utiliser les informations supplémentaires issues de la nature synthétique des images est une première étape. Néanmoins, les limitations intrinsèques afférentes à la réduction du débit binaire dans l'espace de représentation pixel ne peuvent être surmontées que par l'utilisation d'une approche hybride.

La solution proposée dans ce travail doctoral combine l'utilisation de l'approche pixel traditionnelle et d'une approche basée sur l'utilisation de primitives graphiques associées à des profils couleur. L'encodeur proposé se base sur les régions de l'image effectivement observées par l'utilisateur. Cette approche permet d'appréhender la façon dont les utilisateurs observent une scène de synthèse et ainsi d'adapter l'encodage en utilisant différentes qualités. Pour cela, un oculomètre a été utilisé pour la détection des régions d'intérêt. Une corrélation avec ces régions et des caractéristiques bas niveau de l'image a ensuite été construite. Grâce à un traitement temps réel comme une détection des contours, une classification a tout d'abord été établie. Par la suite, l'encodeur met en œuvre divers **espaces de représentation** pour chaque classe ainsi que plusieurs modalités d'encodage. Les modalités mises en œuvre dans notre implémentation sont constituées de l'espace de représentation pixel, ainsi que de celui formé par l'utilisation des primitives graphiques et de leurs profils couleur. Contrairement aux approches conventionnelles qui considèrent une vidéo comme un ensemble de pixels, notre approche est basée sur l'hypothèse qu'un partitionnement dynamique de chaque image constituant la vidéo (en approximant certaines régions par une représentation à base de primitives graphiques et de profils couleur) aura pour conséquence une baisse du débit binaire. Par conséquent, la nouvelle représentation de la vidéo est une **combinaison dynamique** de régions formées soit de pixels, soit de primitives graphiques et de profils couleur. Outre les moyens traditionnels de contrôle du débit binaire d'une vidéo (qui ont déjà été cités dans le paragraphe précédent), nous introduisons une nouvelle dimension de contrôle qui est la **modalité** de représentation.

Pour construire le flux vidéo multimodal, un algorithme de traitement des images est nécessaire. Cette tâche va inévitablement augmenter la complexité, mais aussi le temps de calcul. Néanmoins, des modifications au sein de l'encodeur vidéo permettent de minimiser cette surcharge de calcul. Un module prend la décision sur la modalité à utiliser pour chaque région. Chaque modalité est ensuite compressée avec son encodeur spécifique. Plusieurs optimisations ont été introduites, agissant sur chaque modalité utilisée dans le but de limiter le nombre d'informations redondantes. À titre d'exemple, certaines informations connues sont directement injectées dans l'encodeur pixel permettant à la fois une réduction du temps d'analyse, mais aussi une amélioration de la compacité. Cette thèse s'inscrit dans le développement d'approches permettant d'adresser ces enjeux d'hétérogénéité d'environnements tout en passant à l'échelle du pseudo-temps réel, voire du temps réel tout en prenant en compte le ressenti de l'utilisateur. Plus particulièrement, le but de ce travail est entre autres de fournir une brique logicielle capable de travailler sur des flux vidéo en temps réel, et adaptée notamment au rendu distant.

Chapitre 2

Organisation du document

Tout au long de ce document, un certain nombre d'acronymes sont utilisés. Une note de bas de page les définit lors de leur première utilisation, les occurrences suivantes sont pourvues d'un lien qui permet d'accéder à leur définition dans la table des acronymes. Un glossaire est aussi présent, il utilise un fonctionnement similaire. Certains acronymes sont explicités à l'aide d'une entrée dans le glossaire. Dans certaines figures, des liens hypertextes sont présents, ils sont signalés par l'utilisation de la couleur bleue. Ce manuscrit se subdivise en trois grandes parties, elles-mêmes constituées de différents chapitres. Un rapide résumé est donné ci-dessous pour chacune et chacun d'entre eux.

Introduction

La première partie (page 1) est constituée de deux chapitres introductifs.

Chapitre 1

Ce premier chapitre constitue l'introduction de ce manuscrit. Il expose de façon concise les motivations et les buts de ce travail.

Chapitre 2

Le second chapitre expose la structure utilisée pour la rédaction de ce document. Il permet au lecteur d'avoir une vue d'ensemble sur le déroulé de cette thèse.

État de l'art

La deuxième partie (page 11) est consacrée à un état de l'art. L'ensemble des informations utiles pour appréhender la contribution proposée dans cette thèse est analysé. Le déroulement de l'état de l'art est représenté de manière visuelle par la figure 2.1, chaque étape étant ensuite décrite de manière plus précise par les résumés ci-après.

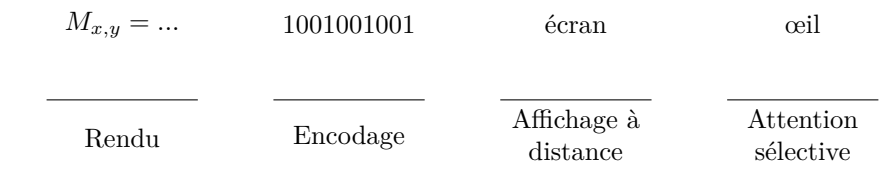


FIGURE 2.1 – Vue d'ensemble des grandes étapes d'une chaîne de rendu à distance

Chapitre 3

Ce chapitre s'intéresse tout d'abord aux différentes représentations utilisées pour le stockage et la manipulation des images numériques. Il est en effet possible de représenter les images numériques de bien des manières. Les principales représentations utilisées ainsi que les méthodes de compression qui leur sont associées sont détaillées.

Chapitre 4

Après le monde des images, ce chapitre aborde les différents aspects mis en œuvre lors de la compression d'une vidéo. Les différentes étapes ainsi que les techniques rendant possible la compression sont passées en revue. Un accent particulier sera mis sur les encodeurs utilisant plusieurs types de représentation pour la compression d'une même vidéo.

Chapitre 5

Après les techniques de compression dédiées aux mondes de l'image et de la vidéo, ce chapitre s'intéresse aux différentes approches utilisées pour une application particulière : le rendu à distance. Les méthodes de transfert de données, mais aussi de compression entre un client et son serveur référant sont détaillées.

Chapitre 6

Ce dernier chapitre de l'état de l'art fait le lien entre les trois précédents, en s'intéressant plus particulièrement au *cloud gaming* qui est une application du rendu à distance. Outre les solutions disponibles (tant libres que propriétaires), les capacités d'adaptation de ces systèmes seront explorées. La notion émergente d'attention sera finalement introduite. Une conclusion résumant les travaux existants conclut cet état de l'art (page 87), tout en introduisant les différents objectifs de cette thèse.

Contribution originale

Cette troisième partie (page 89) met en avant le travail effectué durant cette thèse au moyen de cinq chapitres.

Chapitre 7

Le chapitre détaille comment l'attention de l'utilisateur a été prise en compte dans notre système. À travers l'utilisation d'un oculomètre et d'une expérimentation regroupant différents types de jeux et plusieurs joueurs expérimentés, des informations sur les zones effectivement regardées par le joueur lors d'une session de jeu ont pu être extraites.

Chapitre 8

Ce chapitre introduit de façon conceptuelle le fonctionnement d'un transcodeur multimodal, qui sera nommé **transmodeur** [1] dans la suite de ce document. Toutes les notions utilisées par la suite sont définies et explicitées. Un exemple simple d'une telle compression est mis en avant pour permettre au lecteur une bonne appréciation du concept mis en œuvre.

Chapitre 9

Suite à l'introduction du concept de transmodeur ce chapitre décrit de manière détaillée le fonctionnement de cet encodeur multimodal. Toutes les étapes depuis l'obtention d'une image brute jusqu'à l'empaquetage des données issues des différentes modalités sont passées en revue. Les explications sont basées sur l'utilisation d'une approche bimodale, qui comporte donc deux modalités de nature différente.

Chapitre 10

Après la présentation du fonctionnement détaillé d'un transmodeur bimodal, l'implémentation qui en a été faite est présentée. L'architecture logicielle, les librairies externes utilisées ainsi que l'utilisation des différents programmes et librairies créés sont détaillées. Une discussion quant à la façon d'utiliser la puissance du *cloud* pour paralléliser le processus de transmodage [2] conclut ce chapitre.

Chapitre 11

Ce chapitre s'intéresse aux différents tests et expérimentations réalisés pour valider le fonctionnement du prototype d'encodeur multimodal. Pour cela, l'encodeur multimodal a été intégré à la plate-forme Kusanagi, dans le cadre du projet XLcloud. Les tests ont été conduits sur des vidéos issues de sessions de *cloud gaming* utilisant le jeu Doom3. Ces vidéos de base ne sont pas compressées et permettent donc d'utiliser tous les détails présents dans le jeu. Une conclusion clôture cette partie en résumant les différentes contributions proposées.

Conclusion

La dernière partie (page 149) comporte un dernier chapitre qui conclut ce travail.

1. Transcodeur supportant plusieurs modalités lors de l'encodage
2. Équivalent d'un transcodage supportant plusieurs modalités lors de la compression

Chapitre 12

Ce chapitre résume tout le travail accompli durant cette thèse en proposant un résumé des contributions proposées. Différentes perspectives susceptibles de compléter et/ou d'améliorer le travail déjà réalisé sont finalement abordées.

Deuxième partie

État de l'art

Chapitre 3

Représentation et compression des images statiques

Sommaire

Depuis le début de l'ère informatique, le stockage des données est un problème en soi. Pour le résoudre, deux approches logiques sont possibles : compresser les données à enregistrer dans le but de réduire leur taille ou augmenter la capacité de stockage du système. Nous nous focalisons ici sur la première solution, qui s'intéresse à la compression avec ou sans perte. Une compression sans perte permet de s'assurer que l'information initiale et l'information restituée après la décompression sont exactement identiques. À l'inverse, lors d'une compression avec perte une partie de l'information est définitivement perdue. Évidemment, tout dépend de la nature des données à compresser. Par exemple, un texte ou un programme ne peuvent être compressés que par l'utilisation d'algorithmes de compression sans perte puisque la moindre perte d'information rendra le texte indéchiffrable pour l'humain et le programme ininterprétable par la machine. Il n'en est pas de même pour les données de type audio/vidéo où il est possible d'utiliser des compressions avec perte, le but étant de supprimer des informations peu utiles, voire redondantes. Dans ce cas, lors de la décompression, une partie seulement de l'information originelle sera toujours présente.

On dit couramment qu'une image vaut mille mots : c'est sans doute pour cela qu'elles sont aujourd'hui (comme hier d'ailleurs) omniprésentes. On les trouve toujours dans les journaux, les magazines ou placardées dans les rues mais elles sont aussi très utilisées dans les nouveaux médias : ordinateurs, smartphones, presse en ligne, Internet... L'utilisation et l'affichage des images par nos appareils numériques s'appuient sur la manière dont est représentée l'image. Comme l'introduction de ce chapitre l'a mentionné, il existe plusieurs procédés pour représenter une image. On dénote dix espaces de représentation différents, ceux-ci sont énumérés ci-dessous, accompagnés d'une brève description :

1. Pixel : approche matricielle avec comme élément de base le pixel ;
2. Vecteur : l'image est composée d'un ensemble de primitives graphiques ;
3. Quantification vectorielle : réduction de la taille de l'espace utilisé ;
4. Codage basé sur un modèle : modélisation de la structure et du mouvement ;
5. Fractale : champ scalaire ;
6. Paramétrique : équation mathématique définissant un ensemble géométrique ;
7. Spectrale : analyse des composantes fréquentielles ;
8. Syntaxique : génération de la texture grâce à une grammaire ;
9. Structurelle : définition de structure de base et de règles pour modéliser l'agencement spatial ;
10. Stochastique : approches autorégressives à partir de processus Markoviens.

La compression de ces espaces s'appuie naturellement sur la méthode utilisée pour la représentation de l'image, il existe donc un grand nombre d'algorithmes permettant de compresser les images, que ce soit avec ou sans perte. Certaines applications requièrent une compression sans perte, c'est par exemple le cas des images médicales où chaque détail peut avoir une importance cruciale, mais aussi des images vectorielles de plus en plus utilisées sur le Web. Pour les applications grand public, des compressions avec pertes sont utilisées, une perte limitée d'information n'étant pas gênante.

Parmi les espaces de représentation, on peut mettre en avant les deux ap-

proches les plus utilisées à ce jour : la méthode pixel et la méthode vecteur. Leurs définitions et fonctionnements sont largement détaillés dans la suite de ce chapitre. Les méthodes de représentation restantes, plus marginales sont regroupées dans une troisième partie.

3.1 Méthode pixel

Il s'agit sans nul doute de la méthode la plus utilisée pour le stockage d'images ou de vidéos.

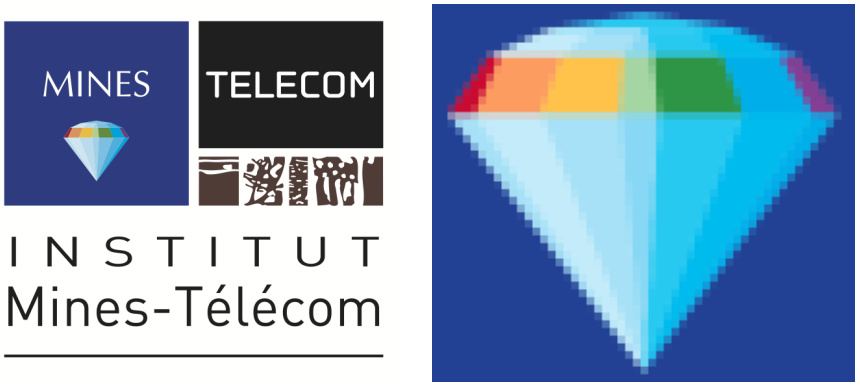
3.1.1 Principes de base

La méthode matricielle, en anglais *raster* ou encore *bitmap* définit donc littéralement une image comme une matrice de largeur n et de hauteur m contenant $n \times m$ éléments. Chaque élément constitue une unité de base, qui dans le cas d'une image est appelée pixel. Les pixels sont généralement de forme carrée et utilisent un espace de couleur propre au format choisi pour stocker l'image (voir 3.1.2). Du fait même que l'image est constituée d'un nombre fini de pixels, certaines opérations, comme l'agrandissement par exemple, peuvent poser problème. En effet, en utilisant une résolution spatiale correcte, les pixels sont trop petits pour être distingués par l'œil humain. L'image apparaît alors non pas comme un ensemble de petits éléments, mais comme un tout. Si l'on effectue un agrandissement trop prononcé sur une partie de l'image, les pixels vont peu à peu se dévoiler : l'image obtenue va alors rapidement se dégrader, du fait du manque d'information à afficher. En conclusion, on peut affirmer que ce type de représentation est dépendant de la résolution spatiale. En d'autres termes, il n'est pas possible d'effectuer des agrandissements vers une résolution donnée sans une perte de qualité. Ceci implique que pour facilement être capable de pratiquer des agrandissements, il faut utiliser une résolution spatiale importante, et donc utiliser plus de pixels. La taille du fichier stockant les données va donc augmenter en conséquence. En plus du nombre de pixels, la taille finale est aussi liée à l'espace colorimétrique utilisé (voir 3.1.2), mais aussi à la précision voulue pour représenter l'image (nombre de couleurs supporté).

Ce type de représentation est largement utilisé pour la manipulation des images naturelles (de photographies donc). En incluant les formats propriétaires, une image matricielle peut être stockée dans des centaines de formats différents, chacun faisant l'objet d'une norme le définissant. Certains utilisent des compressions sans perte (BMP [1], PNG [2]), d'autres avec pertes (JPEG [3]). De nos jours, le format le plus utilisé est sans doute le JPEG, bien qu'il soit nettement moins performant que le JPEG2000 [4].

La figure 3.1 permet de mettre en avant l'un des problèmes les plus importants inhérents à cette représentation : lors d'un agrandissement, il existe un seuil à partir duquel la quantité d'information devient trop faible. À ce stade-là, les pixels apparaissent et la qualité de l'image se dégrade rapidement : on voit nettement apparaître les pixels.

1. *Bitmap*
2. *Portable Network Graphics*
3. *Joint Photographic Experts Group*
4. Second système de compression d'images proposé par le JPEG.



(a) Taille réelle. (b) Agrandissement d'une partie de l'image.

FIGURE 3.1 – Images représentées sous la forme matricielle.

Pour compresser une image de manière efficace, il faut utiliser une compression avec perte. Tous les encodeurs de ce type utilisent le même schéma de principe, présenté dans la figure 3.2. Ces quatre étapes principales sont décrites en détail dans les sections suivantes. L'image brute en entrée et sa représentation compressée en sortie ne sont pas représentées dans la figure.

3.1.2 Espaces colorimétriques

D'autres facteurs doivent être pris en compte lorsque l'on parle de compression d'images ou de vidéos : la perception de l'œil humain. Les espaces colorimétriques sont un domaine relativement complexe puisque les couleurs elles-mêmes n'existent pas. Il s'agit juste d'une façon de représenter ce que l'on voit. Une des manières les plus utilisées pour la représentation de couleurs est la combinaison de trois composantes : le rouge, le vert et le bleu. Cet ensemble forme l'espace de couleurs RGB [5]. Chaque couleur correspond à une longueur d'onde donnée, elle-même choisie à partir des caractéristiques physiologiques des cellules réceptrices composant le système de détection de l'œil. Le système RGB est largement utilisé, puisque les couleurs sont restituées à l'aide de pixels composés eux-mêmes de ces trois composantes. Chaque pixel est stocké sur 24 b, soit 8 b par canal (16 millions de couleurs). Néanmoins pour certaines applications il est nécessaire de garantir une meilleure précision : le nombre de bits par pixel peut alors atteindre 10 voire 12 b. Le format RGB n'est pas le seul possible, d'autres existent comme le format YUV. Cette fois-ci, une couleur est décrite grâce à l'utilisation de la luminance (Y) et de la chrominance (U et V). Un des avantages de ce format est le fait que la majorité de l'information est présente dans la composante Y qui correspond (pratiquement) à la représentation monochrome d'une image en couleurs. Les informations de couleur sont – si nécessaire – reconstituées en utilisant les informations de chrominance. Ce format est de plus particulièrement bien adapté à l'œil humain qui est largement plus sensible au contraste qu'à la couleur. Les informations apportées par les composantes U et V peuvent alors être réduites sans pour autant affecter la qualité perçue

5. *Red Green Blue*

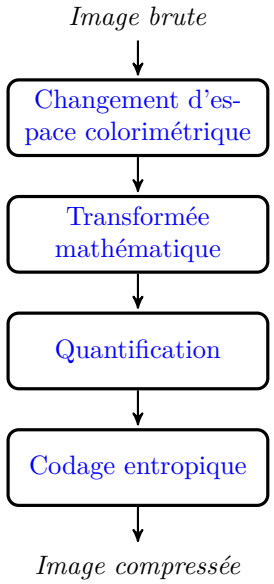


FIGURE 3.2 – Schéma de principe d'un encodeur avec perte utilisant la représentation pixel.

grâce à un sous échantillonnage (*chroma-subsampling* en anglais).

Le format YCrCb, très similaire au format YUV est utilisé pour la compression d'images ou de vidéos numériques notamment par les standards JPEG et MPEG [6]. Il est défini par deux normes de l'ITU [7] [124, 137]. Chaque valeur est stockée par défaut sur 8 b. Les équations 3.1 et 3.2 expriment la relation mathématique existant entre les formats YCrCb et RGB.

$$\begin{aligned} Y &\leftarrow 0,299R + 0,587G + 0,114B \\ Cr &\leftarrow 0.713(R - Y) + \delta \\ Cb &\leftarrow 0.564(B - Y) + \delta \end{aligned} \tag{3.1}$$

$$\begin{aligned} R &\leftarrow Y + 1.403(Cr - \delta) \\ G &\leftarrow Y - 0.714(Cr - \delta) - 0.344(Cb - \delta) \\ B &\leftarrow Y + 1.773(Cb - \delta) \end{aligned} \tag{3.2}$$

$$\text{avec} \quad \delta = \begin{cases} 128 & \text{pour les images sur 8 bits ;} \\ 32768 & \text{pour les images sur 16 bits ;} \\ 0,5 & \text{pour les images en virgule flottante.} \end{cases}$$

Comme pour YUV, différentes versions de YCrCb existent, définies en fonction du sous échantillonnage utilisé sur les composantes de chrominance et notées 4:X:X. Le chiffre 4 représente un taux d'échantillonnage de 13,5 MHz, qui correspond à la fréquence standard pour la numérisation de vidéos analogiques. Les

6. *Motion Picture Expert Group*
7. *International Telecommunication Union*

deux autres chiffres représentent les taux d'échantillonnage de composantes Cb et Cr. Les quatre sous-échantillonnages les plus utilisés sont les suivants : 4:4:4, 4:2:2, 4:2:0 et 4:2:1. Dans la plupart des applications, le format YCrCb4:2:0 est utilisé, ce qui signifie que les composantes Cb et Cr ont une fréquence d'échantillonnage quatre fois moindre par rapport à Y.

3.1.3 Transformée mathématique

Tous les standards de compression utilisent une transformée dans leur processus. L'objectif est de convertir une image dans un nouveau domaine, défini par la transformée. Il en existe beaucoup, mais elles doivent satisfaire à quelques points précis pour pouvoir être utilisées :

a. Procurer un taux de compression optimal : le maximum d'informations doit être stocké dans un minimum de données ;
b. Être réversible, pour qu'une décompression soit possible ;
c. Être implémentable de manière efficace (en matière d'occupation mémoire, de temps processeur...).

Durant les trente dernières années, de nombreuses transformées ont été créées dans le but d'augmenter l'efficacité des algorithmes de compression dans le domaine de l'image et de la vidéo. On peut scinder ces transformées en deux groupes : certaines se basent sur la notion de bloc (en ensemble de pixels est alors considéré) comme la KLT [8], la SVD [9], ou encore la DCT [10] ; d'autres utilisent l'image entière comme la DWT [11]. Les plus utilisées d'entre elles sont la DCT [2] et la DWT [17] (et leurs équivalents inverses IDCT [12] et IDWT [13]), même si elles possèdent toutes deux des avantages et des inconvénients.

Comme leur nom l'indique, les transformées par bloc utilisent des sous-ensembles de l'image originale pour travailler. Si l'on prend pour exemple la norme JPEG, qui utilise la DCT, ces blocs ont une taille de 8×8 pixels. Les équations 3.3 et 3.4 présentent respectivement les expressions de la DCT et de l'IDCT utilisées dans JPEG. $S_{i,j}$ est la valeur du pixel à la position i, j dans un bloc, $S_{u,v}$ est la valeur du coefficient correspondante après application de la DCT.

$$S_{u,v} = \frac{1}{4} C_u C_v \sum_{i=0}^{7} \sum_{j=0}^{7} S_{i,j} \cos\left[\frac{(2i+1)\,u\pi}{16}\right] \cos\left[\frac{(2j+1)\,v\pi}{16}\right] \tag{3.3}$$

$$S_{i,j} = \frac{1}{4} \sum_{u=0}^{7} \sum_{v=0}^{7} C_u C_v S_{u,v} \cos\left[\frac{(2i+1)\,u\pi}{16}\right] \cos\left[\frac{(2j+1)\,v\pi}{16}\right] \tag{3.4}$$

8. *Karhunen–Loève theorem*
9. *Singular Value Decomposition*
10. *Discrete Cosine Transform*
11. *Discrete Wavelet Transform*
12. *Inverse Discrete Cosine Transform*
13. *Inverse Discrete Wavelet Transform*

$$\text{avec} \quad C_u C_v = \begin{cases} \frac{1}{\sqrt{2}} & \text{pour} \quad u, v = 0 \\ 1 & \text{sinon} \end{cases}$$

Les matrices $S_{u,v}$ et $S_{i,j}$ sont de même taille. La première est composée de coefficients issus du calcul de la DCT, la seconde est formée de la valeur des pixels d'une composante (ou de résidus dans le cas de la vidéo, voir 4.1.2). Chaque bloc de coefficients dans le domaine DCT peut être considéré comme une pondération des fonctions de base. Dans le cas d'une DCT utilisant des blocs de 8×8 pixels, il y a 64 fonctions de base, elles sont présentées dans la figure 3.3.

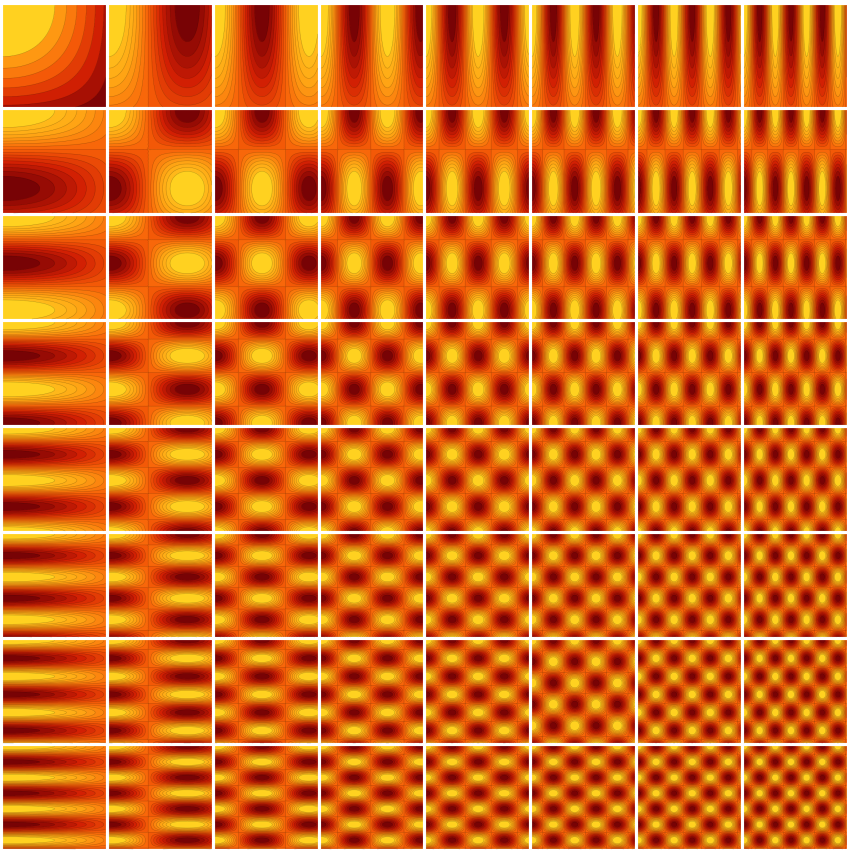


FIGURE 3.3 – Fonctions de base d'une DCT utilisant des blocs de 8×8 pixels.

La DWT est basée sur un ensemble de filtres utilisant des coefficients équivalents à des ondelettes discrètes. L'application de la DWT sur une image est illustrée grâce à la figure 3.4, la suite de ce paragraphe y fait référence pour la description du processus complet. La première étape utilise une paire de filtres, appliqués à un signal discret composé de N échantillons. Dans notre cas, l'image est considérée comme un signal bidimensionnel discret. Un filtre passe-haut et un filtre passe-bas sont appliqués sur l'image I, ils produisent deux nouvelles images représentant respectivement les basses (L) et les hautes fréquences (H). Les images L et H sont ensuite sous échantillonnées d'un facteur 2 dans la di-

rection x, produisant les images Lx et Hx. La troisième étape permet de former une image intermédiaire de même taille que l'image originale en regroupant les images Lx et Hx, comme illustré sur la figure 3.4. Dans la quatrième étape, l'image intermédiaire est à son tour filtrée avec un filtre passe-haut et un filtre passe-bas et donne pour résultat quatre images nommées LL, LH, HL et HH. Comme dans l'étape ②, un sous échantillonnage est réalisé sur ces quatre images, mais cette fois selon la direction y. L'ultime étape regroupe ces quatre images pour former une image ayant la taille de l'image d'origine, celle-ci forme le premier niveau de décomposition en ondelettes. Pour obtenir le second niveau de la décomposition en ondelettes, les étapes décrites ci-dessus sont à nouveau appliquées, non plus sur l'image d'origine, mais sur l'image LL. Pour obtenir une compression permettant une reconstruction de qualité, il est nécessaire de jouer sur le nombre de décompositions, mais aussi sur le degré et la nature des filtres utilisés.

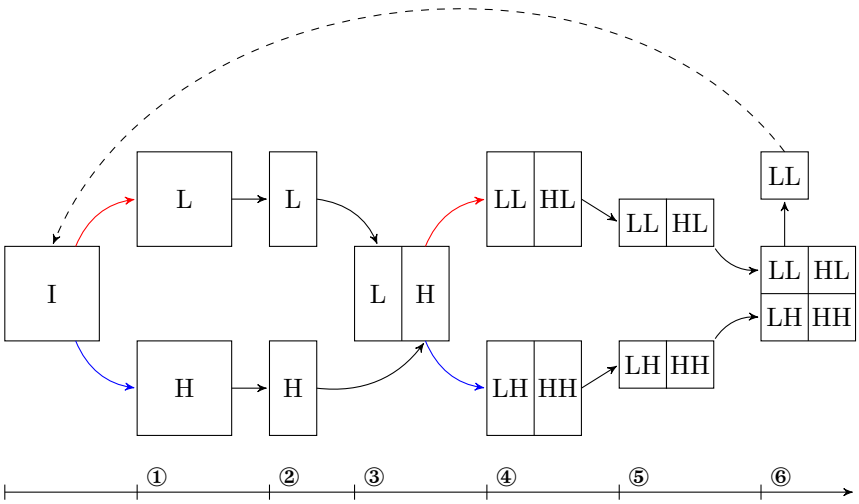


FIGURE 3.4 – Processus de décomposition d'une image en ondelettes grâce à la DWT.

Les standards JPEG et MPEG utilisent à la fois la DCT et la DWT. L'utilisation de la DCT a tendance à faire apparaître des artéfacts à la frontière des blocs, ce qui n'est pas le cas de la DWT qui n'utilise pas de blocs, mais l'image dans son intégralité. Dans la plupart des cas et face à la DCT, la DWT obtient comparativement de meilleurs résultats.

3.1.4 Quantification

Le processus de quantification permet de passer d'un ensemble X à un ensemble Y, de taille plus petite. Dans les standards MPEG, deux types de quantifications sont utilisés : la quantification scalaire et la quantification vectorielle. La première est la plus simple et permet de faire correspondre à une valeur en entrée, une valeur de sortie (dans un ensemble Y de dimension 1 et de taille 1).

La figure 3.5 présente un quantifieur scalaire uniforme (les intervalles sont de longueur constante, le pas de quantification est donc fixe et les niveaux de reconstructions sont uniformément répartis) et un quantifieur à zone morte (c'est à dire non uniforme). L'opération entraîne une compression avec pertes qui va donc permettre de réduire le nombre de bits nécessaires pour coder l'information. Elle est indépendante du type de transformée utilisée au préalable. D'une manière classique, elle est employée pour supprimer les valeurs nulles ou insignifiantes obtenues après application d'une transformée. Quant à la quantification vectorielle, elle permet de coder des valeurs d'un espace vectoriel multidimensionnel en les remplaçant par des valeurs d'un sous-espace discret de plus petite dimension, il s'agit d'ailleurs d'un système de compression à part entière (voir 3.3.1).

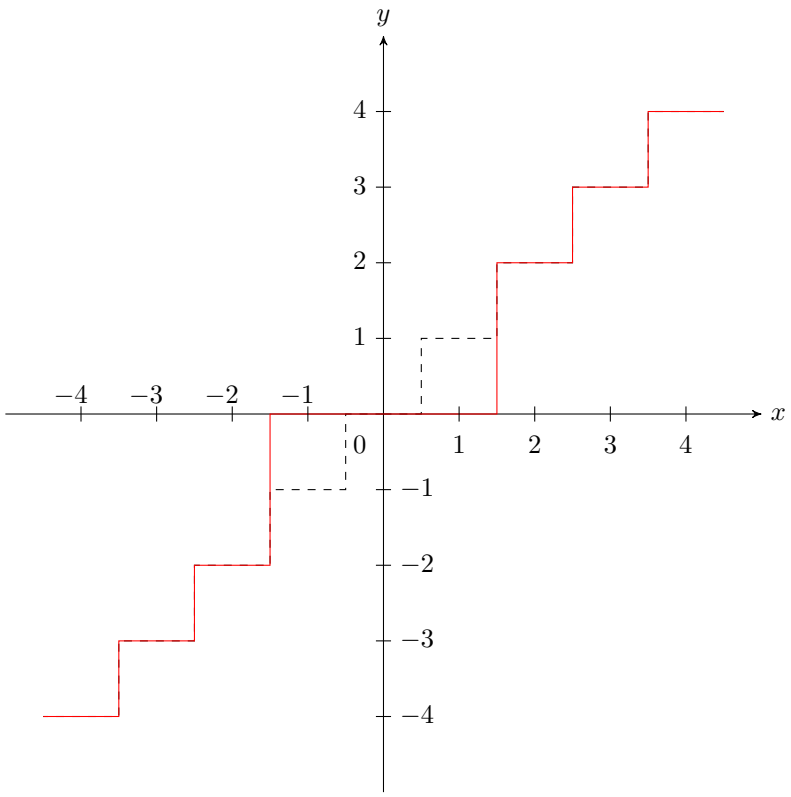


FIGURE 3.5 – Quantification uniforme en noir (pointillés), non uniforme en rouge (trait plein).

3.1.5 Codage entropique

Le codage entropique est une technique de compression sans perte, indépendante du médium utilisé comme support de l'information. Un code (une suite de bits) est attribué à chaque symbole (caractère) arrivant en entrée. L'encodeur compresse les données en remplaçant chaque symbole de taille fixe, par un code de taille variable. La taille de ce code est pratiquement proportionnelle au loga-

rithme de l'inverse de la probabilité. C'est pourquoi les symboles les plus utilisés ont les codes les plus courts. La taille optimale L_s d'un symbole est donnée par le théorème de Shannon :

$$L_s = \log_2 \left(\frac{1}{P_s} \right) \tag{3.5}$$

Son entropie E est alors calculée de la façon suivante :

$$E = \sum_s P_s \log_2 \left(\frac{1}{P_s} \right) \tag{3.6}$$

Les deux codeurs entropiques les plus utilisés sont celui de Huffman et le codeur arithmétique. Ce dernier apporte une alternative au codage de Huffman tout en s'approchant encore plus du taux de compression maximal théorique. Il convertit une séquence de symboles en un seul nombre fractionnaire.

3.1.6 Standards existants

Les paragraphes suivants présentent les deux compressions avec pertes les plus utilisées.

JPEG

Il s'agit d'un standard (depuis 1992) pour la compression d'images, conjointement développé par l'ITU et l'ISO [14]. Officiellement JPEG est le standard ISO/IEC [15] $10918 - 1$, [84] également connu comme la recommandation T.81 pour l'ITU. Cette norme est basée sur l'utilisation de la DCT. Le travail pour l'améliorer se poursuit, et elle pourrait notamment adopter des algorithmes basés sur le DWT.

JPEG2000

JPEG2000 [103] est un nouveau système de codage des images développé par le JPEG. Cette norme est basée sur l'utilisation de la DWT, car cette transformée assure une bonne compacité, mais aussi des possibilités de mises à l'échelle dans le domaine spatial et en qualité. Ce standard a pour but de proposer plus de flexibilité, et notamment un accès à l'image dans le domaine compressé. Ainsi, une image peut être manipulée ou éditée directement depuis sa forme compressée.

3.2 Primitives géométriques et profils de couleur

Méthode alternative de stockage des images, notamment utilisée pour ses caractéristiques permettant de ne pas dégrader le rendu, quelle que soit la résolution spatiale de sortie.

14. *International Standard Organisation*
15. *International Electrotechnical Commission*

3.2.1 Principes de base

Représenter une image de façon vectorielle est aussi possible. Dans ce cas il s'agit d'utiliser ce qu'il convient d'appeler des primitives graphiques, définies et représentées de façon mathématique. Certaines sont simples : le point, la ligne, le polygone. D'autres utilisent des notions plus complexes comme les courbes de Bézier, les spline ou encore les NURBS [16] par exemple. Une image vectorielle est donc constituée d'un ensemble fini de primitives graphiques, qui constitue une géométrie. Chacune est localisée à l'aide d'un repère et possède divers attributs, comme sa couleur, sa forme, ou son épaisseur. Le squelette défini par l'ensemble des primitives graphiques constituant une image vectorielle sert de support à la couleur. Il est ensuite nécessaire de déterminer comment le remplissage des couleurs sera réalisé entre les primitives graphiques. Celui-ci peut par exemple être uni, utiliser un gradient de couleur ou d'autres types de modèles.

Du fait de sa nature, une représentation vectorielle est très facilement modifiable et donc par extension facile à animer. Ce type de représentation est de plus en plus utilisé, par exemple dans l'interface graphique des systèmes d'exploitation (icônes de bureau sous Microsoft Windows 7 par exemple) ou encore pour les sites Web.

Il faut bien noter que la quasi-totalité des représentations vectorielles actuellement utilisées n'a pas été générée automatiquement, mais est le résultat du travail d'artistes ou d'infographistes. De plus, les images utilisant ce type de représentation sont généralement plutôt simples : formes peu complexes, peu de détail (en somme, comme dans une icône ou un logo).

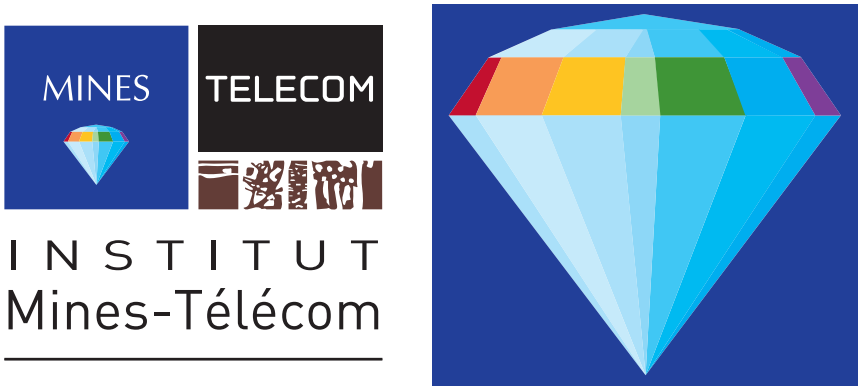
Néanmoins, depuis quelques années, des logiciels de dessins tels qu'Adobe Illustrator ou Corel CorelDraw proposent des outils permettant de générer ce type d'image plus facilement, en utilisant notamment des outils basés sur un maillage de gradients (*gradient mesh* en anglais). Cet outil peut en outre être utilisé pour créer manuellement des images ayant un rendu photo réaliste et peut représenter une grande variété d'images tout en restant simple à utiliser. Au moment de l'affichage à l'écran, la valeur de chaque pixel est calculée à l'aide des primitives graphiques le définissant. On parle alors de rendu. Là encore, un grand nombre de formats existent ; le plus connu d'entre eux est certainement SVG [17], très présent sur Internet. Sa version compressée (sans perte) SVGZ [18] est aussi largement utilisée.

La figure 3.6 met en avant l'avantage principal d'une représentation vectorielle : la qualité d'affichage est maximale, peu importe le facteur d'agrandissement par rapport à l'image de base. La version numérique de ce document permet d'ailleurs au lecteur de vérifier cette affirmation très simplement.

3.2.2 Primitives graphiques et interpolation

Une interpolation est une opération mathématique permettant de modéliser au mieux un ensemble fini de points grâce à une définition formelle. Dans le cas d'une image à vectoriser, cet ensemble de points peut par exemple représenter un contour. L'interpolation consiste alors à déterminer certains paramètres du modèle, pour permettre une approximation des données la plus fine possible.

16. *Non Uniform Rational B-Splines*
17. *Scalable Vector Graphics*
18. Format SVG compressé avec le standard gzip.



(a) Taille réelle. (b) Agrandissement d'une partie de l'image.

FIGURE 3.6 – Images représentées sous la forme vectorielle. Leur représentation matricielle équivalente est donnée dans la figure 3.1.

L'interpolation la plus simple est l'interpolation linéaire qui consiste simplement à joindre les points à l'aide de segments. D'autres types d'interpolation existent, basés sur l'utilisation de fonctions spécifiques comme la fonction sinus, l'utilisation de polynômes voire d'objets plus complexes comme les spline. Pour que la courbe interpolée modélise au mieux les données expérimentales, la méthode des moindres carrés est généralement employée même si d'autres méthodes sont possibles (interpolation d'Hermite par exemple). Les primitives graphiques dont on parle ici sont des objets mathématiques définis de façon formelle, ce sont des représentations implicites. Ces outils possèdent une définition qui leur est propre et permettent de représenter des courbes, mais aussi des surfaces de façon plus ou moins précise. Comme dans tout problème d'optimisation, le but est ici d'utiliser un minimum de place (en nombre de bits ou d'octets) pour stocker des éléments qui permettront une reconstruction la plus fidèle possible. Les paragraphes suivants détaillent les primitives graphiques utilisées dans la représentation vectorielle d'images, et plus généralement dans tous les domaines ayant recours à la modélisation.

Les courbes polynomiales

Un échantillon de points peut dans certains cas être représenté par un polynôme unique. Un polynôme P de degré n et a une indeterminée x est défini comme une expression de la forme :

$$P(x) = a_0 + a_1x^1 + a_2x^2 + \cdots + a_nx^n \tag{3.7}$$

Avec a_i des coefficients réels, non nuls.

Les spline

Dans le paragraphe précédent, un seul polynôme était en mesure de modéliser l'ensemble des données. Néanmoins, dans la plupart des cas, plusieurs polynômes sont nécessaires pour obtenir une interpolation de qualité. L'utilisation

de spline est alors nécessaire. Une courbe spline est une fonction polynomiale par morceaux définie sur un intervalle $[a, b]$ divisé en sous intervalles $[t_{i-1}, t_i]$ tels que : $a = t_0 < t_1 < \cdots < t_{k-1} < t_k = b$. On la note donc $S : [a, b] \to \mathbb{R}$.

Sur chaque intervalle $[t_{i-1}, t_i]$ on définit un polynôme : $P_i : [t_{i-1}, t_i] \to \mathbb{R}$. Ce qui donne pour une spline à k intervalles :

$$
\begin{aligned}
&S(t) = P_1(t) \ , \ t_0 \leq t < t_1 \, ; \\
&S(t) = P_2(t) \ , \ t_1 \leq t < t_2 \, ; \\
&\ \vdots \\
&S(t) = P_k(t) \ , \ t_{k-1} \leq t \leq t_k.
\end{aligned}
$$

La figure 3.7 montre une spline composée de deux segments. Le premier d'entre eux est défini par le polynôme $P_1 = 2(x + 1) - 1$ sur l'intervalle $[-2, 0]$, le second par $P_2 = 1 - 2(x - 1)$ sur l'intervalle $[0, 2]$.

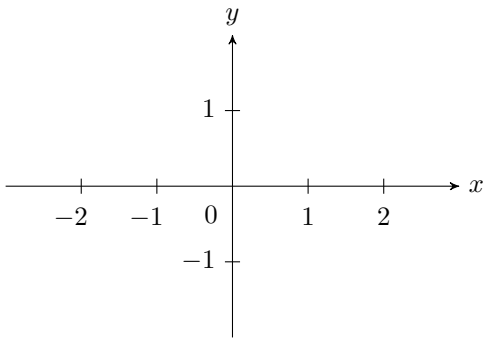


FIGURE 3.7 – Exemple d'une spline de degré 2 sur deux segments. Les deux couleurs distinguent les deux polynômes utilisés.

Le degré de la spline est défini comme étant celui du polynôme P_i de plus haut degré. Si tous les polynômes possèdent le même degré, on dit que la spline est uniforme. Dans le cas contraire, elle est non uniforme. La continuité d'une spline dépend de la continuité au niveau de la jointure des différents polynômes puisque la fonction polynôme est continue. Si pour tout i tel que $0 < i < k$ et pour tout j (dérivées) tel que $0 \leq j \leq n$ l'égalité suivante est vérifiée : $P_i^{(j)}(t_i) = P_{i+1}^{(j)}(t_i)$, alors la spline est de continuité n, notée C_n. Dans l'exemple présenté dans la figure 3.7, un seul problème de continuité est présent, lorsque $x = 0$. La fonction spline la plus simple est de degré 1, c'est-à-dire qu'il s'agit d'une ligne polygonale (une ligne brisée donc). Habituellement la majorité des outils faisant appel aux spline utilisent le degré 3.

Le problème majeur des spline est leur dépendance à un repère, ce qui rend leur utilisation impossible dans les logiciels de CAO [19].

19. *Computer-Aided Design*

Les courbes de Bézier

Contrairement aux spline, les courbes de Bézier [7] sont définies entièrement géométriquement. En d'autres termes, aucun repère n'intervient puisque la construction en est indépendante. À partir de $n+1$ points de contrôle $\mathbf{P}_0, \ldots, \mathbf{P}_n$, la courbe de Bézier $\mathbf{c}(t)$ est définie selon équation 3.8.

$$\mathbf{c}(t) = \sum_{i=0}^{n} \mathbf{P}_i B_i^n(t) \tag{3.8}$$

Où B_i^n représente un polynôme de Bernstein et $t \in [0,1]$. La figure 3.8 présente un exemple d'une courbe de Bézier cubique, qui comporte donc quatre points de contrôle.

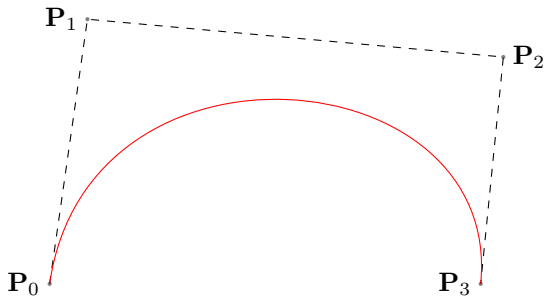


FIGURE 3.8 – Courbe de Bézier cubique avec son polygone de contrôle.

Les courbes B-spline

Une courbe B-spline est une spline conforme aux principes de Bézier. Sa construction est donc géométrique et aucun repère n'est nécessaire. Une courbe B-spline est une combinaison linéaire de B-splines, les B-spline étant la généralisation des courbes de Bézier. La figure 3.9 donne un exemple d'une telle courbe. Une courbe B-spline $\mathbf{c}(t)$ est définie par la fonction suivante :

$$\mathbf{c}(t) = \sum_{i=1}^{n} \mathbf{P}_i B_{i,k,\mathbf{t}}(t) \tag{3.9}$$

La dimension de la courbe $\mathbf{c}$ est égale à celle de ses points de contrôle (ou polygone de contrôle) $\mathbf{P}_i$. En règle générale, la dimension d'une courbe est au moins de 3, mais peut bien évidemment être supérieure. Ainsi, une courbe B-spline est une combinaison linéaire d'une séquence de B-splines $B_{i,k,\mathbf{t}}$ (appelées *B-basis*), uniquement déterminée par un vecteur nodal (*knot vector* en anglais) $\mathbf{t}$ et d'ordre k. L'ordre est équivalent au degré des polynômes, plus un. Les B-splines cubiques ont donc un ordre de 4 et un degré de 3. La variation possible du paramètre t d'une courbe B-spline $\mathbf{c}$ est comprise dans l'intervalle $[t_k, t_{n+1}]$ la courbe est donc une projection de $\mathbf{c} : [t_k, t_{n+1}] \to \mathbb{R}^d$ où d est un espace euclidien de même dimension que celle de ses points de contrôle.

La description complète d'une courbe B-spline est donnée ci-dessous :

dim : la dimension de l'espace euclidien, $1, 2, 3 \ldots$;

n : le nombre de sommets (ou le nombre de B-splines) ;

k : l'ordre des B-splines ;

$\mathbf{t}$: le vecteur nodal des B-splines. $\mathbf{t} = (t_1, t_2, \ldots, t_{n+k})$;

$\mathbf{P}$: les points de contrôle de la courbe B-spline $p_{d,i}$ avec $d = 1, \ldots, dim$ et $i = 1, \ldots, n$.

Les données utilisées dans la représentation doivent en outre respecter certaines conditions :

a. Les composantes du vecteur nodal doivent être croissantes $t_i \leq t_{i+1}$, de plus, deux nœuds t_i et t_{i+k} doivent être distincts : $t_i < t_{i+k}$;
b. Le nombre de sommets doit être supérieur ou égal à l'ordre de la courbe : $n \geq k$;
c. Il doit y avoir k nœuds égaux au début et à la fin du vecteur nodal ; le vecteur nodal $\mathbf{t}$ doit donc satisfaire les conditions $t_1 = t_2 = \ldots = t_k$ et $t_{n+1} = t_{n+2} = \ldots = t_{n+k}$.

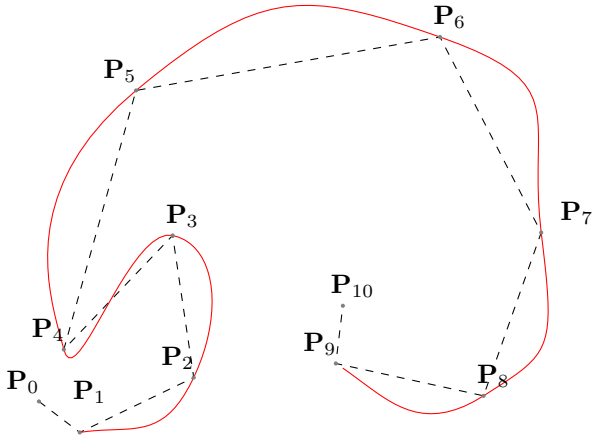


FIGURE 3.9 – Une courbe B-spline avec son polygone de contrôle.

Les B-splines ou *B-basis* Un ensemble de B-splines est déterminé par l'ordre k et par des nœuds. Par exemple, pour définir une simple courbe B-spline de degré 1, il faut 3 nœuds. Une B-spline quadratique est une combinaison linéaire de deux B-splines linéaires, elle est définie par quatre nœuds. Une courbe B-spline d'ordre k est la somme de deux B-spline d'ordre $k-1$, chacune pondérée par un facteur dans l'intervalle $[0, 1]$. Les B-splines d'ordre 1 sont définies de la façon suivante :

$$b_{i,1}(t) = \begin{cases} 1 & \text{si} \quad t_i \leqslant t < t_{i+1} \\ 0 & \text{sinon} \end{cases}$$

La définition complète d'une courbe B-splines d'ordre k est la suivante :

$$b_{i,k}(t) = \frac{t - t_i}{t_{i+k} - 1 - t_i} b_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} b_{i-1,k-1}(t)$$

Le polygone de contrôle Les points de contrôle $\mathbf{P}_i$ en définissent les sommets. Le polygone de contrôle d'une courbe B-spline est l'arc polygonal formé par ses points de contrôle $\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_n$. Si l'on augmente l'ordre, la distance entre le polygone de contrôle et la courbe augmente.

Le vecteur nodal Les nœuds d'une courbe B-spline décrivent les propriétés suivantes de la courbe :

a. La paramétrisation de la courbe B-spline ;
b. La continuité à l'extrémité de chaque intervalle entre les polynômes adjacents.

Deux courbes B-spline possédant le même polygone de contrôle, mais des vecteurs nodaux différents ne sont donc pas identiques.

Les courbes NURBS

Une courbe NURBS est la généralisation d'une courbe B-spline. Leur définition peut être donnée de la manière suivante :

$$\mathbf{c}(t) = \frac{\sum_{i=1}^{n} w_i \mathbf{P}_i B_{i,k,\mathbf{t}}(t)}{\sum_{i=1}^{n} w_i B_{i,k,\mathbf{t}}(t)} \tag{3.10}$$

En plus des données d'une courbe B-spline, une courbe NURBS $\mathbf{c}$ possède une séquence de poids $w_1, \ldots, w_n$. La représentation d'une courbe NURBS est la même que celle d'une B-spline, en y ajoutant les points suivants :

$\mathbf{w}$: une séquence de poids : $\mathbf{w} = (w_1, w_2, \ldots, w_n)$.

Les surfaces B-spline

Une surface B-spline est définie de la façon suivante :

$$\mathbf{s}(u, v) = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \mathbf{P}_{i,j} B_{i,k_1,\mathbf{u}}(u) B_{j,k_2,\mathbf{v}}(v) \tag{3.11}$$

$\mathbf{p}_{i,j}$ étant les points de contrôles, et u et v les paramètres dans les deux directions. L'équation 3.11 montre qu'une surface B-spline est le produit de deux courbes B-spline. Pour représenter une surface B-spline, les éléments suivants sont nécessaires :

dim : la dimension de l'espace euclidien ;

n_1 : le nombre de sommets pour le premier paramètre ;

n_1 : le nombre de sommets pour le second paramètre ;

k_1 : l'ordre des B-spline pour le premier paramètre ;

k_2 : l'ordre des B-spline pour le second paramètre ;

$\mathbf{u}$: le vecteur nodal des B-spline pour le premier paramètre, $\mathbf{u} = (u_1, u_2, \ldots, u_{n_1+k_1})$;

$\mathbf{v}$: le vecteur nodal des B-spline pour le second paramètre, $\mathbf{v} = (v_1, v_2, \ldots, v_{n_2+k_2})$;

$\mathbf{P}$: les points de contrôle de la surface B-spline, $c_{d,i,j}$, $d = 1, \ldots, dim$, $i = 1, \ldots, n_1$, $j = 1, \ldots, n_2$.

Les données utilisées dans la représentation doivent en outre respecter certaines conditions :

a. Les deux vecteurs nodaux doivent être croissants ;
b. Le nombre de sommets doit être supérieur ou égal à l'ordre en veillant à respecter les deux paramètres suivants : $n_1 \geq k_1$ et $n_2 \geq k_2$;
c. Il doit y avoir k_1 nœuds égaux au début et à la fin du vecteur nodal $\mathbf{u}$ et k_2 nœuds égaux au début du vecteur nodal $\mathbf{v}$.

Un exemple d'une telle surface est présent dans la figure 3.10

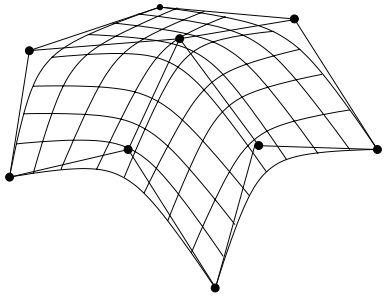


FIGURE 3.10 – Surface définie grâce à l'utilisation de courbes B-spline.

Les surfaces NURBS

Comme dans le cas des courbes, une surface NURBS est la généralisation d'une surface B-spline et est définie de la manière suivante :

$$\mathbf{s}(u,v) = \frac{\sum_{i=1}^{n_1}\sum_{j=1}^{n_2} w_{i,j}\mathbf{P}_{i,j}B_{i,k_1,\mathbf{u}}(u)B_{j,k_2,\mathbf{v}}(v)}{\sum_{i=1}^{n_1}\sum_{j=1}^{n_2} w_{i,j}B_{i,k_1,\mathbf{u}}(u)B_{j,k_2,\mathbf{v}}(v)} \tag{3.12}$$

La représentation d'une surface NURBS est la même que celle d'une B-spline, en y ajoutant les le point suivant :

$\mathbf{w}$: les poids de la surface NURBS, $w_{i,j}$, $i = 1, \ldots, n_1$, $j = 1, \ldots, n_2$, donc $\mathbf{w} = (w_{1,1}, w_{2,1}, \ldots, w_{n_1,1}, \ldots, w_{1,2}, \ldots, w_{n_1,n_2})$.

Les T-spline

Les T-spline [135] sont une généralisation des surfaces B-spline non uniformes. Les améliorations concernent notamment la gestion des points de contrôle. L'utilisation des T-spline autorise ainsi des raffinements locaux ou au contraire la suppression de points de contrôle inutiles (ce qui n'est pas possible avec des surfaces B-spline ou des surfaces NURBS). Il est aussi possible de fusionner

différentes surfaces B-spline définies par des vecteurs nodaux différents en un unique modèle.

3.2.3 Solutions pour la vectorisation des images

Il existe très peu de logiciels permettant d'effectuer la vectorisation d'une image de façon automatique tout en permettant l'obtention d'un résultat photo réaliste et en couleur. Certains outils libres tels que Potrace [20] permettent bien la vectorisation d'une image, mais pas de façon photo réaliste. D'autres outils plus puissants, comme ceux intégrés notamment à Adobe Illustrator sont plus efficaces, mais ne sont pas facilement utilisables (intégrables) par des logiciels tiers. Vector Magic [21], développé par l'université de Stanford est spécialisé dans ce domaine, mais n'est malheureusement plus disponible en version libre. Si on se base sur le fonctionnement des différentes méthodes, on peut séparer les solutions en deux grandes familles. La première utilise un maillage fixé au départ, qui est par la suite affiné par l'utilisation de diverses techniques. La seconde se base sur analyse de l'image, et notamment sur la détection des contours complétée par une segmentation qui seront la base du processus de vectorisation. Le tableau 3.1 regroupe les solutions proposées dans la littérature. Pour chaque solution, la famille à laquelle elle appartient, ainsi que la disponibilité d'un logiciel (code source ou binaire) est précisée. Chaque solution fait l'objet d'un paragraphe la détaillant.

Table 3.1 – Résumé des solutions disponibles dans la littérature.

id	**Nom**	**Famille**	**Logiciel disponible**
0	solution de Armstrong [66]	contours	aucun
1	Vector Magic [51]	contours	produit commercial
2	VIM [150]	contours	binaires disponible
3	solution de Sun [140]	maillage	aucun
4	solution de Zhang [49]	contours	aucun
5	courbes de diffusion [29]	contours	code source disponible
6	ARDECO [112]	maillage	code source disponible

Solution de Armstrong

L'approche proposée par Armstrong [66] est basée sur l'utilisation de lignes et de surfaces B-spline. Les images ne sont pas segmentées automatiquement (utilisation de l'outil lasso magnétique), mais grâce à l'utilisateur. Les contours des différents objets constituant l'image sont approximés à l'aide de courbes B-spline. La couleur de chaque objet est ensuite représentée grâce à une surface B-spline se bornant au contour précédemment établi. Le temps nécessaire au processus est important, même sur de petites images. La figure 3.11 présente le résultat de la vectorisation d'une image naturelle. Aucune précision n'est donnée quant à la taille des représentations ainsi générées.

20. `http://potrace.sourceforge.net`
21. `http://vectormagic.com`

(a) Image d'origine.

(b) Vectorisation avec 3650 points de contrôle.

FIGURE 3.11 – Vectorisation d'une partie de l'image selon la méthode proposée par Armstrong. Image extraite de [66].

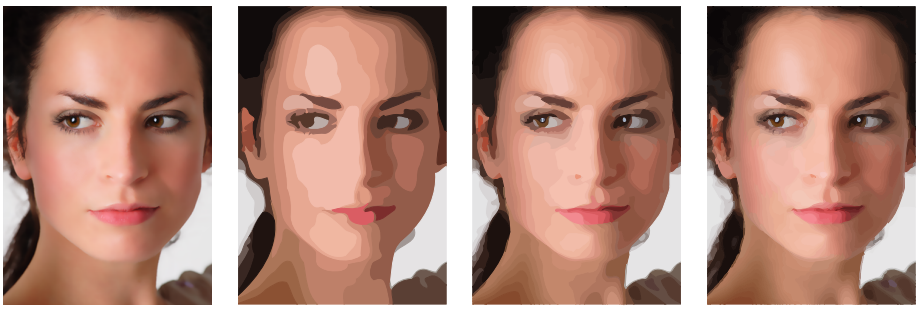
Vector Magic

Le logiciel Vector Magic permet la vectorisation de n'importe quelle image, peu importe qu'il s'agisse d'un logo, d'un dessin ou encore d'une photographie. Il s'agit d'un ancien projet de l'université de Stanford qui était il y a quelques années encore libre de droits, ce qui n'est malheureusement plus le cas aujourd'hui. Le travail de [51] présente le fonctionnement algorithmique du programme. La figure 3.12 ci-dessous permet de se rendre compte du résultat qu'il est possible d'obtenir. Dans la deuxième image, on voit clairement le tracé des courbes de Bézier utilisées dans ce cas. Le résultat n'est pas photo réaliste, puisque les zones construites pas les courbes sont remplies avec des couleurs unies, et non avec des gradients de couleurs. Il est possible moyennant finance de réaliser des vectorisations en ligne, ou d'obtenir une version du logiciel à installer sur une machine. Les résultats sont plutôt bons pour les logos, mais le temps nécessaire à la vectorisation est important (de 5 secondes à plusieurs minutes).

Le logiciel permet un export des images traitées dans des formats standards, tels que SVG, ou EPS [22]. Le tableau 3.2 met en évidence les tailles obtenues avec les trois niveaux de détail disponibles. Le PSNR [23] est utilisé comme métrique pour évaluer la qualité des représentations vectorielles : plus la valeur obtenue est élevée, plus la qualité de l'image restituée est proche de l'original. Théoriquement, la valeur maximale n'est pas bornée, néanmoins lorsque celle-ci atteint 50 dB on peut considérer les deux images comme visuellement identiques.

La figure 3.13 permet de comparer la méthode de vectorisation proposée par Vector Magic par rapport aux compressions matricielles habituelles (JPEG et JPEG2000). Le graphique représente le PSNR en fonction de la taille du fichier de sortie. Les meilleures performances sont donc offertes par l'encodeur JPEG2000. Si l'encodage SVGZ s'approche en terme de taille des encodeurs matriciels, la qualité de restitution est inférieure d'environ 5 dB.

22. *Encapsulated PostScript*
23. *Peak Signal to Noise Ratio*



(a) Image d'origine. (b) Vectorisation avec peu de détail. (c) Vectorisation avec un niveau de détail moyen. (d) Vectorisation avec le niveau de détail maximal.

FIGURE 3.12 – Vectorisation d'une image (extraite de [140]) en utilisant divers degrés de détail, à l'aide de Vector Magic.

TABLE 3.2 – Taille (en octets) des fichiers obtenus après vectorisation avec Vector Magic.

		Format		**Qualité**
		SVG	SVGZ	PSNR
Nom de l'image	Image 3.12b	36051	12618	40, 29
	Image 3.12c	107202	34298	42, 36
	Image 3.12d	319441	95413	43, 92

MPEG-4 : VIM

Ce logiciel de vectorisation fait partie intégrante de la norme MPEG-4 [24] et a pour objet de représenter des images avec un très petit nombre de vecteurs et de couleurs. On trouve plus de détails dans [150] ainsi que dans de nombreux brevets [54, 56, 60–62]. Ce type de contenu est utilisable grâce au langage BIFS [25]. Ce langage de description représente une image sous la forme d'un arbre de scène qui peut contenir plusieurs nœuds de différentes natures [38]. La partie 11 de MPEG-4 [77] présente ce format en détail. VIM [26] représente une image en utilisant trois éléments de base :

1. Des courbes caractéristiques (*charactersitic curves*) : elles sont composées d'une courbe interpolée (Bézier, spline ou B-spline) et d'un profil de couleur. Un profil de couleur est associé à chaque segment de la courbe et peut être de deux types différents (un bord ou une arête) ;
2. De petites zones (*patches*), constituées de quelques pixels et de couleurs significativement différentes de leur entourage. Elles sont représentées par des ellipses et colorées avec des gradients de couleurs ;
3. Des zones de couleurs (*area color points*), représentant les changements de couleur dans le fond de l'image, situées entre les courbes caractéristiques.

24. Nouveau standard d'encodage vidéo du MPEG.
25. *Binary Imaging For Scene*
26. *Vector Imaging Model*

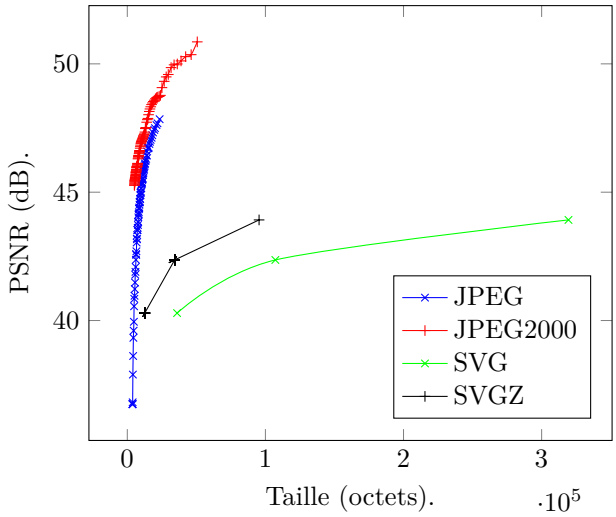


FIGURE 3.13 – Comparaison des performances offertes par Vector Magic par rapport à deux compressions matricielles de référence.

En plus de ces éléments de base, trois autres attributs sont utilisés pour animer l'image : des sous-textures, des squelettes et des objets. Bien que l'image rendue soit en deux dimensions, les informations permettant de la décrire sont en trois dimensions. Il existe un logiciel de référence, mais le code source n'est pas accessible. Seuls des binaires permettent l'encodage et le décodage d'une image. Il est à noter que l'interfaçage avec le langage de scène BIFS n'a pas été réalisé. Un paramètre permet de jouer sur la compression, et comme le montre la figure 3.14, le résultat est proche du photoréalisme. Selon l'exemple donné par [150], les résultats sont plutôt bons (le gain en taille entre l'image originale et sa version vectorisée est d'envion 50%, mais sans qu'aucune notion de qualité ne soit abordée). Il faut néanmoins nuancer ces propos puisque les comparaisons ont été réalisées avec l'encodeur JPEG, qui fait partie de la même génération que MPEG-4. Leurs performances respectives sont maintenant largement dépassées, notamment pour la vidéo grâce au MPEG-4 AVC [27]. De plus, aucune métrique de qualité n'est mise en avant pour comparer les solutions.

Solution de Sun

Dans [58, 140] la représentation vectorielle est créée semi-automatiquement, à l'aide d'un maillage sur lequel sont apposés des gradients de couleurs. Chaque gradient de couleur est composé de quatre ensembles de Ferguson (Ferguson *patches*). Le maillage est défini à l'aide d'une ou plusieurs courbes de Bézier cubique. L'optimisation du maillage réside dans la taille de la maille, qui est optimisée en fonction de la zone de l'image à représenter. Ainsi, plus il y a de détails, plus les mailles sont petites, et inversement pour les zones plus uniformes, les mailles sont plus grandes. Le maillage est donc raffiné en fonction du niveau de détail, grâce au calcul et à la minimisation d'une fonction d'énergie. Ceci permet de simplifier la représentation, et d'alléger le fichier final. Le temps

27. Partie 10 du standard MPEG-4, publié en 2004.

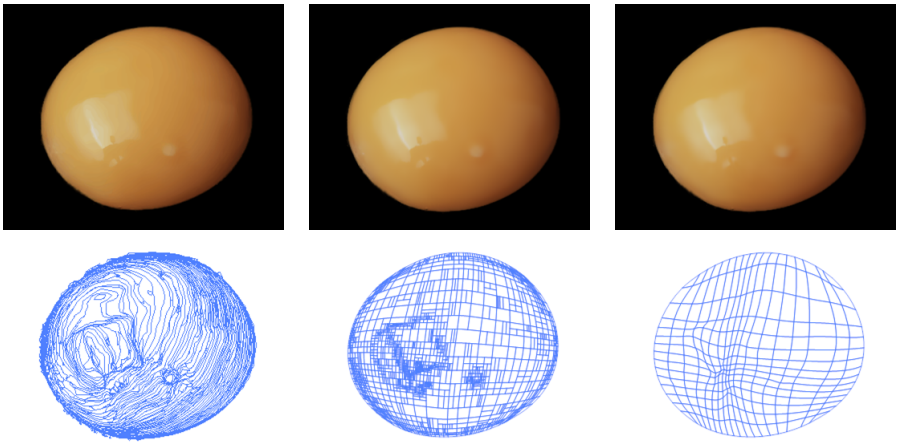
(a) Image originale.

(b) Vectorisation avec une compression des données minimale.

(c) Vectorisation avec une compression des données maximale.

FIGURE 3.14 – Vectorisation avec l'implémentation de référence de l'encodeur VIM.

de vectorisation est cependant extrêmement important – jusqu'à 12 minutes – et aucune information n'est donnée quant à la taille des fichiers produits. La figure 3.15 permet de comparer plusieurs solutions de vectorisation. On distingue bien les mailles de différentes tailles dans l'image 3.15c.



(a) Résultat avec Adobe Live Trace.

(b) Maillage par subdivision.

(c) Maillage de gradient optimisé.

FIGURE 3.15 – Comparaison de plusieurs types de maillage pour la vectorisation d'une image, figure extraite de [140].

Comme l'ensemble des méthodes de vectorisation, La méthode du *gradient mesh* permet une manipulation simple des images. Un utilisateur peut ajouter, supprimer ou modifier des points ou des lignes définissant le maillage. Pour chaque point du maillage, la courbure et la couleur peuvent facilement être modifiées.

Solution de Zhang

Cette approche se spécialise dans la représentation vectorielle de dessins animés [49]. Dans un premier temps, la détection et l'analyse des contours per-

mettent d'extraire des « lignes décoratives » ainsi qu'un masque. Les lignes décoratives constituent les contours importants des objets. Un algorithme de segmentation utilisant une sphère (*trapped-ball segmentation*) est ensuite mis en œuvre pour segmenter l'image en régions. Chaque région est ensuite vectorisée : le contour est formé d'une ou plusieurs courbes de Bézier cubique, la couleur remplissant les régions est représentée par une approximation polynomiale 3D (voir 3.2.2). Chaque région fait partie soit de l'arrière-plan, soit du premier plan. La décision est prise en fonction de la quantité de mouvement par rapport aux images voisines. Pour rendre la compression plus efficace, un même arrière-plan est envoyé pour toute une scène, les objets au premier plan étant ajoutés par dessus.

Dans les dessins animés, les scènes sont souvent simplifiées : il y a moins de détail et beaucoup de zones sont constituées d'un remplissage uniforme ou d'un dégradé de couleur. C'est une des raisons qui peut permettre à leur approche de vectorisation d'être intéressante. Néanmoins, cette tâche est loin d'être simple et peut aboutir dans certains cas à des résultats décevants. Cette solution est adaptée à la vidéo puisque les objets mobiles et les fonds de la scène sont reconstruits séparément. La représentation finale est donc reconstruite sous forme de couches, dans laquelle les régions et les lignes décoratives sont représentées sous forme vectorielle. Les auteurs présentent quelques résultats qui mettent en exergue un gain d'environ 50% en faveur de leur approche, face à l'encodeur Divx 6, basé sur MPEG-4. Comme dans le cas de l'encodeur VIM, les tests sont réalisés avec des encodeurs dépassés. On sait que dans la majorité des cas, MPEG-4 AVC produit des fichiers 50% plus légers [53]. Cette proportion est sans doute sous-évaluée si on l'applique à des dessins animés, qui comportent bien moins de détails qu'une vidéo normale.

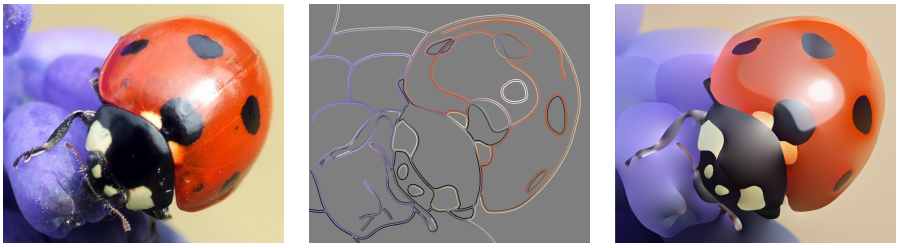
Courbes de diffusion

La méthode de vectorisation présentée dans [29] permet selon ses auteurs d'obtenir une version vectorielle, mais stylisée d'une image (c'est à dire non photo réaliste). L'obtention de la représentation vectorielle est soit manuelle (à l'aide d'un logiciel permettant de manipuler les courbes de diffusions), soit automatique. Cette dernière utilise une représentation multi échelle qui se base sur l'utilisation de plusieurs versions d'une même image à des degrés divers de flous. Plus l'image est floue, moins elle comporte de détail, ce qui permet de les classer par ordre d'importance. Ces détails sont détectés grâce à l'utilisation du détecteur de Canny [8]. Chaque chaîne de pixels représentant un contour est ensuite modélisée avec une courbe de Bézier, interpolée avec la méthode des moindres carrés. Une courbe de diffusion est ensuite définie à partir de la courbe de Bézier à laquelle différents attributs de couleur sont associés (les couleurs de l'image finale sont obtenues par diffusion et application d'un flou). La figure 3.16 montre l'application du processus complet sur une image naturelle.

ARDECO

Dans [112], une nouvelle solution portant le nom de ARDECO [28] est offerte. L'image vectorielle créée est composée de régions délimitées par des spline cubiques. Chaque région est remplie avec une couleur, selon un gradient linéaire

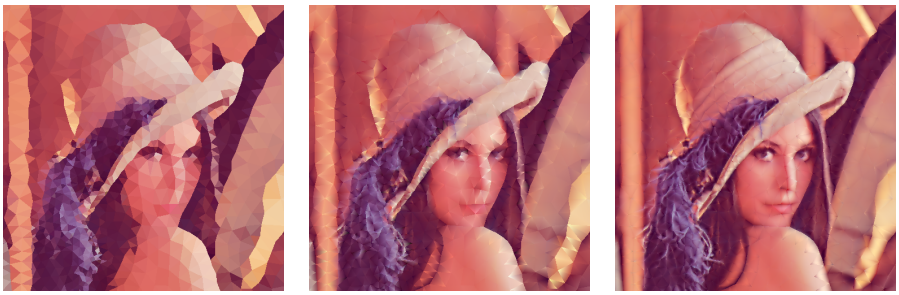
28. *Automatic Region DEtection and COnversion*



(a) Image de base. (b) Détail des courbes de diffusion utilisées. (c) Résultat final.

FIGURE 3.16 – Courbes de diffusion appliquées à une image naturelle, images extraites de [29].

ou circulaire. Le coeur de l'algorithme proposé repose sur une méthode de segmentation qui reconnait les gradients d'ordre important dans l'image originale. Une méthode permettant d'accélérer la convergence du processus est proposée et repose sur un nouveau concept, celui des trixels. Un trixel est une structure intermédiaire qui sépare l'image en triangles. Ainsi, dans les régions riches en détails, les trixels sont petits et leurs contours suivent les discontinuités de l'image. La méthode est lente (on compte en minutes) et le résultat n'est pas photo réaliste. La figure 3.17 expose le résultat de la vectorisation d'une photo en utilisant un nombre plus ou moins important de trixels.

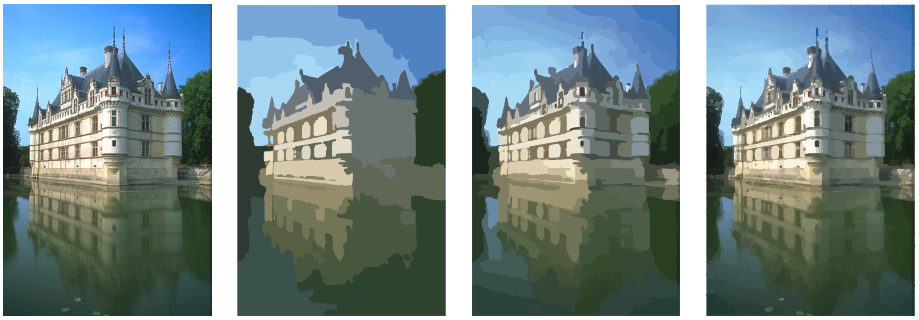


(a) Grands trixels. (b) Trixels moyens. (c) Petits trixels.

FIGURE 3.17 – Vectorisation de Lena avec un nombre variable de trixels, images extraites de [112].

Autres solutions

Les solutions de vectorisation présentées ci-dessus permettent d'obtenir une vue d'ensemble des différentes approches utilisées. Toutes les solutions existantes n'ont d'ailleurs pas été présentées, on peut notamment citer HSVGen [116] ou encore RaveGrid [141]. Comme on peut le voir à partir de la figure 3.18, les versions vectorisées obtenues avec la solution HSVGen sont similaires à celles obtenues avec Vector Magic (voir figure 3.12). Sur les images vectorielles produites par ces deux méthodes, on peut facilement distinguer les primitives graphiques et les profils couleur qui leur sont associés.



(a) Image d'origine. (b) Vectorisation avec peu de détail : 46 zones. (c) Vectorisation avec un niveau de détail moyen : 223 zones. (d) Vectorisation avec beaucoup de détails : 2424 zones.

FIGURE 3.18 – Vectorisation d'une image (extraite de [116]) en utilisant divers degrés de détail, à l'aide de la méthode HSVGen.

Quant à RaveGrid, son fonctionnement est le plus basique de toutes les solutions de vectorisation dont il a été question dans ce chapitre.

3.2.4 Analyse comparative des méthodes de vectorisation

Après voir passé en revue un certain nombre de solutions de vectorisation, cette section se propose de les comparer. Pour cela, deux tableaux ont été constitués. Le premier d'entre eux, le tableau 3.3 fait le point sur le type de primitives géométriques employées par les solutions décrites dans les paragraphes précédents.

TABLE 3.3 – Résumé des primitives géométriques utilisées par les diverses solutions de vectorisation (se référer au tableau 3.1 pour la signification du champ *id*).

		Primitives géométriques			
		Courbes polynomiales	Courbes de Bézier	Spline	B-spline
id	0				×
	1		×		×
	2	×		×	
	3		×		
	4	×	×		
	5	×			
	6		×		

Grâce au tableau 3.3 on constate qu'aucune solution de vectorisation n'utilise l'intégralité des primitives graphiques exploitables. La plupart d'entre elles se contentent d'en exploiter deux. Le tableau 3.4 s'attache quant à lui à une comparaison des performances tant du point de vue de la rapidité de traitement que de la qualité du résultat obtenu. Pour cela, une note comprise entre 1 et

5 est attribuée à chaque solution pour ces deux critères ; la valeur 5 étant la meilleure note.

TABLE 3.4 – Comparaison des performances des différentes méthodes de vectorisation (se référer au tableau 3.1 pour la signification du champ *id*).

		Critères de comparaison		
		Temps de traitement	Qualité du résultat	Format de sortie
id	0	2	2	N/A
	1	2	4	EPS, SVG, PNG, BMP
	2	3	5	fichier binaire non standard
	3	1	5	N/A
	4	3	3	N/A
	5	2	3	fichier XML non standard
	6	3	4	N/A

À partir des données du tableau 3.4, on remarque en premier lieu que seules deux méthodes permettent d'atteindre un résultat photo réaliste : la méthode de Sun et la méthode VIM. Bien que les valeurs affectées à la colonne représentant le temps de traitement évolue de 1 à 3, aucune des méthodes présentées dans ce document ne peut en l'état prétendre au temps réel. Du point de vue des formats de sortie, seul Vector Magic propose l'utilisation de standards stockant les informations sous forme vectorielle.

3.3 Autres méthodes pour la représentation des images

Comme l'introduction l'a explicité, les méthodes pixel et vecteur ne sont pas les seules solutions exploitables pour représenter une image ou plus généralement une texture. Les paragraphes suivants reprennent et détaillent les méthodes alternatives listées au début de ce chapitre.

3.3.1 Quantification vectorielle

La quantification vectorielle utilise une idée de base relativement simple : remplacer par une clé (ou un code) les valeurs d'un espace vectoriel multidimensionnel par des valeurs d'un sous-espace de dimension moindre. Le sous-espace peut contenir un nombre fini ou infini de valeurs. En imagerie, pour assurer un taux de compression acceptable, le sous-espace dispose d'un nombre de valeurs fini (sous-espace discret), celui-ci est alors qualifié de dictionnaire ou *codebook* en anglais. Il s'agit d'une compression avec perte. Un quantifieur Q sur N-niveaux est donc une projection d'un espace vectoriel K-dimensionnel $V = \{v_1, v_2, \ldots, v_K\}$ dans un sous-espace discret. L'ensemble des codes (ou clés) contenus dans le dictionnaire sont notés $W = \{w_1, w_2, \ldots, w_N\}$, avec $N < K$. En résumé, la projection s'écrit :

$$Q : V \to W \tag{3.13}$$

En d'autres termes, il s'agit d'assigner à un vecteur v en entrée, un vecteur w le représentant, celui-ci étant un code contenu dans le dictionnaire. Le quantifieur vectoriel Q est complètement décrit par le dictionnaire W, ainsi que par le fait que toutes les partitions $R = \{r_1, r_2, \ldots, r_N\}$ sont disjointes les unes des autres :

$$r_i = \{v : Q(v) = w_i\}$$

La première étape d'une quantification vectorielle est la formation de l'image. Les données de l'image sont tout d'abord partitionnées en un ensemble de vecteurs. Un nombre important de vecteurs, issus d'images diverses est alors utilisé pour former un ensemble d'apprentissages. Ces données sont utilisées pour générer un dictionnaire, la plupart du temps grâce à un algorithme de regroupement itératif. L'encodage d'une image est ensuite réalisé en cherchant pour chaque vecteur en entrée le code le plus proche dans le dictionnaire. L'index et non pas les valeurs quantifiées est transmis au décodeur, permettant d'atteindre un meilleur taux de compression. Au niveau du décodeur, l'index est décodé et converti en vecteur en utilisant le même dictionnaire. Les différentes implémentations d'un codage par quantification vectorielle peuvent être discriminées en prenant en compte les aspects suivants :

a. La création des vecteurs ;
b. La génération d'un ensemble pour l'apprentissage ;
c. La génération du dictionnaire ;
d. La quantification, pour chaque vecteur v, trouver le meilleur code w_i.

Il existe un nombre important d'algorithmes suivant ces quatre points clés, on peut notamment citer celui proposé dans [6].

3.3.2 Codage basé sur un modèle

Dans ce type d'approche, un modèle 2D ou 3D est tout d'abord construit. Lors de l'encodage, ce modèle est utilisé pour l'analyse de l'image à traiter. Une fois les paramètres du modèle extraits, ceux-ci (ainsi que le modèle) sont transmis au décodeur. La reconstruction s'effectue en utilisant le même modèle que lors de la compression, associé aux paramètres. Le concept de base est présenté au moyen de la figure 3.19. De ce fait, les techniques de base pour le codage basé sur un modèle sont la modélisation et l'analyse d'images, associés à des techniques de synthèse d'image, l'analyse et la synthèse étant basées sur le modèle de l'image. Les techniques de modélisation d'image utilisées pour ce type de codage peuvent être séparées en deux classes : la modélisation de la structure et celle du mouvement. La modélisation de la structure est notamment utilisée pour la reconstruction de scènes, que ce soit en 2D ou en 3D. La modélisation de mouvement est généralement utilisée pour les séquences vidéo, alors que celle de la structure l'est habituellement pour les images seules.

Le modèle géométrique est le plus souvent utilisé pour la description de la structure de l'image. Ce modèle peut être une description basée sur une surface ou sur un volume. L'avantage majeur de la description de surfaces est qu'il est facile de les convertir en une représentation qui peut être encodée et transmise. Dans ces modèles, la surface est approximée par un polygone

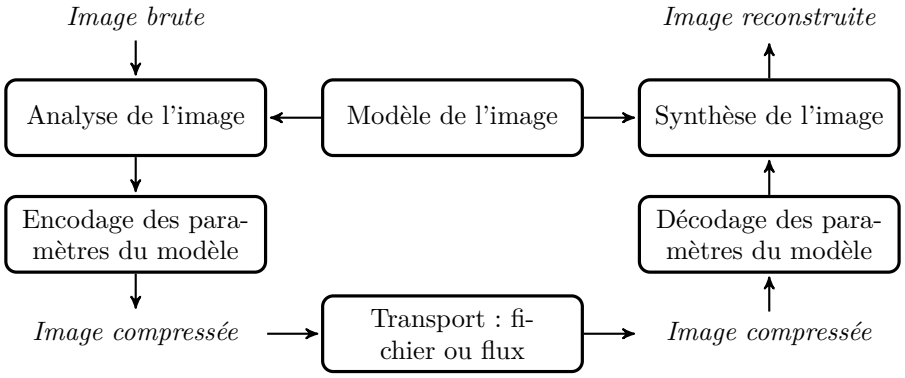


FIGURE 3.19 – Principes de base d'un codage basé sur l'utilisation d'un modèle.

formé de triangles. La forme de la surface est représentée par un ensemble de points, qui représentent les sommets des triangles constituant le maillage. La taille des triangles peut être variable, ajustée à la complexité de la surface à représenter. En d'autres termes, les zones complexes contiennent beaucoup plus de triangles que les zones plutôt lisses. La description basée sur la notion de volume est l'approche naturelle pour la modélisation de la plupart des objets. La plupart des recherches traitant de la description d'un volume se basent sur une approche paramétrique. Cependant, le codage basé sur des modèles est utilisable uniquement sur certains types d'images, étant donné qu'il est très difficile de trouver des modèles d'images permettant de représenter l'ensemble des scènes naturelles. Les quelques exemples ayant obtenu de bons résultats utilisent des modèles de représentation du visage, de la tête ou encore du corps humain dans son ensemble. Ces modèles sont développés pour l'analyse et la synthèse des images animées. Les outils permettant la représentation et l'animation d'un visage ou d'un corps [32] font partie de la norme MPEG-4 et plus précisément de la partie 16, nommée AFX [29] (voir 4.1.3). Le codage basé sur un modèle est particulièrement adapté pour les transmissions à bas débit, voire à très bas débit [21, 145].

3.3.3 Méthode fractale

Cette méthode de compression utilise la notion d'autosimilarité, c'est-à-dire la recherche de similarités à différentes échelles dans une image. Elle apporte de nouvelles idées pour réduire la redondance d'informations utilisant des corrélations à la fois locales et globales, en rupture avec les approches traditionnelles. Une fractale est une forme géométrique dont les détails peuvent être représentés par des objets à différentes échelles et selon divers angles. Elle peut être décrite par un ensemble de transformations comme des transformations affines. De plus, les objets utilisés pour représenter les détails de l'image ont une certaine forme d'autosimilarité et peuvent être employés pour représenter une image de manière récursive. Un exemple de fractale à partir de la courbe de Von Koch est donné par la figure 3.20 qui permet de dessiner un flocon de neige.

Le système de compression fractal a pour but d'approximer l'image avec

29. *Animation Framework eXtension*

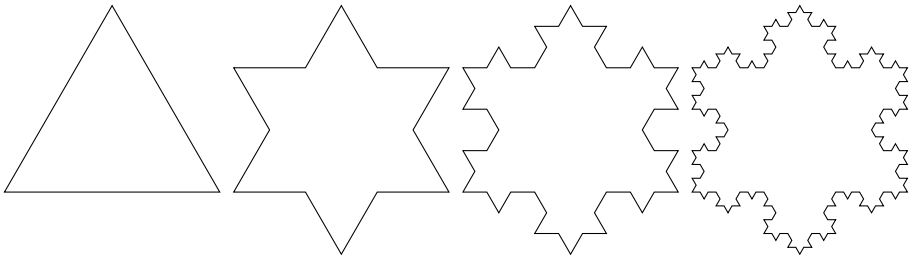


FIGURE 3.20 – Courbe de Von Koch appliquée aux trois côtés d'un triangle équilatéral (de gauche à droite : ordres $0, 1, 2$ et 3).

des fractales, le problème principal est donc la génération des fractales pour une image donnée. Pour cela, il est nécessaire de rappeler la notion de transformation affine contractante. L'équation 3.14 représente une transformation affine 2D sous sa forme matricielle.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = A \begin{bmatrix} x \\ y \end{bmatrix} = B \begin{bmatrix} x \\ y \end{bmatrix} + C = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} \tag{3.14}$$

La matrice B permet une rotation et/ou une mise à l'échelle, la matrice C une translation. Une transformation affine est dite contractive si la distance entre deux points P_1 et P_2 dans le nouvel espace E' est inférieure à cette même distance dans l'espace d'origine E, c'est-à-dire qu'il faut que l'équation 3.15 soit respectée, avec $s \in \mathbb{R}$ et $0 < s < 1$.

$$A(P_1) \quad A(P_2) < sP_1P_2 \tag{3.15}$$

De très nombreuses études ont été publiées sur la théorie des fractales. Diverses techniques utilisant une approche fractale ont été développées, comme la segmentation ou la synthèse d'images et plus généralement le traitement d'images. Les fractales sont particulièrement adaptées à la synthèse de phénomènes naturels (paysages, nuages...). On peut voir dans la courbe de Von Koch – de façon très schématique – la ligne de crête d'une montagne. En introduisant un peu d'aléatoire, cette courbe pourra prendre une forme moins régulière et donc plus naturelle, il s'agit alors d'une fractale non déterministe. Dans [9] les auteurs ont généralisé cette procédure à des espaces à N dimensions. Pour $N = 3$, il est possible de produire des paysages extrêmement variés, et pour $N = 4$ de les animer. Une démonstration des possibilités de synthèse de paysage est présentée dans la figure 3.21.

L'utilisation des fractales dans la compression n'est que relativement récente [18]. Les systèmes de compression à base fractale utilisent l'IFS [30] ; cette approche a été introduite en 1981 [68, 69]. Dans l'approche originale, cette transformation est composée de l'union d'un certain nombre d'applications affines contractantes, appliquées sur l'image entière. Bien que quelques exemples intéressants aient été générés par cette méthode, celle-ci n'était pas autonome (interventions manuelles nécessaires) et donc non utilisable telle quelle. L'algorithme initial était trop complexe et beaucoup trop lent. La compression fractale

30. *Iterated Function System*

(a) Synthèse fractale de montagnes avec de la végétation et des nuages d'orage.

(b) *Monument Valley* au lever du Soleil.

(c) L'anomalie de Botticelli sur la Lune.

FIGURE 3.21 – Paysages générés à l'aide de fractales non déterministes, données extraites de [9], images sous *copyright*.

devient une réalité avec l'introduction du PIFS [31] qui diffère du IFS dans le sens où chaque transformation utilise un sous-ensemble de l'image, et non pas l'image dans sa globalité. Le PIFS permet de vraiment réaliser une compression fractale de manière automatique. Même si les méthodes basées sur l'IFS sont les plus utilisées dans ce domaine, d'autres solutions de codage fractal existent. L'une d'entre elles exploite un schéma de compression basé sur une segmentation utilisant des dimensions fractales.

Le principe fondamental du codage fractal permet une latitude considérable dans la conception et la mise en oeuvre d'un tel système de compression. Les différences existantes entre les solutions de codage fractal peuvent être classifiées selon les critères suivants :

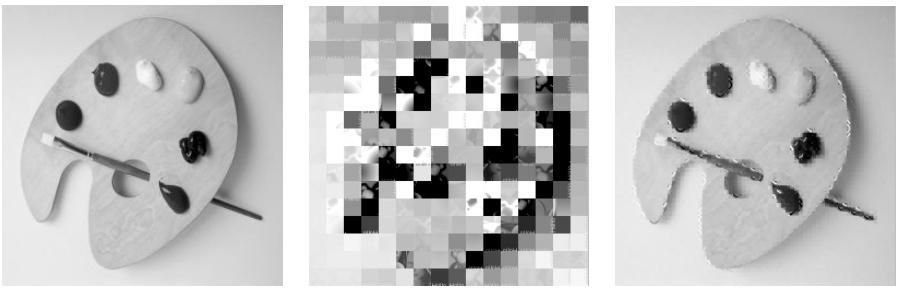
a. Partition imposée de l'image par les *range blocks* ;
b. Composition de l'ensemble des blocs du domaine ;
c. Classe des transformées appliquées sur les blocs du domaine ;
d. Type de recherche utilisé pour localiser des blocs de domaine appropriés ;
e. Représentation et quantification des paramètres des transformées.

Ces critères ne sont pas détaillés ici, mais une revue des différentes variantes de compression fractales est présentée dans [46]. Le domaine de la compression fractale a connu son apogée dans les années 90, néanmoins, des travaux récents sur la mise au point d'algorithmes rapides pour la représentation d'image sous forme fractale existent [149]. Un exemple de compression fractale est présenté dans la figure 3.22.

3.3.4 Méthodes spécifiques

Pour compléter cette analyse de l'état de l'art, rappelons les méthodes paramétriques [26, 148], spectrales [125], syntaxiques [15, 25, 127], structurelles [146] et stochastiques [15, 130]. Ces méthodes ne permettent pas de représenter des images quelconques. À titre d'exemple la figure 3.23 présente des images générées grâce à la méthode syntaxique.

31. *Partitioned Iterated Function System*

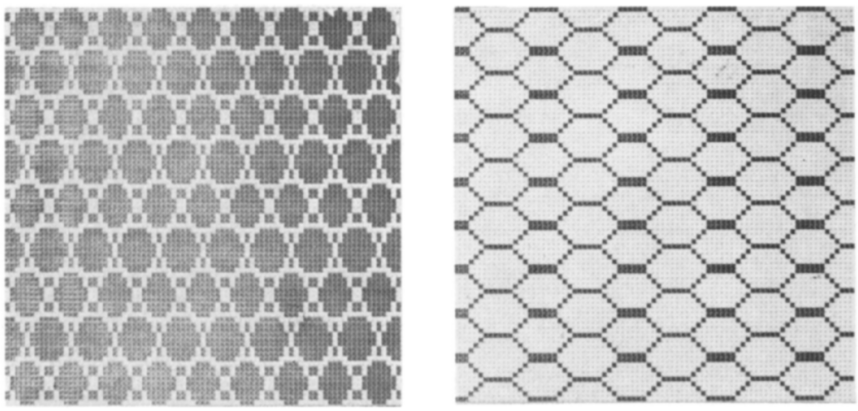


(a) Image d'origine.

(b) Image obtenue après une itération.

(c) Image après quatre itérations.

FIGURE 3.22 – Compression fractale appliquée sur une image en niveaux de gris (sur 8 bits).



(a) Texture basée sur des hexagones réguliers.

(b) Autre exemple d'image qu'il est possible de générer.

FIGURE 3.23 – Textures générées de manière syntaxique, c'est-à-dire à partir d'une grammaire, images extraites de [25].

3.4 Conclusion

Ce premier chapitre de l'état de l'art s'est focalisé sur les différentes méthodes et techniques permettant la représentation sous forme compressée ou non d'une image numérique. Certaines ne sont cependant pas adaptées à notre cas d'usage et seules les deux premières d'entre elles (respectivement les méthodes pixel et vecteur) permettent la représentation de n'importe quelle image. Deux modalités sont alors utilisables pour la représentation d'une image. La représentation sous forme de pixels étant couramment utilisée, une attention particulière a été portée aux solutions à base de primitives graphiques et de profils couleur. Certaines de ces méthodes proposent une qualité d'image équivalente (voire meilleure) à une représentation à base de pixels. L'analyse de l'état de l'art a permis de mettre en avant que les solutions de vectorisation existantes ne sont pas adaptées à notre besoin. Elles sont en effet bien trop complexes et in fine beaucoup trop lentes

pour être utilisées dans une chaîne de rendu distant. En outre, l'utilisation de primitives graphiques complexes a aussi une influence non négligeable sur la quantité de mémoire nécessaire. Pour atteindre notre objectif, il est donc nécessaire d'utiliser une méthode de vectorisation simplifiée, adaptée à notre cas d'usage. Pour cela, l'utilisation des primitives graphiques sera limitée à la notion de rectangle et les profils couleur associés seront représentés grâce à des polynômes.

Chapitre 4

Représentation et compression de séquences d'images

Sommaire

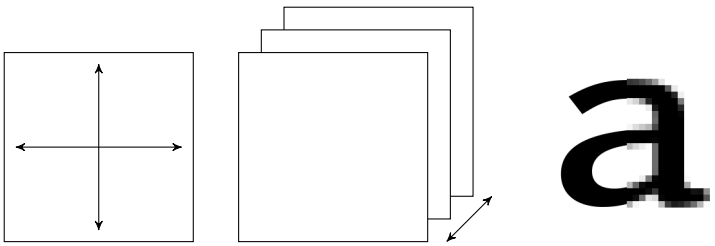
Extension de l'image, la vidéo est devenue capitale de nos jours. Elle nous permet de communiquer, de créer ou encore tout simplement de consommer du contenu vidéo. Un nombre toujours croissant d'applications utilisent quotidiennement la vidéo, on estime qu'à l'horizon 2015 plus de 90% de la bande passante d'Internet sera utilisée pour cet usage [75]. Que ce soit pour stocker des contenus ou les diffuser, peu importe le média physique utilisé (satellite, câble) il faut un système de codage efficace et sûr pour compresser les données. Même avec les moyens de communication actuels comme la fibre optique, il serait difficile de diffuser des flux vidéos non compressés. Un rapide calcul montre tout l'intérêt de la chose. Supposons que l'on veuille envoyer une vidéo en haute définition au standard 1080p [1], il faut donc une bande passante minimale de : $1920 \times 1080 \times 24 \times 3 = 149299200$ octets, soit environ 150 Mo/s (on multiplie le nombre de pixels, le nombre d'images par seconde, le nombre de canaux utilisés par le nombre d'octets représentant chaque information). On comprend alors toute la nécessité d'utiliser un système de compression des flux vidéos. Ainsi, il est possible de réduire la bande passante nécessaire et de diffuser plusieurs canaux en même temps sur un même médium. Contrairement aux images seules, pour lesquelles il existe un grand nombre de représentations possibles, la vidéo est – pour l'instant – uniquement représentée par des pixels. Comme pour les images, une compression avec perte est généralement mise en œuvre pour réduire la quantité d'informations à transmettre. À notre connaissance, un seul projet est en cours pour la réalisation d'un encodeur vidéo utilisant une représentation (entièrement) vectorielle lors des étapes de compression.

4.1 Méthode pixel

Au fil du temps, plusieurs systèmes de compression ont été développés et standardisés par différents groupes ou consortiums, tels que l'ITU, MPEG ou encore l'ISO. Ils sont capables de réduire fortement la bande passante nécessaire, tout en garantissant une qualité satisfaisante. Pour compresser efficacement ce type de média, il est nécessaire d'utiliser ses caractéristiques intrinsèques. On peut considérer une vidéo comme étant une suite d'images diffusées à un certain rythme. On considère alors deux caractéristiques principales : la redondance spatiale et la redondance temporelle, comme l'illustre la figure 4.1.

La redondance spatiale s'applique sur une seule image et vise à déterminer si des parties de l'image sont liées, on cherche alors un degré de corrélation ou de similitude. La redondance temporelle utilise le même principe, mais cette fois non pas dans la même image, mais dans des images successives. Le but est donc de minimiser un maximum les redondances spatiales et temporelles pour maximiser le taux de compression. Il est fondamental de comprendre qu'un système de compression utilise un codeur, mais aussi un décodeur, d'où l'acronyme CODEC [2] habituellement utilisé. Le codeur va utiliser des données brutes, non compressées pour produire un flux compressé qui sera soit diffusé en temps réel, soit enregistré sur un média numérique. Le décodeur exécute l'opération inverse : il reçoit un flux compressé et reconstruit la vidéo. Il faut bien noter que les vidéos originales et reconstruites ne sont pas identiques (la plupart des CODEC utilisent des compressions avec perte). Pour mesurer la différence, il

1. Résolution spatiale de 1920×1080 pixels.
2. *COde DECode*



(a) Redondance spatiale. (b) Redondance temporelle. (c) Gestion de la qualité.

FIGURE 4.1 – Redondances spatiale et temporelle utilisées dans la compression vidéo. La troisième illustration représente la qualité de compression (paramètre Qp, voir 4.1.1).

existe plusieurs moyens et critères objectifs, mais aussi subjectifs. Les métriques objectives les plus utilisées sont sans nul doute le PSNR et le MSSIM [3]. Côté subjectif, la notion de MOS [4] [117] est la plus utilisée.

4.1.1 Principes de base

Pour l'encodage d'une vidéo, celle-ci est divisée en un groupe d'images (GOP [5]), qui contient habituellement trois types distincts d'images nommées I, P et B. Un exemple de groupe d'images est présenté dans la figure 4.2.

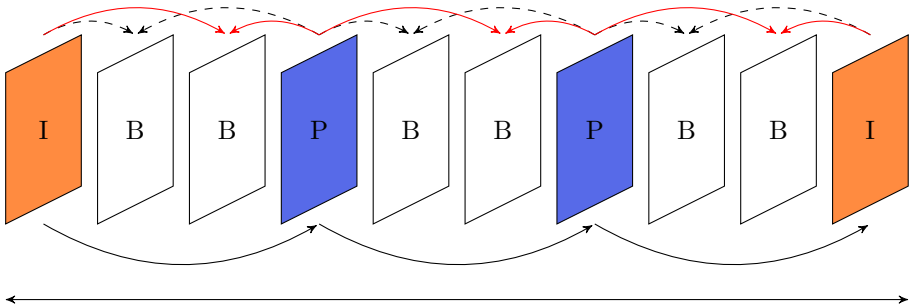


FIGURE 4.2 – Exemple de structure d'un groupe d'images dans un encodeur moderne.

Les images I ou images de référence (*Intra-coded*) ne font référence à aucune autre image, elles sont traitées indépendamment. Elles sont entièrement codées avec le mode « intra ». Elles sont dispersées tout au long du flux vidéo et permettent à l'utilisateur de commencer à lire une vidéo à n'importe quel moment. Un codage sans compensation de mouvement est utilisé, ce qui a pour conséquence un taux de compression modéré. Les images P (*Predictive-coded*) sont

3. *Multi-scale Structural Similarity*
4. *Mean Opinion Score*
5. *Groupe Of Pictures*

compressées en s'appuyant sur des images I et/ou P précédemment encodées. Ce type d'image permet une meilleure compression que les images I en vertu de la compensation de mouvement. Elles servent de référence pour les images P et B futures. Quant aux images B (*Bi-directional-coded*) elles sont codées grâce à l'utilisation d'une compensation de mouvement bidirectionnelle. La compression s'appuie à la fois sur des images I ou P déjà traités (passées), mais aussi futures. Les images B ne sont pas utilisées comme images de référence et ne propagent donc pas d'erreur. L'utilisation de ce troisième type d'image ajoute nécessairement de la latence lors de l'encodage vidéo. Il est possible d'utiliser le même schéma de groupe d'images tout au long du processus d'encodage. Néanmoins, pour améliorer le taux de compression, les encodeurs modernes n'utilisent pas un schéma fixe.

La plupart des encodeurs vidéo actuels comprennent quatre étapes : la prédiction, la transformation, la quantification et finalement un codage entropique. La figure 4.3 en présente le fonctionnement global.

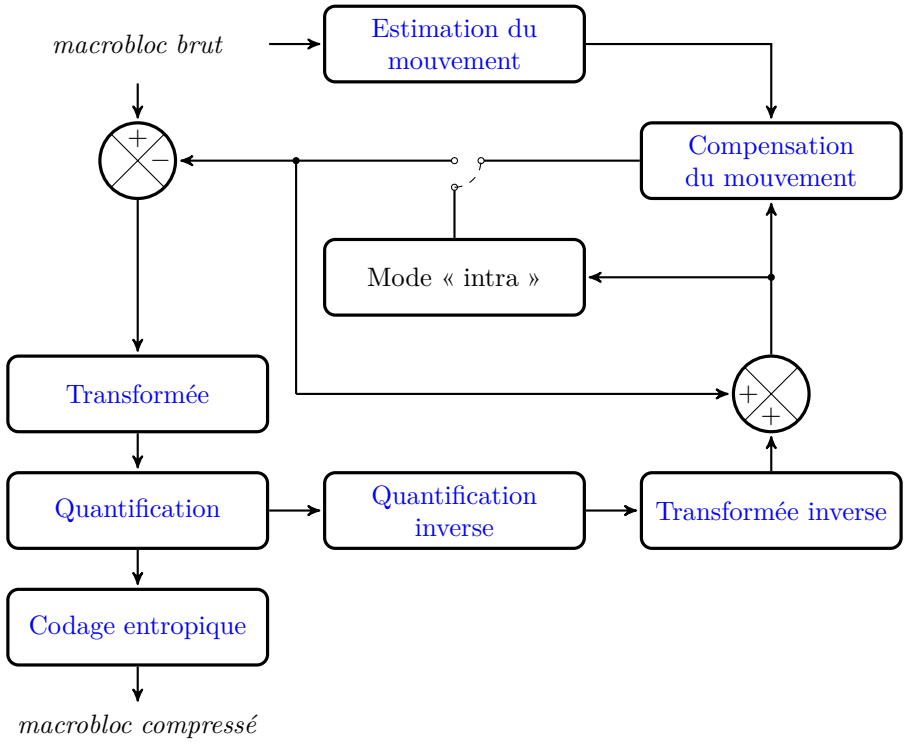


FIGURE 4.3 – Schéma de principe du fonctionnement d'un encodeur vidéo. En entrée du processus, le macrobloc est représenté dans le format YCrCb4:2:0 (voir 4.4).

Les étapes ayant trait à la transformation mathématique et à la quantification ayant déjà été traitées dans la section précédente (voir 3.1.3 et 3.1.4 respectivement), elles ne le seront pas à nouveau ici. Malgré l'utilisation de ces grandes étapes, des différences notoires existent dans les architectures des différents encodeurs. La première opération lors de la compression d'une vidéo est de subdiviser chaque image en macroblocs, qui ont généralement une taille de

16×16 pixels. Chaque macrobloc est alors encodé en utilisant soit le mode « intra » soit le mode « inter » en fonction de son appartenance à une image de type I, ou P et B. Dans le mode « intra », aucun vecteur de mouvement n'est calculé (l'étape de prédiction n'est pas utilisée) : on applique directement la DCT. Les coefficients obtenus sont alors quantifiés, réarrangés, puis enfin codés à l'aide d'un codeur entropique, qui utilise la technique éprouvée du VLC [6]. À ce moment-là, un mécanisme de contrôle peut adapter le facteur de quantification (noté Qp), dans le but de contrôler le débit binaire. Pour le mode de codage « inter », les choses sont quelque peu différentes. Un macrobloc est formé en utilisant une compensation de mouvement unidirectionnelle (images P) ou bidirectionnelle (images B) basée sur une estimation de mouvement. La prédiction pour chaque macrobloc est donc potentiellement basée sur une ou plusieurs images passées ou futures (d'un point de vue temporel), qui ont déjà été encodées et reconstruites. Au macrobloc prédit grâce à la compensation de mouvement, on soustrait alors le macrobloc courant qui va conduire à la formation d'un macrobloc contenant l'erreur résiduelle commise. C'est ce macrobloc qui va alors être encodé en utilisant la DCT. Les vecteurs de mouvements permettant la reconstruction sont ajoutés au flux. Ensuite, comme dans le cas « intra », les coefficients seront quantifiés, réordonnés, et codés de manière entropique en utilisant un codeur VLC.

Les coefficients résultant du codage entropique, ainsi que les informations nécessaires au décodage des macroblocs (tels que le mode prédiction, les vecteurs de mouvements ou encore le facteur de quantification) forment le flux vidéo compressé. Une IDCT et une quantification inverse sont utilisées au niveau du codeur pour les images de références maintenues en mémoire. Ces images décodées sont utilisées pour l'encodage des images futures. Au niveau du décodeur, le flux compressé est lu et découpé. Chaque élément est alors décodé (à l'aide d'un décodeur entropique), les coefficients sont réordonnés puis remis à l'échelle grâce à quantification inverse. L'application de l'IDCT permet de générer les macroblocs formant les erreurs résiduelles. À l'aide des images de références, le décodeur applique la prédiction calculée par le codeur. Le macrobloc prédit et le macrobloc correspondant contenant l'erreur de prédiction sont alors ajoutés l'un à l'autre.

Bien que dans certaines applications spécifiques, quelques éléments supplémentaires soient ajoutés, la structure de base d'un codec reste la même. On peut par exemple ajouter des filtres pour réduire le bruit numérique présent dans les images au niveau de l'encodeur. Le filtrage des images reconstruites au niveau du décodeur est aussi très utilisé, notamment pour tenter de s'affranchir des artéfacts produits par le découpage en blocs. Quoi qu'il en soit, ajouter ce type de fonctionnalités va inévitablement conduire à une augmentation de la complexité.

Dans le schéma de l'encodeur, l'étape principalement responsable de la compression est la quantification. Pour pouvoir contrôler le débit binaire, l'utilisateur peut faire varier le facteur de quantification, nommé Qp dans la plupart des implémentations. Plusieurs stratégies peuvent être appliquées pour le contrôle de ce paramètre tout au long du processus d'encodage, comme utiliser un Qp constant ou faire le choix d'un débit binaire constant (Qp n'est alors

6. *Variable-Length Code*

plus constant). Si la valeur du Qp est faible, la qualité de la vidéo sera bonne, généralement lorsque $Qp = 0$, l'encodeur utilise une compression sans perte. Il faut noter qu'une valeur de Qp élevée entraîne l'apparition d'artefacts visuels ; la figure 4.1c illustre ce phénomène.

4.1.2 Estimation, prédiction & compensation du mouvement

Les paragraphes précédents ont mis en avant les caractéristiques d'une vidéo, et plus précisément les leviers utilisables pour obtenir une compression. L'un d'entre eux, la redondance temporelle, est une technique utilisée par les encodeurs MPEG. Elle est basée sur une estimation du mouvement entre plusieurs images. L'idée de base est de dire que dans la plupart des cas, les images consécutives d'une vidéo sont similaires à l'exception des changements induits par les objets qui ont bougé. Si l'on prend pour exemple l'encodage successif de deux images identiques, il est facile pour l'encodeur de faire la bonne prédiction : l'image courante est codée comme un clone de l'image précédente. Lorsque cette observation est faite, les seules informations à transmettre au décodeur sont la surcharge syntaxique nécessaire à la reconstruction de l'image à partir de l'image de référence. Ce cas simpliste devient nettement plus complexe lorsque du mouvement est perçu entre les images. L'estimation du mouvement revient à une recherche exhaustive dans un espace bidimensionnel, et ce pour chaque macrobloc de luminance (Y). Il est à noter que MPEG ne définit pas la façon dont cette recherche doit être faite, le choix de la méthode à utiliser est laissé à la discrétion du programmeur. Il s'agit néanmoins d'un compromis à faire entre complexité et qualité, tout dépend de l'application en question. On sait qu'une recherche exhaustive dans espace bidimensionnel va conduire au meilleur résultat dans la plupart des cas, au prix d'un coût important en terme de ressources matérielles pour l'encodeur. Comme l'estimation de mouvement est habituellement la tâche la plus coûteuse en temps dans un encodeur vidéo, certains encodeurs se limitent à une certaine zone de recherche, ou utilisent d'autres techniques. La méthode la plus utilisée pour limiter la redondance temporelle est de compenser les mouvements des macroblocs de l'image courante par rapport à une ou plusieurs images de référence.

Dans la suite, les étapes appliquées à chaque bloc de taille $m \times n$ sont détaillées, la figure 4.3 présente ces étapes sous forme de schéma. En général, les blocs sont de forme carrée, ce qui implique que m et n sont égaux. La première étape est de chercher une zone dans l'image de référence et de trouver une région de même taille correspondante dans l'image courante. Cette opération est réalisée en comparant le bloc de l'image courante avec toutes, ou une partie seulement des zones de taille $m \times n$ constituant l'image de référence. Cette première étape permettant de trouver la meilleure correspondance est l'estimation du mouvement. La seconde étape est la compensation du mouvement. La région choisie devient un prédicateur pour le bloc courant. Un bloc résiduel est formé par la soustraction du bloc courant (dans l'image de référence) avec la meilleure correspondance (dans l'image courante). Ensuite, le bloc résiduel est encodé et transmis ainsi que les vecteurs de mouvements. Ainsi, la reconstruction d'un bloc se fait à l'aide du vecteur de mouvement et des données du bloc résiduel.

L'unité de base pour la prédiction et la compensation du mouvement est le macrobloc, correspondant dans la plupart des cas à un ensemble de 16×16

pixels. Comme précédemment évoqué, les standards MPEG représentent chaque image à l'aide du format YCrCb4:2:0. Un macrobloc est donc constitué de 256 pixels pour la luminance (quatre blocs de 8×8), 64 pixels pour la chrominance bleue (un bloc de 8×8) et 64 pixels pour la chrominance rouge (un bloc de 8×8). Comme l'illustre la figure 4.4, chaque macrobloc est donc constitué de six blocs de 8×8 pixels. L'estimation du mouvement d'un macrobloc implique donc la recherche d'une région de 16×16 dans l'image courante. La région à traiter provient d'une image de référence passée ou future qui a déjà été encodée (et décodée). Une région de 16×16 pixels dans la zone de recherche est alors élue en utilisant un critère ressemblance appelée *best match*.

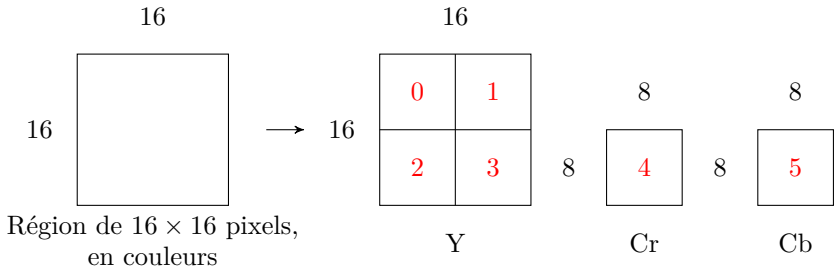


FIGURE 4.4 – Représentation d'un macrobloc dans le format YCrCb4:2:0.

La région élue comme la meilleure correspondance est soustraite du macrobloc courant pour créer un macrobloc résiduel. Ce résidu, ainsi que le vecteur de mouvement décrivant la position du meilleur macrobloc est alors encodé et transmis. Après la reconstruction d'un macrobloc, celui-ci est stocké en tant que référence pour d'autres prédictions de mouvement éventuelles. Le macrobloc prédit est reconstruit en ajoutant le résidu avec le macrobloc de référence pointé par le vecteur de mouvement. Il existe en grand nombre de variations pour l'estimation et la compensation du mouvement. L'image de référence peut être une image passée ou future, mais aussi une combinaison de deux ou plusieurs images déjà encodées. Néanmoins dans certains cas, il est préférable d'encoder le macrobloc sans utiliser le processus de compensation. Les changements entre l'image courante et l'image de référence sont tels que le coût des vecteurs de mouvements est trop important. Une optimisation, notamment mise en œuvre dans les standards MPEG est d'utiliser des tailles variables pour la compensation du mouvement, permettant de grands comme de petits déplacements.

La pratique montre qu'une compensation sur des blocs plus petits peut amener à de meilleurs résultats. Cependant, la complexité et le nombre de vecteurs de mouvements qui doivent être transmis peuvent l'emporter sur l'éventuel gain obtenu. Ce problème est résolu en adaptant la taille des blocs aux caractéristiques de chaque image, en utilisant de grands blocs pour les zones uniformes et des blocs plus petits pour les zones formées d'un nombre important de détails. L'estimation et la compensation du mouvement peuvent s'effectuer à une échelle inférieure au pixel, en utilisant une interpolation. La figure 4.5 présente des compensations de mouvement inférieures au pixel.

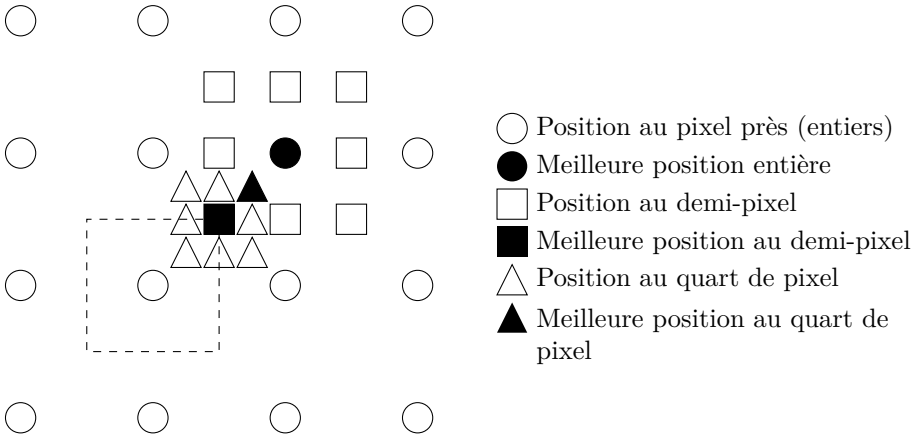


FIGURE 4.5 – Compensation de mouvement : interpolation jusqu'au quart de pixel. Le carré en pointillé représente l'espace occupé par un pixel.

4.1.3 Standards existants

Le groupe MPEG est un groupe d'experts de l'ISO/IEC en charge du développement de standards pour la représentation sous forme compressée des médias audio/vidéo. Depuis sa création en 1988, le groupe a produit plusieurs standards permettant à l'industrie comme à chaque particulier une utilisation simple. Plus de 125 standards existent aujourd'hui, mais nous nous focalisons ici seulement sur ceux relatifs à la compression vidéo. La famille MPEG en comprend cinq : le MPEG-1 [7], le MPEG-2 [8], le MPEG-4, le MPEG-4 AVC et le tout récent HEVC [9]. Les paragraphes suivants permettent au lecteur un survol rapide de ce que sont ces standards, mais aussi de leur évolution au fil du temps. Une vision détaillée des diverses normes est présentée dans [53].

MPEG-1

Ce standard a été développé par le groupe MPEG, sur demande de l'ISO en 1988. Il s'agissait à l'époque de développer un standard permettant de représenter des images en mouvement, avec une bande-son associée pour le stockage numérique. La première partie du travail fut terminée en 1991 et est connue sous le nom MPEG-1 [79], ou sous le standard ISO/IEC 11172 ayant pour titre *Coding of moving picture and associated audio*. La cible de ce premier format était les applications requérant un stockage numérique à des débits binaires pouvant atteindre 1,5 Mb/s.

MPEG-2

Un grand nombre d'applications demandant une résolution spatiale plus importante, comme celles basées sur le format CCIR 601 [10], une seconde phase de travail débute avec le MPEG-2 en 1990. En effet le MPEG-1 utilisait en entrée

7. Premier standard d'encodage vidéo du MPEG.
8. Deuxième standard d'encodage vidéo du MPEG, il s'agit d'une mise à jour de MPEG-1.
9. *High Efficiency Video Coding*
10. Format vidéo en 525×625

le format d'image SIF [11] qui possédait une résolution spatiale limitée. Il s'agit d'une extension de MPEG-1 qui apporte plus de flexibilité dans les formats d'entrée, des débits binaires plus importants ainsi qu'une meilleure tolérance aux erreurs. La norme a été conçue pour offrir des débits binaires variant de 4 à 15 Mb/s, ce qui répondait aux besoins des applications SDTV [12] et HDTV [13]. MPEG-2 [91] est définie par le standard ISO/IEC 13818 ou comme la recommandation H.262 [14] par l'ITU.

MPEG-4

Comparée aux standards MPEG-1 et MPEG-2, la compression offerte par MPEG-4 permet d'atteindre une qualité supérieure à des débits binaires inférieurs, tout en supportant des vidéos en haute résolution. Le MPEG-4 est une avancée majeure dans la compression des médias vidéo/audio, permettant la distribution de contenus à travers toutes sortes de technologies d'accès : de l'ADSL [15] à la fibre optique en passant par le Wi-Fi [16], le satellite ou tout simplement le DVD [17]. Ce standard est une boite à outils permettant de créer des flux vidéos et de les décoder pour n'importe quel contenu multimédia. À travers l'utilisation d'une structure standardisée, n'importe quelle forme de média peut être utilisée : du texte, des images ou encore des objets 2D ou 3D. Dans la norme MPEG-4, alors que les données vidéo/audio sont compressées à l'aide de techniques éprouvées basées sur les standards précédents, les éléments graphiques, le texte ou les objets synthétiques possèdent leurs propres méthodes de compression. En effet l'utilisation des caractéristiques intrinsèques des différents médias à compresser permet une représentation plus efficace en apportant de surcroît plus de flexibilité. MPEG-4 propose des outils de synthèse avancés permettant la description d'un son, l'animation des visages et des corps, la gestion des maillages 2D et 3D et le support de graphismes vectoriels. Cette norme permet en outre de diffuser des scènes multimédias interactives, mêlant 2D et 3D. La représentation d'un média sous la forme d'un graphe de scène permet la séparation de l'interaction avec chaque objet composant la scène, mais aussi un rendu adapté à chaque bande passante.

La partie 2 du standard MPEG-4 aussi connue sous le nom de MPEG-4 *visual* [78] permet une compression hybride mêlant images naturelles (à base de pixels) et images de synthèse (sous forme de maillages 2D et/ou 3D et de textures synthétiques). À cette fin, la partie 2 du standard comprend des outils et des algorithmes supportant la compression des différents objets pouvant composer une image. Pour la compression de vidéos naturelles grâce à MPEG-4 *visual*, le standard fournit les technologies de base pour le stockage, la transmission et la manipulation des textures dans un environnement multimédia. Ces outils permettent le décodage et la représentation d'objets atomiques de forme arbitraire relatifs à une image ou à une vidéo appelés « objets vidéos » (aussi notés VO [18]). La norme MPEG-4 met à disposition des algorithmes de

11. *Source Input Format*
12. *Standard-Definition TeleVision*
13. *High Definition TeleVision*
14. Standard d'encodage vidéo de l'UIT-T, équivalent de MPEG-2.
15. *Asymmetric Digital Subscriber Line*
16. *Wireless Fidelity*
17. *Digital Versatile Disc*
18. *Video Objects*

compression permettant une représentation efficace d'objets vidéos de formes arbitraires. Une grande partie des fonctionnalités introduites dans les normes MPEG-1 et MPEG-2 sont présentes comme la compression de vidéos standard (sous forme rectangulaire) ainsi que toutes les options relatives à la compression en elle-même telles que la modification des résolutions spatiales, temporelles et en qualité.

La partie 10 est une mise à jour majeure du système de compression vidéo. Elle définit une nouvelle norme d'encodage, nommée MPEG-4 AVC ou H.264 [19] et conjointement développée par le MPEG et l'ITU [76, 93]. Ce standard de compression vidéo se base sur les mêmes principes que ses prédécesseurs, en ajoutant un grand nombre de nouvelles fonctionnalités permettant de réduire encore davantage le débit binaire, tout en maintenant une bonne qualité. Le gain important dans l'efficacité de la compression est à mettre au détriment de la complexité et de la puissance nécessaire. La compression H.264 est basée sur la prédiction et la compensation du mouvement des macroblocs et sur l'utilisation d'une transformée. Selon les tests, MPEG-4 AVC permet de réduire le débit binaire de moitié (pour une qualité similaire) par rapport au standard précédent, MPEG-4. L'amélioration de l'efficacité tient dans deux contributions majeures :

1. La prédiction spatiale dans les images « intra », en utilisant des blocs spatialement voisins d'un bloc cible précédemment codé ;
2. Des améliorations dans l'encodage entropique (CAVLC [20] et CABAC [21]).

MPEG-4 AVC utilise à la fois des prédictions spatiales et temporelles pour améliorer le taux de compression. La compression « intra » image utilise une prédiction spatiale s'appliquant pour chaque partie de l'image, notée *slice* dans la norme. Une image est habituellement constituée d'une seule partie.

HEVC

Cette nouvelle norme est finalisée depuis janvier 2013 et est désignée sous la nomenclature ISO/IEC 23008 ou comme le H.265 [22] pour l'ITU. Elle doit succéder au MPEG-4 AVC/H.264. Ses principaux buts sont la prise en charge de la compression vidéo en très haute définition (2K, 4K, 8K...) ainsi que la diminution du débit binaire nécessaire à la transmission d'une vidéo sur les réseaux (à qualité constante). De ce fait, un plus grand nombre d'usagers sera éligible aux services audiovisuels, quelle que soit la technologie d'accès utilisée. Lors de sa normalisation, le HEVC a été évalué de façon subjective. Les tests montrent un gain de compression de l'ordre de 50% en 720p et de 60% en 1080p par rapport au MPEG-4 AVC et à qualité équivalente.

4.2 Méthode vecteur

Il n'existe actuellement aucun encodeur vidéo manipulant uniquement des primitives graphiques (associées à des profils couleur) en lieu et place de pixels.

19. Standard d'encodage vidéo de l'UIT-T, équivalent de MPEG-4 AVC.
20. *Context-Adaptive Variable-Length Coding*
21. *Context-Adaptive Binary Arithmetic Coding*
22. Standard d'encodage vidéo de l'UIT-T, équivalent de HEVC.

Néanmoins, des recherches à ce sujet sont en cours à l'université de Bath, au Royaume-Uni. De précédentes études ont permis la construction d'images vectorielles et leur rendu [30, 126]. La qualité du rendu atteint par cette méthode est équivalente à une représentation à base de pixels, c'est pourquoi ces images sont qualifiées de VPI [23]. Les chercheurs lançant ce nouvel encodeur vidéo à base vectorielle aiment à croire que leurs travaux vont entraîner la mort du pixel dans les prochaines années. L'encodeur a été lancé et mis en avant lors de la neuvième conférence CVMP [24] et porte le nom de VSV [25]. Chaque image est considérée comme un ensemble de contours avec des couleurs. La figure 4.6 met en avant une image traitée par cet encodeur.

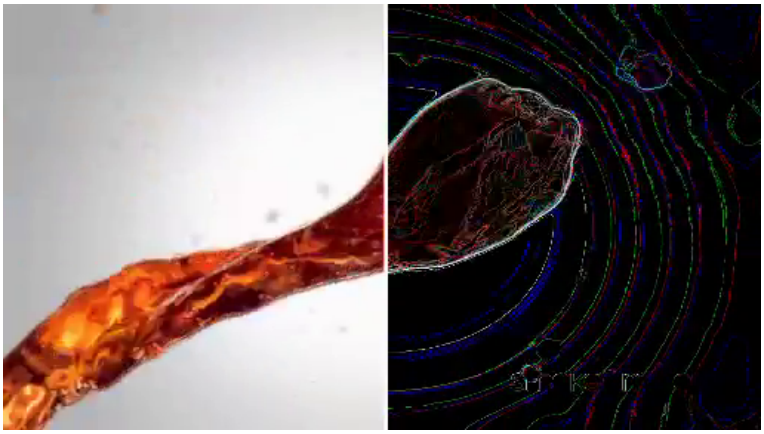


FIGURE 4.6 – Image extraite d'une vidéo vectorisée, montrant les primitives graphiques nécessaires à la reconstruction de l'image. Vidéo disponible en ligne[26].

On peut voir dans la partie droite de l'image les primitives graphiques utilisées pour sa modélisation. À l'heure actuelle, le problème majeur est le temps nécessaire au processus pour générer une vidéo au format VSV. Le temps réel n'est pour l'instant pas envisageable, même si le fonctionnement de VSV est selon ses auteurs facilement parallélisable. Utiliser un nombre suffisant de *thread* pourrait éventuellement permettre d'atteindre le temps réel. Cet encodeur produit donc un flux vidéo indépendant de la résolution à laquelle il sera affiché tout en maintenant une qualité maximale. Les images sont construites avec des modèles mathématiques (voir 3.2.2) qui permettent une représentation continue de l'information contrairement au monde pixel, qui utilise une approche discrète. À partir du même fichier vidéo l'utilisateur peut modifier le nombre d'images pas seconde, la profondeur des couleurs ou encore la résolution spatiale. Les informations sur cette nouvelle approche sont relativement peu nombreuses. Même si un brevet [59] permet de comprendre le fonctionnement détaillé de la vectorisation d'une image au format VPI, le passage à la vidéo n'est pas décrit.

Outre les travaux menés à l'université de Bath, on peut citer ceux réalisés dans [10]. Les auteurs y comparent en effet la compression d'un dessin animé (sous forme de scènes 2D) en utilisant deux méthodes distinctes : une com-

23. *Vectorized Photographic Images*
24. *Conference on Visual Media Production*
25. *Vectorized Streaming Video*
26. `http://www.cs.bath.ac.uk/vsv`

pression se basant sur le standard BIFS et une seconde se servant de ce même standard, légèrement modifié par les auteurs. Aucune vectorisation n'est réalisée dans la méthode proposée, les primitives graphiques composant les différentes séquences animées sont connues. Le but des auteurs est de mettre en avant le fait que la compacité offerte par le standard d'encodage BIFS peut être largement améliorée, en appliquant juste quelques régles simples. Les résultats obtenus confortent leur point de vue, puisque la taille des fichiers compressés grâce à leur version modifiée du standard BIFS est bien inférieure à leur pendant, traité avec le standard BIFS. Ces travaux mettent aussi en avant la pertinence de l'utilisation d'une compression vectorielle face à une approche pixel pour la compression de dessins animés : la taille des fichiers peut-être divisée par trois. Il faut toutefois panacher ces résultats, obtenus avec MPEG-4 : il en serait différemment si les tests étaient menés avec le standard MPEG-4 AVC qui offre un gain moyen (en terme de débit binaire) de 50% face à son aîné.

4.3 Compression bas et très bas débit de séquences d'images

Sur la base des représentations présentées dans les paragraphes précédents (voir les chapitres 3 et 4), diverses optimisations ont été introduites pour atteindre le bas et le très bas débit. Une compression vidéo très bas débit est définie comme ayant un débit binaire compris entre 8 et 64 kb/s [31]. Dans la suite de cette section, les méthodes permettant d'atteindre le très bas débit sont séparées en deux familles : la première comporte des méthodes utilisant la notion de bloc, la seconde regroupe des méthodes alternatives.

4.3.1 Encodage classique par bloc

On trouve dans la littérature un certain nombre d'articles [63, 89, 19] essayant de concilier les avantages des différentes transformées mathématiques existantes. Par exemple, dans [89] les auteurs utilisent à la fois la DWT et la DCT. Lors de la compression d'une vidéo, les images I et P représentent toute deux une image, mais leur contenu est différent. En effet, alors que les images I représentent réellement une image, les images P sont obtenues par soustraction : leurs caractéristiques sont donc bien différentes. Ce mécanisme a pour effet de produire des images avec beaucoup de détails, entrainant l'apparition de hautes fréquences. La DWT étant bien plus efficace sur les images haute fréquence, elle y est utilisée. La DCT est elle uniquement appliquée pour le traitement des images I.

4.3.2 Encodage non classique

Là encore, il existe beaucoup d'approches différentes, chacune d'entre elles se focalisant généralement sur l'optimisation d'une étape de la compression vidéo.

Encodage récursif des pixels

Dans cette méthode, le vecteur de mouvement est estimé de façon récursive, en minimisant une fonction de non-ressemblance, non linéaire, entre deux régions

situées dans deux images consécutives. Le terme région utilisé peut s'appliquer à un groupe de pixels comme à un pixel seul. L'algorithme orignal fut proposé par [28], par la suite diverses optimisations algorithmiques seront proposées [16].

Encodage basé sur le flux optique

Cette approche est bien plus précise pour l'estimation des vecteurs de mouvements que les méthodes d'encodage récursif des pixels ou les méthodes de recherche par bloc. En effet, dans ce cas-là, un vecteur de mouvement est calculé pour chaque pixel [95]. Le flux optique est la distribution 2D des vitesses apparentes des mouvements détectés entre deux images.

Encodage basé sur la quantification vectorielle

La quantification vectorielle [23] est une technique de compression vidéo non standard, mais très efficace dans le cas de la compression de données, puisqu'elle exploite la corrélation entre les composantes d'un vecteur. Le codage optimal peut être atteint en considérant une dimension de vecteur infinie : dans ce cas, la corrélation entre tous les composants est exploitée.

Encodage basé sur un modèle

Ce type de codage est un composant fondamental du codeur MPEG-4, le concept existait cependant bien avant. Ce type de codage fut introduit en 1981 par [147], il est alors dérivé du codage par objet. Ce type de codeur est souvent réservé à quelques domaines spécifiques comme la vidéo phonie dans lesquels on considère seulement un ou deux objets, ainsi qu'une connaissance de la scène courante. Depuis son introduction, ce type de compression a continué d'évoluer et il existe des articles bien plus récents comme [145] dans lequel il est question de l'encodage d'un visage.

4.4 Représentation et compression hybrides de séquences d'images

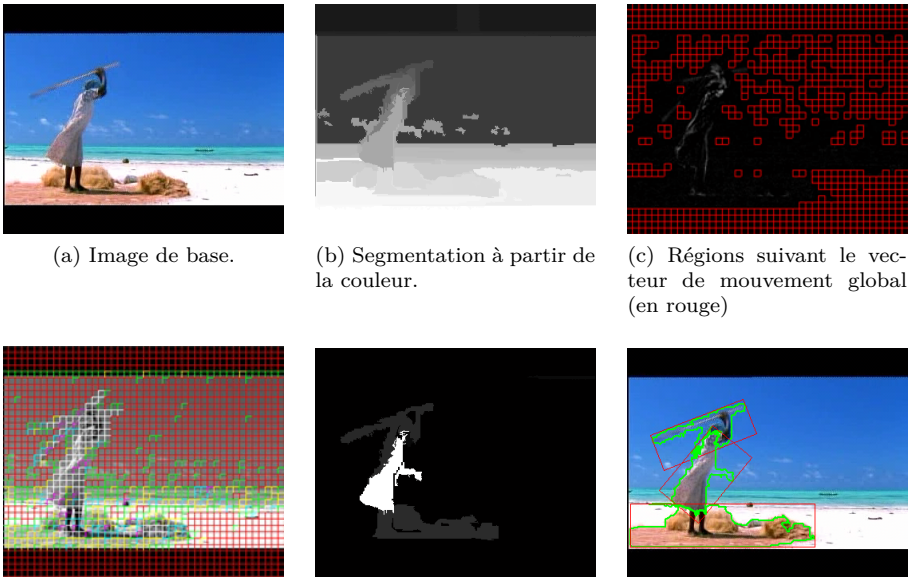
Cette section regroupe des méthodes de compression vidéo non standard, utilisant au moins deux méthodes différentes – mais associées – pour la compression d'une même vidéo. La littérature ne contient que peu d'articles à ce sujet. À notre connaissance, seules trois répondent au critère précédemment cité. Elles sont toutes présentées ci-dessous sous la forme de paragraphes distincts.

Dans [55, 105] certains macroblocs sont considérés comme pouvant être codés de manière paramétrique, en utilisant un processus autorégressif. Pour sélectionner les blocs, un critère basé sur la reconnaissance des bords est utilisé. Les textures mouvantes peuvent être modélisées par un processus autorégressif spatio-temporel, qui est en fait une version tridimensionnelle du modèle de base. Dans le système d'encodage proposé, les images I sont traitées par l'encodeur MPEG-4 AVC. Seules les images P sont potentiellement codées avec la solution proposée, en utilisant un macroblocs de 16×16 pixels comme élément de base. Les macroblocs sont ensuite classés en deux catégories : ceux contenant des

bords ou non. Les blocs ne contenant pas de bords sont codés avec le schéma proposé. La qualité a été mesurée avec une métrique subjective – le MOS – puisque la reconstruction est faite de manière statistique.

Les auteurs de [151] proposent de supprimer certains macroblocs durant la phase de compression, pour ensuite les reconstituer par synthèse au niveau du décodeur. Les zones de l'image manquantes lors de la décompression sont recréées grâce à l'utilisation d'un algorithme d'*inpainting* (*spatio temporal patch searching*). Le système fonctionne avec un GOP composé d'images I, P et B. Cette solution à l'avantage de réduire le nombre d'informations que l'encodeur MPEG-4 AVC a à envoyer, tout en évitant d'envoyer de nouvelles informations nécessaires à la méthode compression alternative.

La solution proposée par [41] combine une approche par bloc et un codage à base de modèle. L'utilisation d'un codage à base de modèle a pour but l'utilisation de la redondance temporelle sur le long terme. La détection des objets est réalisée grâce à l'utilisation d'algorithmes ciblant les ROI [27], plus précisément à partir d'une analyse en composante principale. Les zones d'intérêt (ROI) sont alors segmentées par un algorithme *graph-cut* [13]. Une analyse des zones résultantes est ensuite réalisée à des fins d'optimisations. Pour suivre l'évolution des régions d'intérêt au fil de la vidéo, un *tracker* est mis en œuvre. Chaque ROI est alors définie par un rectangle et est ensuite représentée à l'aide d'un modèle (un AAM [28]). Les six grandes étapes de cette approche sont présentées au moyen de la figure 4.7.



(a) Image de base.

(b) Segmentation à partir de la couleur.

(c) Régions suivant le vecteur de mouvement global (en rouge)

(d) Régions suivant d'autres vecteurs de mouvement.

(e) ROI sous forme de labels.

(f) Silouettes des objets d'intérêt (en vert).

FIGURE 4.7 – Les différentes étapes du processus utilisé dans l'encodeur hybride de [41].

27. *Region Of Interest*
28. *Active Appearance Model*

4.5 Conclusion

Au moyen de ce chapitre, une revue de l'état de l'art sur la compression vidéo a été proposée. Le fonctionnement générique d'un encodeur a tout d'abord été rappelé. Cette vue d'ensemble des rouages internes utilisés lors d'un encodage nous a permis d'identifier les leviers sur lesquels il est possible d'agir pour réduire le débit binaire. Le chapitre 3 a mis en évidence que la représentation d'une image grâce à l'utilisation de pixels n'est pas la seule possible. Une représentation utilisant des primitives graphiques et des profils couleur peut avantageusement être utilisée. Dans le domaine de la vidéo, l'encodage vectoriel n'en est qu'à ses balbutiements, d'importants défis restent à relever. Certaines solutions hybrides mêlant l'approche pixel avec une seconde méthode de compression – fondamentalement différente – existent cependant. La dernière section de ce chapitre a mis en avant leur fonctionnement et a montré que notre approche est différente. En effet les rares solutions hybrides présentes dans l'état de l'art utilisent toutes une première approche à base de pixels (grâce à l'encodeur MPEG-4 AVC), mais aucune n'utilise la notion de représentation vectorielle comme seconde méthode de compression. Il est aussi nécessaire de noter que ces trois méthodes utilisent des versions standard de la norme MPEG-4 AVC. L'approche hybride que nous proposons mêlera donc une compression pixel associée à une représentation sous forme de primitives graphiques associées à des profils couleur tout en proposant diverses optimisations afin que le fonctionnent des deux encodeurs soit optimal.

Chapitre 5

Rendu à distance

Sommaire

Ce chapitre a pour but de présenter une liste exhaustive des méthodes pour jouer, ou d'une manière plus générale, pour visionner et interagir avec du contenu 2D ou 3D à distance. La littérature est très riche dans ce domaine et un grand nombre de travaux peuvent être mentionnés [67, 96]. En prenant en compte comme facteur discriminant la nature des données transmises entre serveurs et clients, il est possible de distinguer quatre familles :

1. Les commandes graphiques ;
2. L'utilisation de pixels ;
3. Les primitives graphiques 2D et/ou 3D ;
4. La transmission d'objets 3D.

Les différentes catégories présentées ci-dessus font l'objet d'une description approfondie dans les sections suivantes. Une classification plus détaillée sera proposée à la fin de ce chapitre. Une section présentant des méthodes utilisant une combinaison des méthodes de base complétera ce chapitre.

5.1 Solutions à base de commandes graphiques

L'idée générale derrière cette solution est d'intercepter les commandes graphiques envoyées par une application à la carte graphique à travers l'utilisation d'une API [1] donnée. Le rendu est donc calculé côté serveur, et les commandes graphiques interceptées sont envoyées au client. Celui-ci est alors en mesure de réaliser le rendu. Il existe actuellement deux interfaces de programmation graphiques que sont DirectX [2] (pour les systèmes d'exploitation Microsoft Windows et Windows Phone) et OpenGL [3] qui est compatible à la fois avec les systèmes d'exploitation Linux et Microsoft Windows.

L'interception est réalisée en remplaçant la librairie utilisée par le système avec une API qui possède des prototypes de fonctions identiques. De ce fait aucun changement n'est nécessaire dans l'application appelant cette librairie, la modification étant totalement transparente. Ce principe est d'ailleurs utilisé dans WireGL [99]. Dans la librairie, les commandes graphiques sont traitées, compressées et envoyées au client à travers le réseau. Ceci implique que le terminal client doit posséder certaines spécificités matérielles. Puisque les opérations de rendu sont faites sur le client, la carte graphique du serveur n'est pas sollicitée. C'est un avantage qui permet de faire baisser les capacités de calcul requises sur le serveur. Cela signifie aussi que la résolution à laquelle sera affichée le rendu n'impacte pas les performances du serveur.

Il faut cependant savoir que certaines commandes graphiques ont besoin d'un retour de la part de la carte graphique. La solution la plus simple est alors de demander ces données au client. Il ne faut néanmoins pas oublier de prendre en compte le délai d'acheminement sur le réseau. Si le nombre de requêtes reste raisonnable, le réseau sera capable d'assumer cette tâche sans problème, mais si leur nombre vient à augmenter de manière significative, cette solution ne sera

1. *Application Programming Interface*
2. Collection propriétaire de bibliothèques destinées à la programmation d'applications multimédia.
3. Pendant libre de DirectX.

plus adaptée à des systèmes en temps réel.

Une solution à ce problème est proposée par [72]. Leur proposition est simple et astucieuse : simuler l'état de la carte graphique côté serveur. Quand le jeu démarre, la simulation est initialisée en prenant en compte les capacités graphiques du terminal client. Pour cela, une seule requête est utilisée. Quand le jeu attend des informations de la part de la carte graphique (du terminal client donc), il les reçoit de la part de la simulation locale. Cette solution ajoute une surcharge, mais l'amélioration des performances du jeu est nettement plus importante.

Un autre comportement peut être observé entre le client et le serveur : la bande passante nécessaire change de façon irrégulière au cours du temps. Ceci est dû à la nature même des commandes graphiques. En effet, une grande partie des données est transférée au début, ou lorsqu'il y a un changement de scène. La taille de ces données peut être limitée en utilisant des systèmes de compression ou de mise en cache, comme réalisé dans [120].

5.2 Solutions à base de pixels

Dans cette approche le rendu est entièrement réalisé sur le serveur. L'image de sortie est capturée, puis envoyée sur le client. Différentes approches ont été imaginées pour minimiser la taille des données de sortie, à travers l'utilisation d'un système d'encodage adapté. Il est possible de les séparer en trois sous-catégories : l'utilisation d'images, de vidéos et de X11.

5.2.1 Image

Dans cette catégorie, les images sont simplement compressées et envoyées indépendamment les unes des autres sur le réseau. Dans l'architecture proposée dans [87] des images complètes sont envoyées seulement lorsqu'il n'y a pas de mouvement. Des images dont la résolution spatiale a été réduite sont transmises lorsqu'une certaine quantité de mouvement est atteinte. Cette tactique permet de réduire la bande passante nécessaire, mais si l'application utilisée génère la plupart du temps des images de taille réduite, la qualité obtenue ne sera pas bonne. Cette approche permet aussi de capturer une image au moment voulu, et non à des intervalles de temps réguliers comme l'impose la méthode basée sur un flux vidéo. Ceci permet d'avoir un meilleur contrôle sur le nombre d'images envoyées par seconde, et donc de le réduire, ou tout simplement de ne pas envoyer d'image quand il n'y a pas eu de changement. Inversement, quand il y a beaucoup de mouvement, le nombre d'images par secondes peut être augmenté pour refléter fidèlement les changements. Envoyer constamment des images en haute qualité est une bonne solution pour certaines applications comme de la réalité mixte [107] ou des rendus photo réalistes [65], mais elle n'est pas vraiment adaptée quand il y a beaucoup de mouvement, comme dans les jeux vidéos, par exemple. Ainsi la quantité de données à envoyer peut rapidement devenir très importante, et dégrader de façon tout aussi rapide la réactivité du système. Dans certains cas [71], on peut restreindre le nombre de possibilités de déplacements et ainsi prédire le prochain mouvement du joueur. En utilisant ces informations, le système peut rendre les différentes possibilités de déplacement et envoyer les images ainsi prédites au client (les images correspondant au mouvement le plus probable étant priorisées). Cependant, dans la plupart des jeux, la restriction du

mouvement ne peut être une option, ce qui rend l'application de cette méthode délicate.

5.2.2 Vidéo

La caractéristique principale de cette catégorie est d'utiliser un flux vidéo pour transférer les images rendues par le serveur sur le client. Les images sont capturées avec un débit constant, encodées avec un codec vidéo [74], puis diffusées au client. Par exemple dans l'architecture proposée dans [120], les jeux vidéos fonctionnent dans des machines virtuelles. Ils peuvent ainsi partager les capacités matérielles présentes sur le serveur, et en particulier l'accélération graphique. Dans [98] le serveur peut ne pas se limiter à une seule machine, mais être un ensemble d'ordinateurs se partageant le travail. Des scènes plus complexes peuvent alors être rendues et affichées, ce qui n'aurait pas pu être possible avec l'utilisation d'un seul ordinateur. Les données médicales utilisées dans [108], ou les données provenant d'un scanner 3D peuvent occuper plusieurs dizaines de gigaoctets en mémoire, ce qui rend leur simple manipulation sur un seul ordinateur impossible. Côté client, seul un lecteur vidéo est nécessaire. Il se doit de supporter le flux vidéo qu'il reçoit et d'utiliser le décodeur adapté pour décoder le flux entrant. La puissance doit être suffisante pour qu'un décodage en temps réel de la vidéo soit possible. Un des avantages à utiliser un flux vidéo est que le débit peut être prédit. Ceci permet d'adapter de façon fine et précise les capacités requises au niveau du réseau de transport. Cette solution est acceptable pour les appareils disposant d'une connexion au réseau rapide, ce qui n'est généralement pas le cas des terminaux mobiles qui ont accès à des bandes passantes bien plus faibles. La solution est alors d'adapter la vidéo aux capacités du réseau, ce qui va engendrer une baisse de qualité et donc une expérience utilisateur qui devient vite inacceptable.

5.2.3 X11

Ce type d'architecture est principalement utilisé pour les applications de bureau à distance. Les formes de base utilisées par le système d'exploitation (fenêtres, boutons, boite de dialogue) sont utilisées pour optimiser le protocole de transmission. L'un des premiers d'entre eux est le X11 qui a été introduit dans [36]. Toutefois, si un élément ne fait pas partie du groupe de base, il est envoyé en tant qu'un ensemble de pixels au client. Il existe plusieurs tentatives ayant pour but d'améliorer le protocole X11, une des plus récentes [5] propose d'utiliser seulement cinq des fonctions de base. Ils utilisent aussi un système de mise en mémoire cache, qui permet de ne pas envoyer des commandes qui seraient rendues inutiles par d'autres (chevauchement), il en résulte une optimisation du trafic réseau. Si l'utilisateur regarde une vidéo sur le serveur, le système la détecte et la diffuse au client. Cette méthode ne supporte cependant pas la 3D. Un support de la 3D a été envisagé dans [136] où il est proposé de rendre les scènes 3D en utilisant OpenGL côté serveur, et d'envoyer ensuite le résultat sous forme d'images au client. La solution devient alors similaire à celles basées sur l'envoi d'images seules.

5.3 Solutions à base de primitives graphiques

Le principe de base appliqué dans cette architecture est d'utiliser un serveur pour stocker des données. Quand le client a besoin de rendre une scène, il demande les données au serveur, qui les lui envoie en retour. Le type de primitive graphique peut généralement être 2D ou 3D, ce qui permet de constituer deux groupes distincts.

5.3.1 Primitives 2D

Un exemple d'architecture utilisant des primitives graphiques est présenté dans le travail de [83]. Un nouveau module est introduit après les opérations de rendu. Celui-ci permet de convertir des images 3D en un ensemble de vecteurs 2D en extrayant différentes sortes de lignes (les contours notamment) et en les traitant jusqu'à obtenir des vecteurs 2D. Les vecteurs 2D sont ensuite envoyés au client, où ils sont rendus localement. À cause de la nature même des données 2D, aucun calcul complexe n'est nécessaire, ce qui est une bonne chose pour les terminaux mobiles qui ne supportent pas la 3D, mais aussi pour ceux limités en puissance de calcul.

5.3.2 Primitives 3D

Avec des techniques basées sur les vecteurs 3D, la géométrie d'un objet peut est approximée par un ensemble de vecteurs, opération qui est réalisée dans une phase de préparation. Les vecteurs 3D sont ensuite envoyés au client, qui peut alors effectuer les opérations de rendu. Évidemment, le terminal client doit supporter la 3D. Puisqu'il ne rend que des vecteurs, il n'est pas nécessaire d'avoir un support de fonctionnalités avancées, comme les textures. Rendre des lignes prend en outre bien moins de temps processeur que rendre un maillage texturé. En conséquence, le nombre d'images par seconde sera bien meilleur avec un matériel équivalent. La création de vecteurs 3D prenant beaucoup de temps processeur, cette étape est réalisée au moment où les objets sont chargés en mémoire et non lorsqu'ils sont demandés par le client. Le traitement est réalisé en rendant chaque objet avec sa texture associée, puis en extrayant les vecteurs. Cette méthode permet d'obtenir plus de détails sur les objets par rapport à celle où seul le contour est extrait (mentionné dans 5.3.1). Une architecture basée sur une approche utilisant des vecteurs 3D a été proposée par [128]. Elle est mise en œuvre pour afficher le modèle 3D d'une ville, à des fins de navigation. Cette application a été testée en utilisant un terminal mobile et les auteurs ont conclu que ce type de système de rendu était approprié et permettait de reconnaître les façades des bâtiments.

5.4 La transmission d'objets 3D

Dans ce groupe, des objets graphiques en 3D sont transférés. Du côté client, le rendu d'objet doit évidemment être supporté. Puisqu'on se positionne sur l'utilisation – entre autres – de terminaux mobiles, le serveur doit prendre en compte leurs capacités, mais aussi être en mesure d'adapter le contenu si besoin. On peut séparer une nouvelle fois ce type de techniques en deux groupes : le

premier où seulement un objet 3D est transféré, le second où plusieurs objets 3D sont envoyés. Une optimisation est éventuellement appliquée, sur l'objet seul dans le premier cas, et sur l'ensemble des objets dans le second.

5.4.1 Un seul objet

La plupart des difficultés au transfert d'un objet 3D sont liées à la nature non fiable du réseau. Ce problème apparaît quand les objets doivent être envoyés et reçus avec une latence faible. Le protocole TCP [4] est certes fiable, mais fait augmenter de manière significative la latence sur des réseaux avec de fortes pertes. L'alternative offerte par le protocole UDP [5] permet de délivrer l'information avec une latence plus faible, mais il n'est pas fiable. Du fait de l'importance primordiale de la latence, la plupart des applications interactives 3D utilisent le protocole UDP. La particularité non fiable d'un protocole peut être en partie résolue en utilisant des codages résistants aux erreurs. En fonction de la manière dont sont écrites les données, on peut classer les algorithmes en deux catégories : segmentation et progressif.

Les algorithmes de segmentation [4, 47] découpent le maillage en plusieurs parties qui sont transférées au client séparément. Ainsi, si une partie du maillage est perdue, les autres segments sont toujours présents. La partie manquante peut alors être régénérée (en aucun cas on ne peut être sûr que cette partie ressemble à la vrai, qui a été perdue) en utilisant une interpolation, qui aura un coût : le client devra fournir plus de temps processeur. Différentes techniques permettant d'optimiser la segmentation du maillage existent et promettent d'obtenir le meilleur résultat visuel possible.

Les algorithmes progressifs suppriment des points dans le maillage, les simplifiant ainsi comme cela a été fait dans les solutions proposées par [65] et [70]. Les détails au sujet des points supprimés sont stockés et regroupés dans une seule structure de données. Il est ainsi possible de restaurer ces points dans le maillage. La suppression de points est réalisée en plusieurs étapes, jusqu'à l'obtention d'une forme présentant le nombre de points voulus. Chaque étape permet de définir un nouveau niveau de détail (LOD [6]). Puisque la restauration des points dépend des niveaux de détail précédents, si un niveau de détail est perdu, il est impossible de restaurer le maillage à un niveau de détail plus élevé. Le plus gros problème survient lorsque le maillage de base est perdu. Cela veut dire que tous les niveaux de détails qui sont ensuite reçus ne peuvent pas être reconstruits et que l'objet 3D ne pourra donc pas être rendu dans la scène. Le maillage de base est donc, dans la plupart des cas envoyé en utilisant des moyens sûrs : soit en utilisant le protocole TCP, soit en utilisant une version sûre d'UDP.

5.4.2 Plusieurs objets

Le défi de transférer plusieurs objets 3D est principalement dû à l'ordre dans lequel ces objets sont envoyés au client. Habituellement, les objets composant

4. *Transport Control Protocol*
5. *User Datagram Protocole*
6. *Level Of Detail*

une scène sont organisés sous la forme d'un arbre, que l'on appelle le graphe de scène. Il contient la position de chaque objet dans la scène, les regroupements éventuels et les connexions existantes entre eux. Dans le but de réduire l'empreinte mémoire, il est aussi possible d'avoir des occurrences multiples d'un même objet dans différentes parties de la scène. L'objet n'est pas copié, mais une instance de cet objet est créée dans la branche en question du graphe. Dans la plupart des cas, la structure du graphe de scène est simple et relativement plate. Elle inclut au moins un niveau et n'est donc pas prise en compte dans les opérations de compression. Une application basée sur une cartographie de ville virtuelle produit un graphe de scène de taille importante, où un grand nombre de données doit être manipulé. Les données ne sont donc pas toutes transférées en même temps, mais de façon progressive. Elles sont de plus choisies en fonction de la position de la caméra virtuelle. Seuls les bâtiments proches sont rendus en détail. Ce type d'architecture a été proposé par [121] et amélioré dans [122]. Les bâtiments sont regroupés en fonction de leur position dans l'espace. Cependant, seuls les groupes à l'intérieur du champ de vision et proches de la caméra sont transférés au client. Dans chaque groupe, les bâtiments situés dans le champ de vision sont à leur tour ordonnés en fonction de leur distance à la caméra. À partir de cette distance, différents niveaux de détail sont alors utilisés, et seuls ceux nécessaires sont chargés. Les bâtiments les plus éloignés ne sont tout simplement pas texturés, une simple coloration étant utilisée à la place. Les bâtiments spécifiques, comme ceux servant de repère sont traités séparément. L'architecture proposée dans [80] utilise plusieurs métriques pour calculer si un bâtiment doit être envoyé ou non au client. Les bâtiments sont classés après le calcul d'une carte d'importance. Les systèmes, basés sur une base de données, créent une carte pour chaque utilisateur en fonction de ses requêtes. Le système apprend ce que l'utilisateur a cherché par le passé, et utilise ce savoir pour classer les résultats. Les ressources matérielles du terminal client, ainsi que la bande passante entrent aussi en considération. Dans [3] une représentation plus extensible est utilisée, permettant une allocation optimisée des bits de données pour la transmission. Dans le cadre de la navigation urbaine, un modèle 3D optimisé de téléchargement progressif est présenté dans [120].

5.5 Solutions mixtes/adaptatives

Les méthodes adaptatives utilisent une combinaison de deux ou plusieurs méthodes précédemment mentionnées. On peut regrouper ces méthodes en fonction du critère qui va permettre de décider de l'utilisation d'une méthode ou d'une autre.

5.5.1 Quantité de mouvement

Une méthode proposée dans [81] détecte la fréquence à laquelle l'image est modifiée à l'écran. Si les changements sont minimes, une méthode basée sur l'image est utilisée. Mais lorsque cette même fréquence de modification devient trop importante, le système change de méthode et utilise alors une solution basée sur la vidéo. L'avantage apporté par l'adaptation est relatif à l'utilisation faible de la bande passante lorsque les changements sont rares. Lorsque la quantité de mouvement augmente de façon importante, par exemple dans le cas d'une vidéo

ou d'un jeu, le système reste réactif, tout en limitant l'augmentation de la bande passante. Dans [83], une approche hybride a été adoptée. Lorsque la quantité de mouvement entre deux images n'est pas très importante, le protocole X11 est utilisé, mais lorsqu'elle augmente et atteint un seuil prédéfini, un flux vidéo MPEG-4 AVC alias H.264 est utilisé.

5.5.2 Mouvement de la caméra

L'architecture utilisée dans cette méthode est basée sur des niveaux de détails différents. Le modèle qui comporte un niveau de détail bas est utilisé par le client, et celui avec un niveau de détail élevé par le serveur. Quand l'utilisateur se déplace dans la scène, seul le modèle dégradé est affiché. Quand le mouvement de l'utilisateur cesse, le client envoie les paramètres de la caméra au serveur. Celui-ci calcule alors une image de meilleure qualité et l'envoie au client. Ce système proposé par [106] n'est pas adapté aux applications où le mouvement est quasiment toujours continu, comme dans les jeux ou les vidéos.

5.5.3 Distance à la caméra

Dans le cas où il y a beaucoup d'objets dans la scène, ceux qui sont proches de la caméra ont habituellement une importance plus élevée que les autres. De ce fait, en triant les objets en fonction de la distance qui les sépare de la caméra, une priorité peut-être attribuée. En fonction des capacités du terminal cible, les objets les plus proches sont envoyés avec un maximum de détails, ceux un peu plus loin avec moins de détails et les objets vraiment éloignés ne sont pas envoyés du tout, ou alors simplifiés au maximum. Dans [102] les auteurs proposent de rendre les objets près de la caméra sur le terminal cible, quand tous les autres le sont sur le serveur. Ceci permet de réduire la bande passante nécessaire, d'augmenter la fréquence à laquelle la scène est rendue, et d'afficher le résultat avec une image de qualité acceptable. Un autre exemple est la méthode proposée par [121] et [122], où les bâtiments éloignés de la caméra sont affichés avec très peu de détails : la couleur est utilisée en lieu et place de la texture.

5.5.4 Importance auprès de l'utilisateur

Dans un scénario de navigation, les utilisateurs cherchent souvent un endroit spécifique. Par conséquent, on peut utiliser comme critère ce que recherche l'utilisateur, en plus de sa localisation. À partir de cette idée, des travaux ont été menés dans [80] où une base de données est utilisée pour retrouver des objets. Par exemple, si l'utilisateur cherche un hôtel dans une zone donnée, les hôtels (localisés par le système) seront affichés avec un maximum de détail, les bâtiments proches avec moins de détail et les autres bâtiments seront uniquement représentés en 2D.

5.6 Cas du *cloud gaming*

Le *cloud gaming* est aujourd'hui en plein essor. Un certain nombre de solutions aussi bien commerciales (OnLive, Gaikai ou encore StreamMyGame) que libres [96] existent. L'intégralité de ces solutions utilise une approche pixel, à

base de vidéo (voir 5.2.2) comme solution de transport depuis le serveur jusqu'aux clients. La norme MPEG-4 AVC, référence actuelle dans le monde de l'encodage vidéo est privilégiée pour la compression. Le *cloud gaming* étant une partie importante du travail proposé dans ce manuscrit, le chapitre 6 lui est entièrement consacré.

5.7 Résumé et conclusion

Les sections précédentes nous permettent de classer les différentes méthodes de rendu à distance selon plusieurs critères. Ceux-ci sont énumérés ci-dessous :

a. La bande passante nécessaire (BP) au bon fonctionnement du service distant ;
b. Les saturations éventuelles du réseau (SAT) ;
c. L'expérience perçue par l'utilisateur (EU), c'est-à-dire la qualité effective perçue pour un utilisateur donné ;
d. La puissance de calcul nécessaire (PC) du côté client ;
e. La fluctuation de la bande passante (FBP).

Table 5.1 – Comparaison de toutes les méthodes présentées.

		Critères				
		BP	SAT	EU	PC	FPB
Solutions proposées	Commandes graphiques	oui	non	oui	non	non
	Pixels	non	oui	oui	oui	oui/non
	Primitives 2D	oui	oui	non	oui	oui
	Primitives 3D	oui	oui	non	oui	oui
	Un objet 3D	oui	oui	non	non	oui/non
	Plusieurs objets 3D	oui	oui	oui	non	oui/non

La solution utilisant des commandes graphiques n'est clairement pas adaptée à la situation. En effet, elle demande une puissance de calcul importante côté client, mais est aussi sujette de par son fonctionnement à entraîner des fluctuations importantes de la bande passante qui peuvent entraîner dans certains cas une saturation du réseau. Cette approche ne sera donc pas retenue. Il en est de même pour les solutions utilisant un ou plusieurs objets 3D, qui demandent une puissance importante côté client. En fin de compte, seules trois entrées du tableau semblent satisfaire à nos besoins. La solution à base de pixels, et celle utilisant des primitives graphiques. Il est intéressant de remarquer l'alternance des « oui » et des « non » entre ces deux familles, néanmoins l'utilisation de primitives graphiques 2D est moins contraignante. La meilleure solution de rendu à distance semble donc une approche hybride, mêlant pixels et primitives graphiques 2D.

Chapitre 6

Le *cloud gaming*

Sommaire

Les chapitres précédents ont mis en avant les différentes solutions pour la représentation d'une image, et ont plus particulièrement présenté une vue d'ensemble des différentes méthodes utilisées par les applications de rendu à distance. Ce nouveau chapitre se focalise sur une application particulière : le jeu à distance, plus familièrement qualifié de *cloud gaming*. Le *cloud gaming* est un exemple d'utilisation du *cloud computing*. Le *cloud computing*, ou informatique dans les nuages en français, est constitué d'un ensemble de machines physiques (des serveurs puissants d'un point de vue du processeur et de la mémoire vive notamment) hébergées le plus souvent chez un prestataire de services. Ce parc matériel est exploité à travers l'utilisation de systèmes d'exploitation spécifiques nommés hyperviseurs. Il en existe plusieurs types, de différentes générations, mais ils ont tous le même rôle : fournir une couche d'abstraction pour la gestion des capacités matérielles. Des machines virtuelles (car elles n'ont pas un accès direct au matériel) sont ensuite instanciées par le biais d'un hyperviseur (voir figure 6.1).

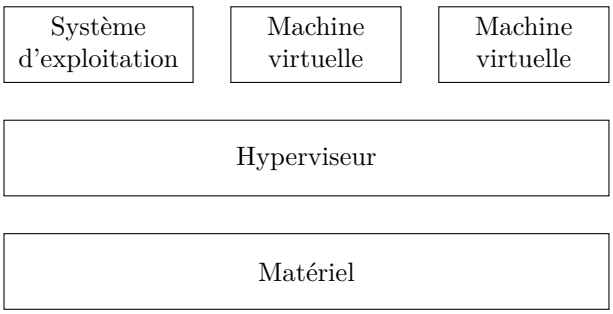


FIGURE 6.1 – Virtualisation de machines grâce à l'utilisation d'hyperviseurs.

Ce type d'architecture permet une grande souplesse d'utilisation : création, destruction de machines virtuelles à la volée, modification des ressources matérielles d'une machine... Le but est ici de proposer à l'utilisateur de jouer à n'importe quel jeu via le paradigme du rendu à distance. Pour réduire les coûts, les entreprises qui se sont lancées dans ce type d'activité cherchent bien sûr à maximiser le nombre d'instances (de jeux) tout en minimisant le nombre de machines physiques nécessaires. C'est en partie pour cette raison qu'elles utilisent toutes des *cloud*. Cette partie qui va permettre au jeu d'être exécuté sur un ou un ensemble d'ordinateurs physiques n'est qu'une partie d'un problème plus vaste. En effet, pour permettre à une application de jeu à distance de fonctionner correctement, plusieurs paramètres sont à prendre en compte :

a. La puissance nécessaire au jeu (CPU [1] et GPU [2]) ;
b. La technologie de rendu à distance utilisée (voir chapitre 5) ;
c. La gestion du réseau (bande passante, diffusion et latence) ;
d. La gestion des entrées/sorties côté client (affichage et envoi des informations du clavier et de la souris).

Actuellement, certains problèmes de performance subsistent. On peut notamment citer le problème de la virtualisation des cartes graphiques, qui n'est

1. *Central Processing Unit*
2. *Graphics Processing Unit*

toujours pas entièrement résolu à ce jour. Les communications entre une application graphique hébergée dans une machine virtuelle et la carte graphique (au sens matériel) ne sont pas efficaces : bande passante limitée, temps de latence importants. La puissance des cartes graphiques, pourtant essentielle dans des applications de *cloud gaming*, n'est donc pas encore correctement utilisée dans un environnement virtuel. Au travers d'un exemple générique d'architecture de *cloud gaming*, les paragraphes suivants offrent un aperçu des solutions existantes ainsi que de leur fonctionnement.

6.1 Solutions existantes

La majorité des solutions de *cloud gaming* sont propriétaires, mais il existe néanmoins des plates-formes totalement libres ou sous la forme de projets de recherche. Ces différentes approches sont traitées ci-dessous. Malgré leurs différences, il est possible de séparer la chaîne de rendu à distance en sept modules, qui sont présentés au moyen de la figure 6.2.

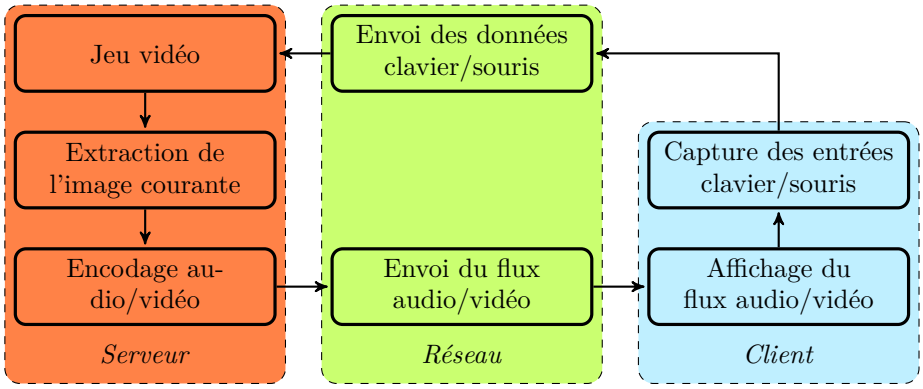


FIGURE 6.2 – Schéma synoptique d'une chaîne de rendu à distance pour le *cloud gaming*.

Dans la figure 6.2 la colonne de gauche représente les opérations effectuées sur le serveur, la colonne du milieu représente les échanges réseau et la colonne de droite constitue le client.

6.1.1 Solutions commerciales

Les systèmes actuels comme OnLive [3], Gaikai [4] ou encore StreamMyGame [5] mettent à disposition les technologies, les SDK [6] et aussi les services permettant de publier un jeu qui deviendra alors accessible aux clients à travers leur plate-forme. Ces services font référence à la première génération de *cloud computing* et prennent en charge le portage des moteurs de jeux sur les serveurs de manière transparente [100]. Le principal avantage est que le jeu lui-même est

3. `http://www.onlive.com`
4. `https://www.gaikai.com`
5. `http://www.streammygame.com`
6. *Software Development Kit*

juste adapté, il n'est en revanche pas optimisé pour le *cloud*. Par conséquent, chaque instance d'un jeu est indépendante et occupe les ressources qui lui sont nécessaires. Ces solutions sont basées sur l'envoi d'un flux vidéo, principalement compressé avec l'encodeur MPEG-4 AVC alias H.264.

6.1.2 Solutions libres & projets de recherche

Il n'existe pour l'instant qu'une seule solution (à notre connaissance) de *cloud gaming* entièrement libre [96]. Celle-ci prend pour parti de fournir une chaîne complète de rendu à distance adaptée à la problématique des jeux vidéos. Elle emploie pour cela des librairies bien connues et permet la mise en place d'un environnement de test relativement rapidement. Les modules permettant la capture des images rendues, ainsi que ceux permettant d'envoyer les événements créés par le clavier ou la souris sont dépendants de la plate-forme utilisée. Malgré le titre de l'article, aucune optimisation ayant trait à l'utilisation de la puissance du *cloud* n'est mentionnée. Chaque module de la chaîne de rendu est utilisé tel quel et ne fait pas l'objet d'optimisations particulières. Des technologies éprouvées comme la suite de protocole RTP [7] pour la diffusion en continu, et l'encodeur x264 [8] pour l'encodage du flux vidéo sont mises en œuvre.

Kusanagi [9] est un projet européen ayant pour but de fournir une infrastructure de rendu à distance pour des applications 2D ou 3D et ce de manière transparente de bout en bout. Plus généralement, il s'agit s'une solution logicielle qui permet à un utilisateur d'interagir à distance avec une scène multimédia grâce à un client multimédia ou directement pas le biais d'un navigateur grâce à une extension. L'architecture utilisée par Kusanagi est composée de trois éléments principaux : une application compatible avec Kusanagi, le *lobby* (un serveur de gestion) et un client MPEG compatible. La figure 6.3 présente cette architecture. Ce projet se focalise principalement sur les optimisations qu'il est possible d'apporter aux différentes étapes que sont le rendu, la compression, la diffusion en continu et l'affichage tout en s'attachant à n'utiliser que des standards issus du groupe MPEG. L'architecture proposée par Kusanagi est basée sur l'envoi d'un flux vidéo compressé à l'aide de MPEG-4 AVC, celui-ci étant décodé côté client par le lecteur GPAC [10] [111].

Le greffon Kusanagi

Le greffon Kusanagi est un ensemble logiciel qui permet à une application 3D d'être contrôlée à distance. Pour accomplir cette transformation, il est nécessaire de gérer l'envoi et la réception de deux types de données :

1. Les données constituant le flux audio/vidéo, du serveur vers le client ;
2. Les commandes permettant l'interaction, du client vers le serveur.

Pour permettre la transmission des commandes à l'instance du jeu distant, un serveur HTTP [11] est utilisé. Lors du lancement d'une session de jeu, le *lobby*

7. *Real-time Transport Protocol*
8. Implémentation *open source* de la norme MPEG-4 AVC.
9. `http://www.kusanagi.eu`
10. *GPAC Project on Advanced Content*
11. *Hyper Text Transfert Protocole*

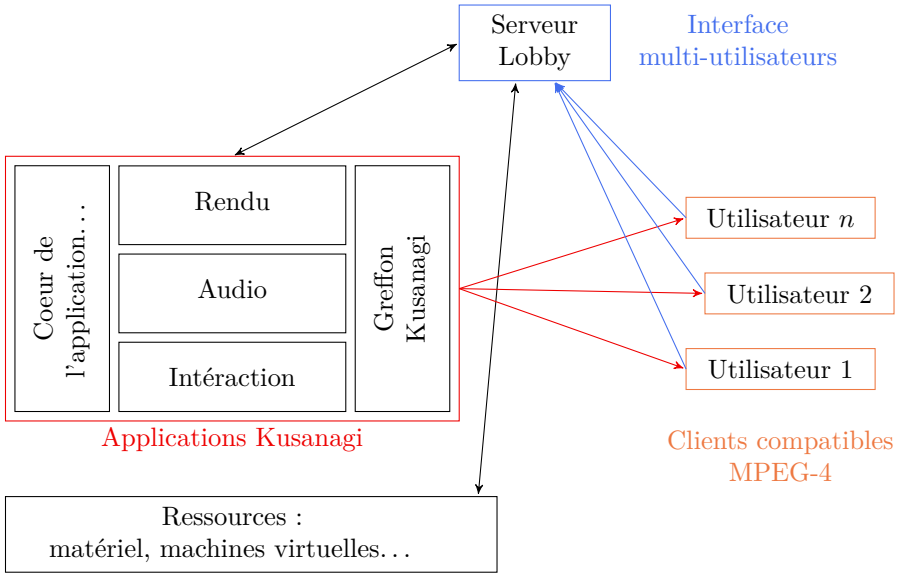


FIGURE 6.3 – Architecture de Kusanagi, image extraite de [143].

est mis en œuvre et décide des numéros de ports à utiliser. La communication est ensuite directe, entre le client et le serveur, le *lobby* n'ayant plus qu'un rôle de surveillance.

Le *lobby*

Le but de Kusanagi est d'offrir aux clients légers la capacité d'utiliser des applications requérant normalement une puissance de calcul importante. Ceci est rendu possible en laissant le terminal client contrôler l'application à distance grâce à l'envoi des données d'interaction et à la réception et à l'affichage d'un flux vidéo en temps réel sur son écran. l'application s'exécutant de son côté sur une machine puissante. Le *lobby* est nécessaire lorsque le système utilise plusieurs clients, plusieurs applications et plusieurs serveurs permettant de lancer ces applications. Il permet de gérer les interactions entre clients et serveurs, c'est-à-dire qu'il établit les connexions client/serveur nécessaires. Il s'assure que chaque client envoie les données d'interaction à la bonne application et que chaque flux vidéo est diffusé au bon client. Outre son rôle d'aiguilleur, le *lobby* s'acquitte d'autres tâches, elles sont mentionnées ci-dessous :

a. Contrôler l'ensemble du trafic entrant ;
b. Interpréter les requêtes des clients, vérifier leur validité et leur répondre ;
c. Gérer l'authentification des clients ;
d. Lancer les applications demandées par les clients ;
e. Limiter le nombre de connexions simultanées ;
f. Créer un lien entre un client et une application, et gérer sa vie.

On peut donc voir le *lobby* comme un administrateur de la plate-forme Kusanagi. Avant d'être lancé, il doit connaître un certain nombre de paramètres dont

le nombre maximum de clients autorisés, la liste des applications utilisables ou encore le nombre de serveurs d'applications... lorsqu'un utilisateur veut lancer une session de *cloud gaming*, la première étape est son authentification. L'identifiant et le mot de passe du joueur sont envoyés au *lobby* qui va vérifier la véracité de ces informations grâce à sa base de données. Si les informations sont correctes, la communication se poursuit, sinon elle est rompue. Dans la seconde étape, le client choisit une application à lancer dans la liste affichée à l'écran.Le *lobby* contrôle tout d'abord si le nombre de clients maximal est atteint. Cette vérification est importante, elle permet de gérer les ressources matérielles d'un serveur en limitant le nombre d'applications s'exécutant en même temps. Si tel n'est pas le cas, des numéros de ports uniques sont attribués à l'application et au client. L'application est alors lancée sur un serveur, ainsi que la diffusion vers l'adresse IP [12] du client et sur le port choisi. Le client reçoit les informations nécessaires à la réception du flux vidéo. À partir de cet instant, le client et l'application communiquent directement entre eux, sans passer par le *lobby*. Celui-ci s'assure simplement à intervalle de temps régulier que le client, l'application et les connexions sont actifs. L'arrêt d'une application gérée à distance se fait soit par le client (lorsqu'il décide de quitter ou de changer d'application), soit par le *lobby* si l'application ne répond plus (le client en est informé, et repasse à l'étape deux). Les ports qui étaient réservés pour la communication entre l'application et le client sont alors à nouveau rendus disponibles. La figure 6.4 offre un aperçu visuel du fonctionnement du *lobby*.

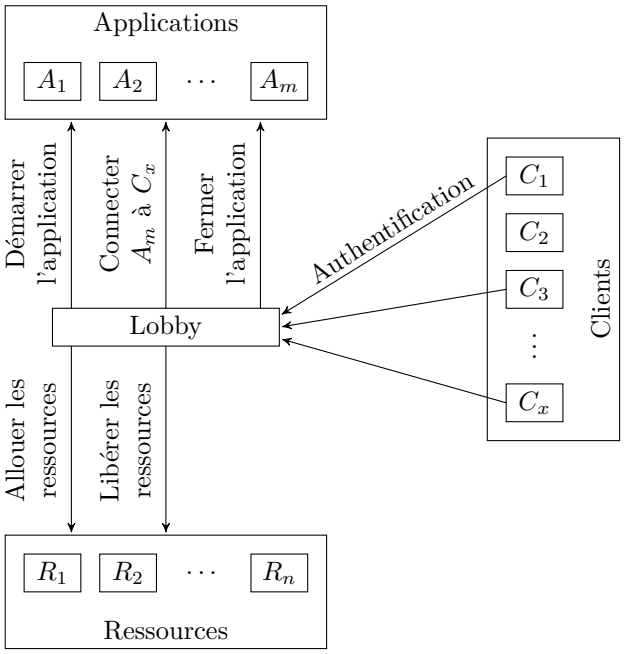


FIGURE 6.4 – Fonctionnement du *lobby*, image extraite de [143].

12. *Internet Protocol*

Le client MPEG-4

Le troisième composant de l'architecture Kusanagi est le client MPEG-4, il s'agit de GPAC [111]. Par l'intermédiaire du standard MPEG-4, ce client autorise la gestion de sessions multimédias interactives pour n'importe quelle application graphique (voir 4.1.3). GPAC prend en charge un nombre important de CODEC audio et vidéo et de conteneurs, la lecture pouvant s'effectuer localement, ou à distance (par le réseau). La réception des médias diffusés est réalisée grâce à la suite de protocole RTP/RTCP [13] qui elle-même s'appuie sur RTSP [14]. Le protocole de la couche transport peut être modifié, UDP et TCP étant tous deux supportés. GPAC est un lecteur multimédia *open source*, il fonctionne sur la plupart des plates-formes (Windows, Linux, Mac OS, Android et IOS). Le logiciel est disponible sous la forme d'un fichier exécutable, mais aussi en tant qu'extension pour inclure du contenu directement dans une page Web.

Optimisations apportées

Néanmoins et contrairement aux autres solutions proposées dans la littérature, un certain nombre d'optimisations ont été apportées au niveau de l'encodage vidéo pour permettre une réduction de la bande passante nécessaire, mais aussi pour maintenir le fonctionnement de la chaîne de rendu lors d'une fluctuation des caractéristiques du réseau (bande passante, latence, gigue...).

Dans [143] une adaptation en temps réel est réalisée en fonction d'une métrique, calculée grâce aux informations circulant sur le réseau. Des mesures de latence sont réalisées avant le lancement du système, mais aussi périodiquement tout au long de la diffusion. Ces mesures ont pour but de déterminer la latence introduite par la diffusion du flux vidéo lui-même. L'évolution de cette métrique permet de modifier la valeur du paramètre de quantification (Qp dans la plupart des implémentations) et ainsi d'éviter la congestion. Le but recherché est de garantir la jouabilité au niveau de l'utilisateur : il est préférable de dégrader un peu la vidéo en maintenant une certaine fluidité plutôt que de subir une véritable coupure. Dans [142] les auteurs prennent pour parti de considérer que les zones lointaines sont moins importantes que les zones proches (de la caméra virtuelle) du point de vue de l'utilisateur. Comme l'illustre la figure 6.5 et pour chaque image à encoder, la carte de profondeur est extraite du moteur de rendu, puis simplifiée en six niveaux d'importance. En fonction de ce niveau, chaque macrobloc est compressé avec un paramètre de quantification adapté. Les macroblocs considérés comme importants sont traités avec une valeur de Qp plus faible (en conservant une meilleure qualité) que leurs homologues estimés moins importants. Cette approche semble être mieux adaptée pour les jeux se déroulant dans un environnement extérieur, c'est-à-dire pour les scènes disposant d'une grande amplitude par rapport aux valeurs de leur carte de profondeur. Dans le cas de jeux se situant à l'intérieur, l'amplitude des valeurs de la carte de profondeur étant bien plus faible, il est plus complexe de prendre une décision quant aux macroblocs à considérer comme plus ou moins importants. On peut tout de même mettre en défaut cette approche, notamment lorsqu'un joueur tente d'éliminer des ennemis placés loin de sa position. Dans ce cas, bien que le joueur soit réellement focalisé sur ces ennemis, les données issues de la

13. *Real-time Transport Control Protocol*
14. *Real-Time Streaming Protocol*

carte de profondeur les situeront loin, les macroblocs correspondant seront donc compressés avec une valeur de Qp plus importante. Ce cas de figure dégrade donc la qualité dans une zone importante de l'image.

(a) Image vue par le joueur.

(b) Carte de profondeur correspondante, quantifiée sur six niveaux.

FIGURE 6.5 – Image de jeu avec sa carte de profondeur associée, images extraites de [142].

Bien qu'apportant un certain nombre de nouveautés, la solution proposée par Kusanagi n'est pas adaptée au marché actuel. En effet, l'architecture de base est bâtie autour d'une approche client/serveur relativement simple. De ce fait, un nombre très limité d'utilisateurs (typiquement un ou deux joueurs seulement) peut être pris en charge par un même serveur physique. Cette limitation engendre un coût matériel important et il est nécessaire de maximiser le nombre d'utilisateurs sur chaque nœud de calcul. La solution est dans l'adoption d'un modèle *cloud* qui permettra une utilisation plus souple et de bien meilleures performances. Le fait d'avoir un grand nombre d'utilisateurs par nœud implique autant de flux vidéos sortants et il est donc nécessaire d'essayer de limiter la bande passante dévolue à cet usage.

6.2 Architecture *cloud* de Kusanagi : XLcloud

Le projet XLcloud se situe à l'intersection de cinq domaines technologiques :

1. La virtualisation ;
2. Le *cloud computing* ;
3. Le *remote rendering* (voir chapitre 5) ;
4. La simulation interactive ;
5. Le *grid computing*.

Les technologies de virtualisation ont traditionnellement été perçues comme un handicap pour les applications de calcul intensif. En effet, le fait d'intercaler un hyperviseur entre les applications et le matériel introduit forcément une dégradation des performances. Cette dégradation, présente sur des centaines, voire des milliers de nœuds pose un réel problème de performance pour les déploiements à grande échelle. Dans [119] on constate par exemple que le

partage des ressources physiques comme la mémoire cache entre machines virtuelles est de nature à diminuer la rapidité de traitement des tâches du fait d'interférences entre les processus. Cependant, depuis l'arrivée des familles de processeurs Intel® VT et AMD-V™, des efforts importants tant au niveau matériel [1, 82] que logiciel [97, 35] ont été accomplis. Ces apports sont tels que la dégradation des performances, notamment au niveau des entrées/sorties deviendrait acceptable comparée aux bénéfices qu'apporte la virtualisation comme la migration à chaud ou la création d'un instantané (*snapshot*) d'une machine virtuelle. Ces nouvelles possibilités permettent par exemple de remédier à une panne matérielle, de faire de la répartition de charge ou encore de créer un point de reprise. Mais le meilleur argument en faveur de la virtualisation pour le HPC [15] consiste en la possibilité de créer un paquet contenant l'environnement complet d'un utilisateur, c'est-à-dire le système d'exploitation, les applications, les librairies ainsi que les compilateurs. Un tel paquet est stocké sous la forme d'une image disque préinstallée et préconfigurée [131].

Le concept de *cloud computing* est à l'origine d'un marché en très forte croissance, principalement tiré par de grandes firmes américaines : Google, Amazon, IBM ou encore Microsoft. La principale innovation apportée par ce concept, par rapport à la notion classique de *datacenter*, est le recours massif à la virtualisation des services par rapport à l'infrastructure matérielle. Cette virtualisation permet en particulier l'automatisation complète de la mise en production des ressources de traitement et de stockage dans des délais de mise en oeuvre relativement courts. Il en résulte une agilité de provisionnement des ressources bien supérieure à celle jusque là permise par les processus traditionnels. Cette mise à disposition automatisée a rendu possible la notion de fourniture en libre service et à la demande des différentes ressources du *datacenter*, d'où la dénomination d'IaaS [16]. L'état de l'art des plates-formes IaaS concerne essentiellement la fourniture de deux types de ressources : l'espace de stockage et la puissance de calcul. Cependant et par analogie avec les architectures utilisées par les réseaux IP, on peut considérer que les architectures IaaS fonctionnent aujourd'hui en mode « *best-effort* » par rapport au service fourni. En effet, la puissance de calcul est partagée entre plusieurs utilisateurs (*multi-tenancy*) et est fournie sans réellement prendre en compte les besoins de performances des applications. L'élasticité par rapport aux besoins de calcul n'est alors que faiblement exploitée et n'intervient pas dans la prise en compte fine des contraintes liées par exemple à l'exécution d'applications temps réel. On peut donc envisager la définition de classes de services – comme pour la QoS [17] utilisée dans les réseaux – qui permettraient en particulier de définir des niveaux de priorité supérieurs pour les applications interactives. Des travaux [92, 118, 119] existent déjà dans cette direction. Ainsi, dans [119] les auteurs proposent un système nommé « *Q-Clouds* » qui permet d'assurer un niveau de performance minimum, indépendamment du niveau de charge. Cette solution introduit par ailleurs la notion de « *Q-States* » qui autorise la définition de niveaux de QoS additionnels, pour chaque application et en fonction de contraintes spécifiques.

Maintenir un bon niveau d'interactivité impose d'être capable de conserver une mise à jour des terminaux connectés à un rythme situé entre 30 et 60 images

15. *High-Performance Computing*
16. *Infrastructure-as-a-Service*
17. *Quality of Service*

par seconde en tenant compte des phénomènes les plus rapides. La gestion de la liaison entre le client et le serveur, ainsi qu'entre les serveurs eux-mêmes, doit garantir un comportement du réseau apte au maintien d'une telle fréquence.

6.3 Adaptation et adaptabilité

Quand on parle d'adaptation, il est nécessaire de faire la différence entre deux notions : l'adaptation avant que le service ne démarre, et l'adaptation au cours de la diffusion du service. La première permet par exemple d'adapter la vidéo à la taille de l'écran du terminal qui l'affiche. La seconde va se focaliser sur les modifications qui pourraient avoir lieu entre le serveur et le client, donc sur le réseau. Pour agir directement sur la taille du flux vidéo (on parle de débit binaire), on intervient traditionnellement sur trois paramètres :

1. La résolution spatiale : le nombre de pixels à traiter ;
2. La résolution temporelle : le nombre d'images par seconde dans la vidéo ;
3. La qualité : elle dépend en grande partie du paramètre de quantification Qp, qui peut lui-même être géré par différentes stratégies.

Dans la suite de ce chapitre, le terme **modalité** doit être interprété de la manière suivante : il s'agit d'une combinaison des trois paramètres sus-cités, chacun aboutissant à un résultat différent. Il est maintenant nécessaire de faire la différence entre une vidéo enregistrée et une vidéo diffusée en direct, ces deux usages ne permettant pas forcément l'utilisation des mêmes approches. On peut considérer quatre techniques différentes :

1. Générer plusieurs modalités d'un même fichier vidéo ;
2. Modifier les paramètres d'encodage à la volée ;
3. Compresser la vidéo grâce à l'utilisation d'un encodeur scalable ;
4. Découper la vidéo en (petits) segments et les compresser selon différentes modalités.

La première approche vise à produire autant de modalités nécessaires, en fonction notamment des résolutions spatiales des terminaux. Une même vidéo est donc présente plusieurs fois, compressée de n façons différentes, ce qui a un impact sur la capacité de stockage du système. Cette adaptation est plutôt mise en œuvre en début de lecture, le choix de la version à utiliser étant laissé à la discrétion de l'utilisateur, ou pris à défaut de manière automatique. Si on prend pour exemple les services de stockage de vidéos en ligne, la plupart d'entre eux (YouTube, Vimeo...) réalisent une adaptation au début, avant que l'utilisateur ne commence à visionner la vidéo, mais pas pendant la diffusion.

La deuxième solution, appliquée lors d'une diffusion, consiste à utiliser un même flux vidéo tout au long de la transmission. Les paramètres de compression, notamment le paramètre de quantification Qp sont modifiés au besoin. C'est cette approche qui a été mise en œuvre dans le projet Kusanagi (voir 6.1.2 pour plus de détail).

Le troisième point fait état d'un encodage vidéo scalable, plus connu sous l'acronyme SVC [18]. Il s'agit d'inclure dans le flux vidéo produit, différentes mo-

18. *Scalable Video Coding*

dalités d'encodage, celles-ci étant choisies avant le début de la compression. Une seule opération va donc permettre de créer plusieurs versions d'un même flux vidéo. La couche de base, c'est-à-dire la modalité utilisant les paramètres de compression les plus agressifs peut être raffinée par la réception des couches successives. Certaines contiennent des données permettant d'accroître le nombre d'images par secondes, d'autres agissent sur la qualité de la vidéo. Dans [45] une plate-forme d'adaptation en temps réel est mise en œuvre grâce à l'utilisation de l'extension SVC de l'encodeur MPEG-4 AVC.

Pour la diffusion des contenus produits par les trois premières approches, on peut considérer que les deux systèmes les plus utilisés sont la suite de protocole RTP/RTCP et les protocoles de diffusion basés sur HTTP comme HLS [19] ou DASH [20]. Le premier est utilisé depuis des années (la première RFC [21] datant de 1996) et permet le transport de n'importe quel type de données nécessitant le temps réel. Il utilise des ports spécifiques, qui peuvent être bloqués par bon nombre de pare-feu. Le premier protocole de diffusion utilisant HTTP a été mis en œuvre par Apple. Les avancées offertes sont notamment la capacité à traverser les pare-feu sans problème (en utilisant le port standard du Web, le port 80 ou le 443 pour la version chiffrée) ou encore la possibilité d'adapter le flux vidéo au cours de la diffusion. Quelques limitations (certains formats vidéos ne sont pas pris en compte) empêchent son utilisation par le plus grand nombre. Malgré les efforts d'Apple, HLS n'a pas été standardisé.

La dernière approche considère une vidéo comme une suite de segments, chaque segment étant indépendant et disponible sous différentes modalités. Le protocole DASH ou plus précisément MPEG-DASH est le standard du groupe MPEG actuel décrivant la diffusion de contenus multimédia à partir de serveurs HTTP standards. Tout comme le HLS, cette norme va permettre aux applications de diffusion de disposer d'un débit binaire adaptatif. La première partie a été publiée en avril 2012 sous le nom ISO/IEC $23009-1$ [85]. Elle présente la description des médias ainsi que les formats utilisables pour représenter les segments. Ce standard répond aux besoins de l'industrie de pouvoir fournir des éléments multimédias de façon segmentée et de manière dynamique en utilisant des requêtes HTTP partielles (méthode *GET*). La relation entre les segments est établie dans une description de la présentation multimédia (MPD [22]) qui est mise à la disposition du client avant le début réel du *streaming*. En utilisant la MPD, le client prévoit les requêtes HTTP pour obtenir les segments individuels nécessaires en utilisant soit le format ISO de base (ISOBMFF [23]), soit le flux de transport MPEG-2 (M2TS) [24]. La MPD définit un modèle de données hiérarchique comprenant des périodes, des ensembles d'adaptation et de représentation, ainsi que les segments actuels. Alors que les périodes permettent de séparer les éléments multimédias de manière temporelle, les ensembles d'adaptation et de représentation permettent de fournir différentes versions encodées d'une même portion de vidéo en jouant sur des paramètres tels que le débit binaire, la résolution spatiale, le nombre d'images par secondes, la langue... L'adaptation dynamique est décrite comme un changement entre les représen-

19. *HTTP Live Streaming*
20. *Dynamic Adaptive Streaming over HTTP*
21. *Requests For Comments*
22. *Media Presentation Description*
23. *ISO Base Media File Format*
24. MPEG-2 *Transport Stream*

tations fournies par les ensembles d'adaptation à des moments prédéfinis dans le temps. La manière dont ce changement est fait n'est pas décrite dans le standard et laisse donc le champ libre à différentes solutions.

Il est à noter que les exigences de faible latence – nécessaires dans les plates-formes de *cloud gaming* – ne sont pour l'instant pas abordées dans le standard. En fin de compte, la version courante de MPEG-DASH se concentre uniquement sur l'envoi et la réception de vidéos traditionnelles (audio/vidéo) et de leurs éventuels sous-titres associés.

6.4 Vers des modèles prenant en compte l'attention

Dans les sections précédentes, divers moyens d'optimiser la compacité d'un flux vidéo ont été passés en revue. Pour adapter la compression vidéo, certains se basent sur des métriques de congestion du réseau quand d'autres utilisent des considérations empiriques. Dans une application comme le *cloud gaming* un autre paramètre est pourtant à prendre en compte : le joueur lui-même. Il est donc intéressant d'analyser le comportement du joueur lors d'une session de jeu. Une fois que les données importantes aux yeux du joueur sont connues, il devient plus simple d'adapter la compression, en agissant par exemple sur des zones peu importantes. Ce type d'approche est relatif à la qualité perçue (plus communément appelée QoE [25]), cette métrique étant subjective. L'évaluation de la QoE est un sujet d'actualité dans le monde multimédia [42], et il a récemment été fait la démonstration que la qualité perçue est une donnée bien personnelle [109]. Dans [86] un *eye-tracker* est utilisé pour mettre en lumière une corrélation entre la facilité d'utilisation d'un site Web et les données recueillies par cet *eye-tracker*.

6.4.1 Attention visuelle

Dans notre cas d'usage, l'AVU [26] est forcément une donnée importante, les paragraphes suivants lui sont dédiés. Le lecteur pourra au besoin utiliser l'annexe A qui rappelle les principales notions permettant d'appréhender le fonctionnement du système de vision humain.

À chaque fois qu'un être humain regarde le monde qui l'entoure, il se focalise consciemment ou inconsciemment sur une petite partie de la scène qu'il perçoit. En d'autres termes, une sélection perceptive de l'attention est utilisée. Dans la plupart des cas, cette action est réalisée par le déplacement des yeux, ciblant un endroit particulier dans le champ de vision. Il est néanmoins possible de déplacer son attention vers la zone périphérique du champ visuel sans mouvement oculaire. Le plus souvent, ces deux mécanismes sont d'ailleurs utilisés ensemble. Ce processus met en avant le changement d'attention : d'implicite elle devient manifeste. Par exemple, lorsqu'une personne regarde un paysage, l'attention implicite permet de détecter une forme ou un mouvement dans le champ visuel, la vision périphérique permet alors de deviner de quoi il s'agit.

25. *Quality of Experience*
26. Attention Visuelle de l'Utilisateur

Le regard se déplace ensuite à cet endroit pour permettre au cerveau d'accéder à des informations plus détaillées. Ainsi, un changement d'attention est souvent initié par l'attention implicite, qui va engendrer les mouvements oculaires nécessaires pour focaliser l'attention, qui devient alors manifeste.

Outre le fait d'avoir un domaine efficace de la vision très limité, les yeux sont également assez lents pour noter les changements dans les images si on se base sur la fréquence de rafraichissement des écrans actuels. Des études ont montré que la rétine a besoin d'environ 80 ms pour pouvoir voir une nouvelle image (dans des conditions d'éclairage normales). Cela ne signifie pas que la personne a pris conscience des changements, mais seulement que l'œil a perçu un changement. La capacité à enregistrer une image dépend également de l'intensité lumineuse de la scène. On peut comparer cela à la vitesse d'obturation d'un appareil photo dans un environnement sombre. Plus elle est courte, plus l'image obtenue est sombre et floue. En revanche, si une photo d'un objet bien éclairé est prise, même si la vitesse d'obturation est courte ce problème ne se reproduira pas. En plus du temps nécessaire pour capturer une image, l'œil exige également du temps pour que l'image disparaisse de la rétine. Ce phénomène de persistance dépend également de l'intensité lumineuse. Par exemple, si l'œil est exposé à une lumière très brillante comme un flash d'appareil photo, l'image du flash sur la rétine reste longtemps imprimée (on parle alors de persistance rétinienne). En plus de la sensibilité de l'œil à la lumière, la vitesse à laquelle nous percevons un objet dépend aussi de ce que nous observons [33, 34]. Par exemple, lors de la lecture dans des conditions d'éclairage normales, la plupart des personnes n'ont besoin que de 50 à 60 ms pour percevoir un mot. Cependant, dans le cas d'une image, plus de 150 ms sont nécessaires avant qu'une interprétation puisse être réalisée.

6.4.2 Application au *cloud gaming*

L'intérêt autour du *cloud gaming* ne cesse d'augmenter, beaucoup de chercheurs s'intéressent à améliorer les différentes parties du système. Certains s'intéressent à la qualité d'expérience perçue [101], d'autres à des méthodologies permettant de comparer la latence des différentes plates-formes [73]. Dans [64], un modèle conceptuel, le GAM [27] est introduit. Il estime l'importance des régions dans chaque image d'un jeu en fonction de l'attention que le joueur leur porte. Il s'agit d'un modèle représentant l'AVU, celui-ci permet de réduire le débit binaire lors de la diffusion de manière efficace. Pour toutes les images du jeu, le GAM estime l'importance de chaque macrobloc selon la perspective du joueur et réduit la bande passante allouée aux macroblocs jugés moins importants. Pour la conception du GAM, il est nécessaire de trouver des facteurs impactants le degré d'importance d'une région donnée dans une image. Un des facteurs les plus importants est la logique du jeu, il est donc nécessaire de considérer le genre du jeu et ses objectifs. Un autre facteur important est la façon dont le joueur regarde et prend en compte l'image. De manière globale, l'attention du joueur est donc influencée par le but recherché et les stimuli qu'il perçoit. Les auteurs de [64] qualifient ces deux processus de la perception humaine de *top-down* (ce que le joueur doit faire pour avancer dans le jeu) et *bottom-up* (couleur, intensité, textures) respectivement. Le GAM considère ces

27. *Game Attention Model*

deux types d'attention en utilisant les solutions proposées par [104] et [129] respectivement et combine leurs résultats pour définir une carte d'importance. De manière concise, le GAM fonctionne en quatre étapes, décrites ci-dessous :

1. Lors du rendu d'une image, la liste des objets ainsi que l'activité du joueur est envoyée au GAM (partie *top-down*, carte de priorité);
2. Lorsque le rendu d'une image est terminé, celle-ci est envoyée au GAM (partie *bottom-up*, carte de saillance) ;
3. La carte finale modélisant l'attention du joueur est générée en utilisant une résolution au macrobloc près ;
4. Les macroblocs de mêmes importance sont regroupés dans des *slices* et injectés dans l'encodeur qui utilise le FMO [28]. Chaque *slice* est compressée avec une valeur différente de Qp, en fonction se son importance.

(a) Une seule valeur de Qp.

(b) Trois valeurs de quantification différentes.

FIGURE 6.6 – Compression d'une image avec (6.6b) et sans (6.6a) l'utilisation du GAM, images extraites de [64].

La figure 6.6 présente une image avec et sans l'utilisation du GAM. Sans utilisation du GAM, l'image est entièrement traitée avec une valeur de Qp de 30. L'image encodée après application du GAM utilise trois valeurs de quantification ($30, 35$ et 40) respectivement utilisées pour les macroblocs importants, moyennement importants et peu importants. Dans l'image 6.6b S1, P1 et P2 représentent respectivement les régions importantes de la carte de saillance, et les régions d'importance moyenne et haute de la carte de priorité. Une étude subjective basée sur le standard [117] montre que l'intégration de ce modèle dans un système de *cloud gaming* permet de réduire le débit binaire d'environ 50% en moyenne, tout en conservant une bonne qualité d'expérience pour les utilisateurs. La qualité objective a baissé de 13% alors que la qualité subjective de seulement 4.75% (étude sur un panel de trente utilisateurs). Les auteurs mettent enfin en évidence que l'utilisation des deux cartes (saillance et priorité) permet d'obtenir un meilleur équilibre entre le débit binaire et la qualité subjective par rapport à l'utilisation d'une seule carte de priorité.

Le travail réalisé par [64] propose une avancée dans la compression du flux vidéo lors d'une session de *cloud gaming*, néanmoins la méthode utilisée pour déterminer où l'utilisateur regarde n'est pas franchement adaptée au cas du *cloud gaming* puisque la majorité des tests ont été réalisés sur des images sta-

28. *Flexible Macroblock Ordering*

tiques ou des vidéos. En effet, la façon dont l'utilisateur apprécie les images du jeu est totalement différente (par rapport à l'appréciation d'une image ou d'une vidéo). Lors d'une session de *cloud gaming*, le joueur est actif et prend part à l'action ; au contraire lorsqu'il regarde une vidéo, il est passif. Les auteurs ne font en outre aucune différence entre les différents types de jeux disponibles, et utilisent à chaque fois les mêmes modèles. On peut regretter l'absence d'une étude sur les performances qui aurait eu le mérite de quantifier l'impact dans la chaîne de rendu de l'utilisation du modèle proposé. Il est aussi nécessaire de prendre en compte le ressenti du joueur [110], chaque personne étant unique et possédant ses propres critères quant à la qualité d'expérience vécue.

C'est pourquoi, dans le cadre de ces travaux, nous avons pris le parti d'étudier pour différents types de jeux comment l'utilisateur se comportait et plus particulièrement où se focalisait son attention visuelle (l'AVU). Pour cela nous avons utilisé un *eye-tracker*.

6.5 Conclusion

Comme explicité dans [96], aucune des solutions ne paraît satisfaisante. Il n'existe pas de solution intégrée aux systèmes de *cloud gaming* pour répondre aux problèmes liés à l'adaptation. De plus, aucune approche ne tire parti des spécificités des images du jeu, pourtant accessibles au moyen du moteur de rendu. À notre connaissance, seul le projet Kusanagi prend en charge un certain nombre d'optimisations (encodage vidéo adapté, conditions du réseau) qui vont permettre une adaptabilité du système, et ce, de manière transparente. Il y a un point commun dans toutes ces solutions de *cloud gaming* : pour l'encodage comme pour l'adaptation, la vidéo est toujours considérée comme un ensemble de pixels. Toutes les mises en œuvre utilisent une version standard de MPEG-4 AVC et lorsqu'une adaptation est possible elle est toujours confinée au monde pixel.

Au vu des limitations mises en avant dans les paragraphes précédents, le but de ces travaux est de proposer une approche mixte dans le but de garantir une adaptabilité maximum, prenant en compte les conditions de transmission et les capacités de calcul du terminal client. La plate-forme existante, construite durant le projet Kusanagi, n'utilise actuellement qu'une représentation pixel 2D pour transmettre les données depuis le serveur vers le(s) client(s). Le point fondamental va être de considérer l'image générée par le jeu comme un mélange entre une représentation utilisant des pixels 2D et une représentation utilisant des primitives graphiques. L'introduction de ce nouveau levier aura pour conséquences de réduire la bande passante nécessaire, mais aussi d'augmenter l'étendue de la plage des débits binaires possibles, ce qui permettra d'atteindre un nombre plus large d'équipements et donc de clients. Pour parvenir à cela il sera cependant nécessaire de proposer une solution compatible avec les contraintes imposées par ces systèmes temps réel (temps de traitement, latence...).

Conclusion de la partie II

Les quatre chapitres précédents ont permis d'apprécier l'état de l'art dans les domaines nécessaires à la réalisation de ce travail de thèse, à savoir la compression des images et des vidéos, le rendu à distance associé à des applications 3D, les solutions de *cloud gaming* et enfin la notion d'attention sélective, basée sur l'AVU. On se propose de tenir compte des spécificités du contenu et des conditions d'utilisation pour contrôler la distribution des pertes d'informations au moyen d'une compression adaptée. De manière factuelle, il est possible de lister les différentes étapes nécessaires à la réalisation d'un encodeur multimodal ; elles sont au nombre de sept :

1. Estimer statistiquement pour chaque application 3D des régions d'intérêt ;
2. Mettre en évidence l'indépendance de ces régions d'intérêt pour une même application 3D et par rapport aux utilisateurs ;
3. Construire des cartes d'importance ;
4. Partitionner l'image par rapport à ces caractéristiques ;
5. Construire l'espace hybride ;
6. Compresser cet espace hybride ;
7. Transmettre l'espace hybride ;
8. Décoder et afficher le rendu de l'espace hybride.

La réalisation des objectifs présentés va permettre à ce travail d'aller au-delà de l'état de l'art. Une réponse est apportée à l'ensemble des points abordés ci-dessus dans la partie suivante, qui regroupe l'ensemble des contributions réalisées durant cette thèse.

Troisième partie

Contribution originale

Chapitre 7

Estimation des cartes d'attention

Sommaire

Pour être en mesure d'adapter au mieux l'encodage vidéo lors d'une session de *cloud gaming*, il est nécessaire de connaître les zones de l'image dans lesquelles le joueur focalise son attention. Une solution est l'utilisation d'un oculomètre (*eye-tracker* en anglais) qui va permettre de suivre les mouvements oculaires de l'utilisateur et ainsi de connaître, pour chaque image, la zone effectivement regardée. La perception visuelle humaine n'étant pas uniforme pour l'ensemble du champ visuel (voir 6.4), il sera ainsi possible d'adapter l'encodage vidéo sans pour autant dégrader la qualité perçue.

7.1 Application au *cloud gaming*

L'état de l'art, et plus particulièrement le chapitre 6, a permis de montrer que les applications de *cloud gaming* commencent tout juste à utiliser la notion d'attention pour proposer une adaptation de la compression. Au vu de leur limitations, nous proposons une technique permettant d'affiner les modèles d'AVU dans le but d'améliorer leurs performances.

L'objectif est dans un premier temps de déterminer les zones regardées en fonction du jeu et de l'utilisateur. Pour cela, des modifications ont été apportées à la plate-forme Kusanagi dans le but de lui permettre de gérer un *eye-tracker*, permettant ainsi lors de l'affichage de chaque image à l'écran, d'évaluer et d'enregistrer les données relatives à la position de chaque œil. Pour réaliser des tests, deux jeux sont disponibles : Doom3 et 0 A.D.. Le premier est un jeu de tir en vue subjective (FPS [1]), quant au second il s'agit d'un jeu de stratégie en temps réel (RTS [2]). La première phase de test est réalisée en utilisant une compression vidéo classique (aucune adaptation, Qp constant) pour être certain que l'attention visuelle du joueur n'est pas perturbée. Après un étalonnage du dispositif d'*eye-tracking*, chaque testeur joue pendant un temps donné (sessions de 20 minutes). Pour conserver une cohérence, tous les tests sont réalisés dans la même pièce, avec le même environnement. Notre panel de testeurs est constitué de cinq joueurs avertis, qui ont tous réalisé trois sessions de tests avec les différents jeux proposés. La suite de ce chapitre utilise les notations suivantes :

a. S0 représente la session 0 (le premier test d'un joueur) ;
b. T0 représente le testeur 0.

L'association des deux notations présentées ci-dessus permet d'identifier chaque test effectué par les joueurs, ainsi T2S1 représente le deuxième test effectué par le troisième joueur.

7.2 Métriques mises en œuvre

L'analyse des données issues des différents tests rend nécessaire l'utilisation de métriques adéquates permettant de déterminer si une corrélation est présente, mais aussi d'en évaluer le degré. On peut distinguer deux points de vue : estimer la similarité ou au contraire analyser la divergence. Nous avons choisi d'utiliser une métrique issue de chaque approche, respectivement le coefficient

1. *First-Person Shooter*
2. *Real-Time Strategy*

de corrélation de Pearson [50, 40] et la divergence de Jensen-Shannon (il s'agit d'une version améliorée de la divergence de Kullback–Leibler), basée sur l'entropie de Shannon [37]. Le calcul du coefficient de corrélation r_p est limité à la comparaison de deux variables numériques, ce qui n'est pas le cas de la divergence de Jensen-Shannon D^{π}_{JS} qui peut être généralisée à n distributions, comme explicité par [22]. Les méthodes de calcul de ces deux métriques sont respectivement rappelées par les équations 7.1 et 7.2. La divergence de Jensen-Shannon D^{π}_{JS} n'est pas utilisée telle quelle ; une métrique notée d^{π}_{JS} basée sur celle-ci est utilisée à la place, elle est présentée par l'équation 7.3.

$$r_p = \frac{\sum_{i=1}^{N}(x_i - \bar{x}) \cdot (y_i - \bar{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2} \cdot \sqrt{\sum_{i=1}^{N}(y_i - \bar{y})^2}} \tag{7.1}$$

avec :

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \quad \text{et} \quad \bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$$

$$D^{\pi}_{JS}(f_1, \ldots, f_M) = H\left(\sum_{m=1}^{M} \pi_m y^m\right) - \sum_{m=1}^{M} \pi_m H(y^m) \tag{7.2}$$

$$d^{\pi}_{JS}(f_1, \ldots, f_M) = \sqrt{D^{\pi}_{JS}(f_1, \ldots, f_M)} \tag{7.3}$$

avec :

$\pi_m > 0, (m = 1, \ldots, M), \sum_{m=1}^{M} \pi_m = 1, y^m \sim f_m (m = 1, \ldots, M)$

$f_1, \ldots, f_M$: M distributions de probabilités ;

H : l'entropie de Shannon ;

π_m : un coefficient appliqué à chaque distribution.

Pour rappel, les valeurs issues du calcul du coefficient de corrélation sont bornées entre -1 et $1 : -1 \leq r_p \leq 1$. Si $r_p = 0$, les variables ne sont pas corrélées. En revanche, la corrélation entre deux variables devient de plus en forte à mesure que r_p s'approche des valeurs extrêmes -1 ou 1. En ce qui concerne la distance de Jensen-Shannon, les valeurs s'échelonnent de 0 à $\sqrt{ln(2)} : 0 \leq d^{\pi}_{JS} \leq \sqrt{ln(2)}$. Si $d^{\pi}_{JS} = 0$, la divergence est nulle, ce qui implique une forte corrélation.

7.3 Présentation des jeux utilisés

Le fonctionnement et les interfaces des deux jeux cités précédemment sont présentés et décrits dans les deux paragraphes suivants.

7.3.1 Jeu Doom3

La figure 7.1 est une image représentative de l'environnement de jeu de Doom3. Comme dans presque tous les FPS, outre l'environnement graphique du niveau, on retrouve au premier plan trois éléments essentiels énumérés ci-dessous :

1. Le réticule au centre de l'écran permettant la visée ;
2. L'état des munitions, en bas à droite de l'écran ;
3. Le niveau de vie du joueur, situé en bas à gauche.

Le joueur n'a pas une vue globale du jeu ni du niveau en cours. Il se déplace et découvre au fil du scénario les différents environnements proposés.

FIGURE 7.1 – Capture d'écran lors d'une session de Doom3.

7.3.2 Jeu 0 A.D.

La figure 7.2 présente l'environnement de jeu proposé par 0 A.D.. Les jeux de type RTS sont très différents des FPS, le réticule de visée n'est par exemple pas nécessaire. En revanche on retrouve un système de gestion des personnages et des actions, situé en bas et au centre de l'image. Le joueur a une vue globale du niveau et peut se déplacer là où il le souhaite.

7.4 Analyse pour un jeu et un testeur donnés

Dans un premier temps, on se focalise sur les données enregistrées pour un jeu et un testeur en particulier. Il s'agit de vérifier que pour un jeu donné le testeur réagit de la même manière, mais aussi de mettre en avant le fait que le type de jeu influe grandement sur l'AVU. Chaque jeu fait l'objet d'un paragraphe détaillant les résultats obtenus au moyen d'un exemple représentatif. Les données relatives à l'ensemble des tests menés par les différents joueurs sont présentes en annexe. Les cartes de chaleur (*heat maps*) obtenues pour chaque joueur (pour chaque jeu et chaque session de test) sont exposées dans l'annexe B.1 via les figures B.1, B.2, B.3, B.4 et B.5. Quant aux différentes métriques évaluées, elles

FIGURE 7.2 – Capture d'écran lors d'une session de 0 A.D..

sont rassemblées dans l'annexe B.3 au moyen des tableaux B.1 et B.2. Pour autoriser une comparaison entre les deux métriques utilisées, on considère les cartes de chaleur de manière probabiliste : chaque valeur représente la probabilité pour un macrobloc d'être regardé par le joueur au cours du temps.

7.4.1 Jeu Doom3

Les résultats présentés ci-dessous ont été obtenus avec le jeu Doom3 et le testeur 2, ils sont représentés sous la forme de cartes de chaleur. L'unité sur les axes x et y est le macrobloc (16×16 pixels). Sur l'axe z on comptabilise le nombre de fois où l'utilisateur a regardé un macrobloc particulier de l'image. Pour permettre une comparaison plus aisée, le nombre d'occurrences est normalisé. Ainsi la figure 7.3 fait état des macroblocs les plus souvent regardés par les deux yeux.

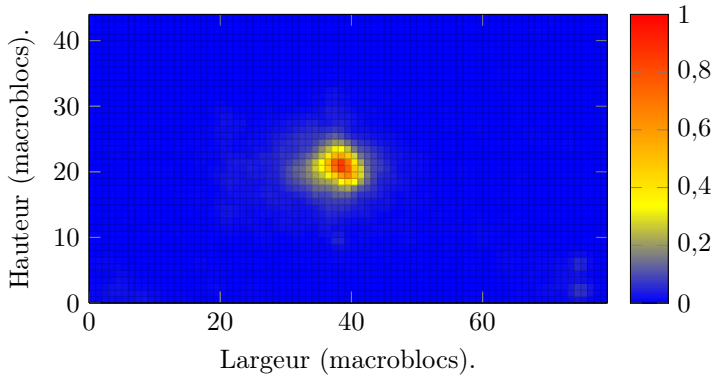


FIGURE 7.3 – Nombre d'occurrences (normalisé) enregistrées en fonction du macrobloc lors d'une partie de Doom3 par le testeur 2 (données issues des deux yeux et du dernier test unitaire).

Lors de ce test, on voit nettement que le joueur est focalisé sur le centre

de son écran. Ce résultat semble à posteriori assez logique pour ce type de jeu (FPS) puisqu'il est nécessaire de placer les ennemis dans le réticule de son arme pour avoir une chance de les éliminer. La carte de chaleur sommant les données issues des deux yeux (figure 7.3) présente un pic sous la forme d'une gaussienne presque parfaite, au centre de l'image. Au moyen du tableau B.1 on voit rapidement une forte corrélation entre les différents tests pour un même joueur (les tests entre joueurs sont présentés dans les sections suivantes). La figure 7.4 le montre visuellement : aucune valeur n'est inférieure à $0,84$, ce qui implique une forte corrélation entre les différents tests de chaque joueur.

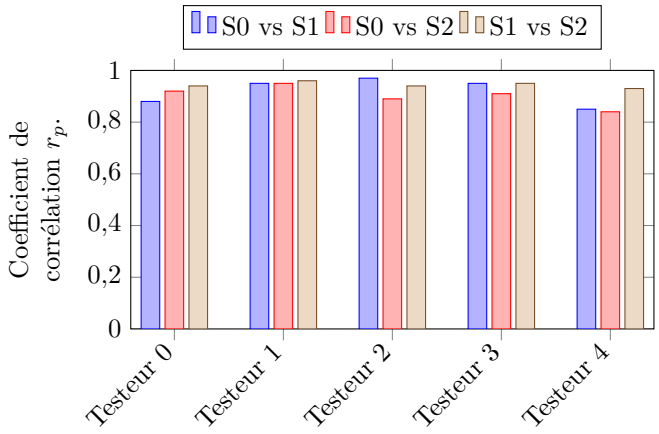


FIGURE 7.4 – Étude de la corrélation entre les trois tests effectués par chaque joueur sur Doom3 à l'aide du coefficient de corrélation.

7.4.2 Jeu 0 A.D.

La figure 7.5 présente le résultat obtenu par le testeur 2. On constate immédiatement que le résultat est bien différent de celui obtenu pour le jeu précédent. Cette fois les zones les plus souvent regardées sont plus dispersées, mais on peut tout de même noter que certaines parties de l'image ne sont presque jamais directement regardées par le joueur.

Comme pour le précédent jeu, le tableau B.2 détaille les métriques obtenues. Par le biais de la figure 7.6 qui expose visuellement les résultats, on voit immédiatement que le degré de corrélation est moins important que celui obtenu sur les tests effectués avec Doom3. Néanmoins, les valeurs sont toujours supérieures à $0,57$; ce qui permet d'affirmer que les données sont là encore corrélées.

7.5 Analyse en fonction du jeu

Après avoir mis en avant le fait que les résultats des tests permettent d'affirmer qu'une corrélation est possible (pour un joueur et un test donnés), on s'intéresse maintenant à établir une corrélation éventuelle entre tous les joueurs. En d'autres termes, il s'agit de prouver une corrélation des résultats en fonction du jeu. Une approche similaire à celle utilisée dans la section précédente est

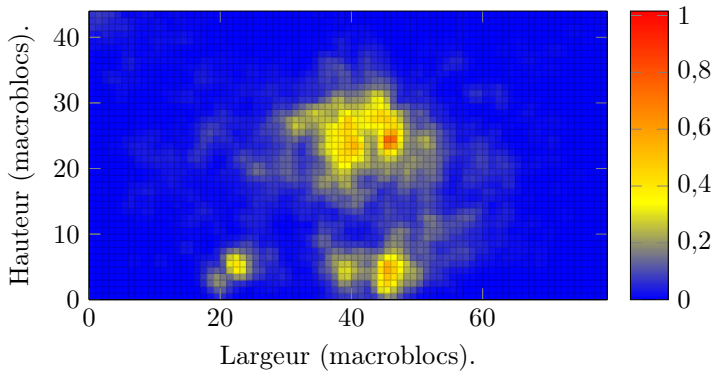


FIGURE 7.5 – Nombre d'occurrences (normalisé) enregistrées en fonction du macrobloc lors d'une partie de 0 A.D. par le testeur 2 (données issues des deux yeux et du dernier test unitaire).

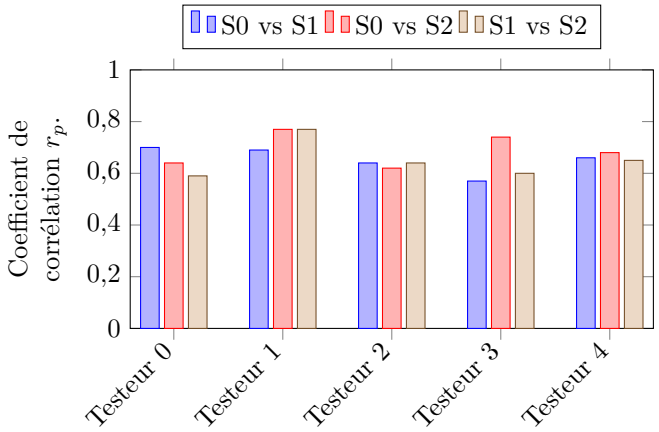


FIGURE 7.6 – Corrélation des différents tests réalisés par chaque joueur sur 0 A.D. à l'aide du coefficient de corrélation.

utilisée. Les cartes de chaleur utiles à cette section sont regroupées dans l'annexe B.2. La figure B.6 contient les cartes de chaleur moyennes obtenues par chaque testeur pour le jeu Doom3. La figure B.7 tient le même rôle, mais pour le jeu 0 A.D.. Comme précédemment, les données numériques sont regroupées dans les deux tableaux présents dans l'annexe B.3.

7.5.1 Jeu Doom3

En utilisant les données de tous les testeurs et de tous les tests réalisés avec le jeu Doom3, on obtient la carte de chaleur présentée par la figure 7.7. Encore une fois, on peut considérer que tous les joueurs regardent au centre de l'écran, en effet on retrouve une gaussienne presque parfaite.

Les valeurs des coefficients de corrélation calculées entre les différents testeurs sont données dans le tableau B.1. Le calcul est réalisé en utilisant la moyenne de tous les tests pour chaque joueur. On peut mettre facilement en

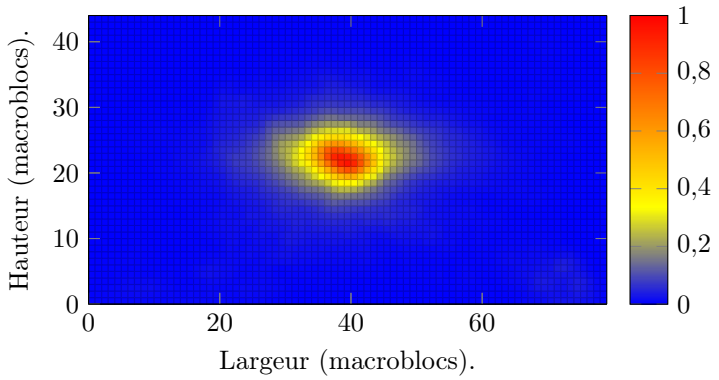


FIGURE 7.7 – Moyenne des zones regardées par l'ensemble des joueurs et pour tous les tests avec le jeu Doom3, notée $\bar{x}_{Doom3}$.

avant une forte corrélation, puisque la valeur la plus faible atteint $0,80$. La figure 7.8 offre un aperçu visuel du degré de corrélation. La valeur la plus faible est de $0,85$.

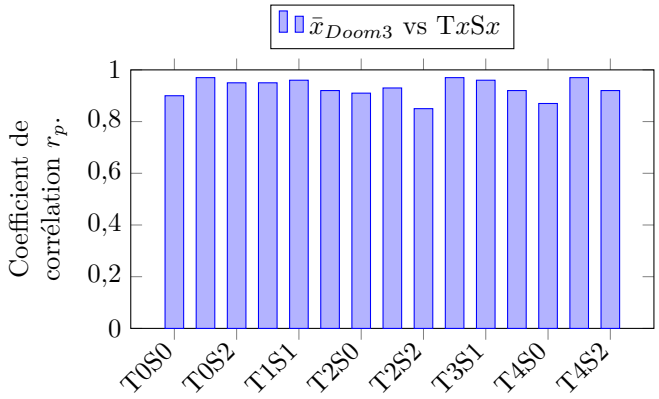


FIGURE 7.8 – Coefficient de corrélation de chaque test face à la moyenne globale pour le jeu Doom3.

7.5.2 Jeu 0 A.D.

De manière analogue la figure 7.9 présente la moyenne des zones regardées pour l'ensemble des tests réalisés. Le simple fait de regarder la figure B.7 permet là aussi de comprendre qu'une similarité entre les joueurs existe.

Les coefficients de corrélation calculés entre la moyenne globale et chacun des tests sont illustrés par la figure 7.10. Le degré de corrélation est moins important que celui obtenu lors des tests avec Doom3, mais une corrélation existe bel et bien.

La figure 7.11 expose les résultats obtenus avec le calcul de la distance Jensen-Shannon. Une divergence nulle étant représentée par la valeur 0, on retrouve des résultats similaires à ceux obtenus grâce au calcul du coefficient de

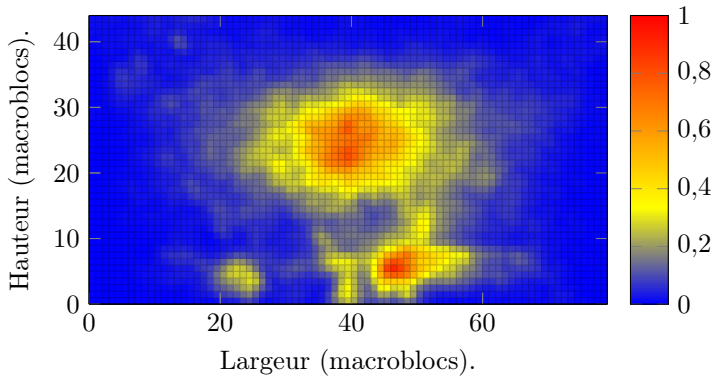


FIGURE 7.9 – Moyenne des zones regardées par l'ensemble des joueurs et pour tous les tests avec le jeu 0 A.D., notée $\bar{x}_{0A.D.}$.

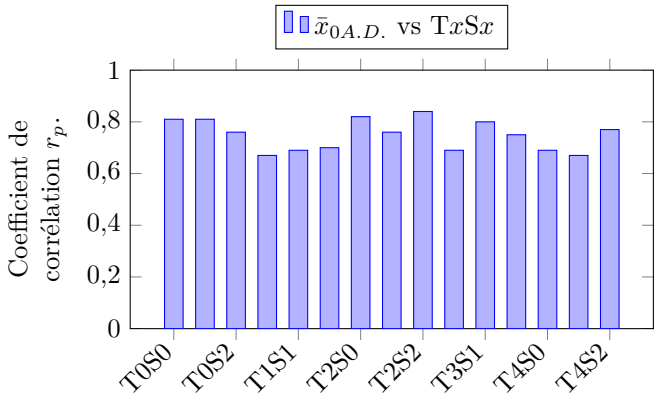


FIGURE 7.10 – Coefficient de corrélation de chaque test face à la moyenne globale pour le jeu 0 A.D..

corrélation. Le fait que les données issues de Doom3 ont un degré de corrélation plus important que celles obtenues avec 0 A.D. conforte la première analyse basée sur l'étude du coefficient de corrélation.

Les deux lignes verticales représentent la valeur de la distance de Jensen-Shannon obtenue en utilisant le calcul généralisé permettant de comparer la divergence de n distributions. Dans le cas de Doom3, la valeur obtenue est de $0,397$: ce qui permet de mettre en avant une divergence très faible des données. Pour 0 A.D., la valeur est de $0,548$.

Les résultats issus des tests réalisés dans ce chapitre permettent notamment de mettre en avant deux points importants :

1. Pour un jeu donné, le joueur focalise son regard sur un nombre limité de zones ;
2. La répartition des zones les plus souvent regardées dépend du jeu lui-même et non pas des utilisateurs.

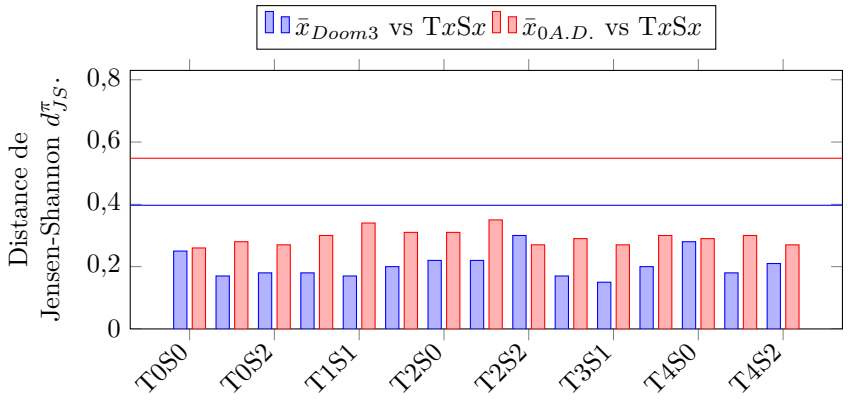


FIGURE 7.11 – Distance de Jensen-Shannon de chaque test face à la moyenne globale.

Dans le cas de l'utilisation du *cloud gaming*, ces données sont cruciales. En effet, un travail d'adaptation supplémentaire va pouvoir être entrepris en fonction des résultats expérimentaux obtenus. À travers une phase d'apprentissage, il est donc possible d'établir une carte illustrant les zones importantes d'un jeu. Cette même carte peut ensuite être utilisée lors de l'encodage vidéo pour adapter au mieux les modalités de compression. Cette approche apporte plusieurs avantages. Tout d'abord, aucun dispositif de mesure n'est nécessaire côté client (comme un *eye-tracker* par exemple) puisque la carte des zones d'importance est dépendante du jeu et non pas des joueurs. Ensuite, le fait même de faire varier les modalités de compression en diminuant la qualité dans les zones de moindre intérêt va permettre à la fois un gain en bande passante (débit binaire plus faible), mais aussi en terme de temps.

7.6 Conclusion

Grâce à plusieurs expérimentations, ce chapitre met en évidence que l'AVU est sélective. Ces différents tests, réalisés avec plusieurs joueurs sur deux jeux de nature différente (Doom3 et 0 A.D.) ont permis de prouver deux faits intéressants. Tout d'abord, le type de jeu utilisé est prépondérant dans la distribution des zones les plus regardées par le joueur. La deuxième conclusion met en évidence l'indépendance du joueur pour un jeu donné. C'est-à-dire qu'il est possible de mettre en avant une corrélation entre les cartes d'attention visuelle de tous les joueurs pour un jeu donné. Pour optimiser la compression d'un flux vidéo issu du *cloud gaming* il suffit donc de faire une étude avec un échantillon représentatif de joueurs pour permettre la modélisation d'une carte d'attention visuelle générique, carte qui sera utilisée pour agir sur le moteur de compression. Cette approche permet en outre de ne pas impacter (ou très peu) la qualité perçue en bout de chaîne.

Chapitre 8

Formalisation

Sommaire

Contrairement aux approches traditionnelles qui considèrent une vidéo comme un ensemble de pixels et utilisent un seul encodeur pour traiter l'image entière, notre approche est basée sur l'hypothèse qu'un partitionnement dynamique des images constituant la vidéo et que l'approximation de certaines zones avec une représentation paramétrique va permettre de réduire le débit binaire. Le découpage d'une vidéo en objets dans le but de les compresser de manière différente est l'idée de base de la norme MPEG-4 (voir 4.1.3), néanmoins la manière permettant de réaliser ce travail n'est pas explicitée par ce standard. Nous pensons que le fait de n'utiliser non pas un seul, mais plusieurs encodeurs va permettre d'atteindre un meilleur taux de compression tout en préservant la qualité en terme de PSNR. Nous présentons ici une nouvelle façon d'encoder un flux vidéo grâce à l'utilisation de **modalités**. Une modalité est un ensemble de zones compressées par un encodeur spécifique. Chacune d'entre elles correspond à une ou plusieurs parties de l'image, celles-ci étant particulièrement adaptées au système de compression d'un encodeur particulier. Avant d'aller plus loin, il est nécessaire de redéfinir certains termes, qui auront une signification propre dans ce manuscrit. Une approche mathématique est proposée pour une définition formelle des éléments nécessaires.

8.1 Transmodalité

Telle qu'elle est utilisée ici, la transmodalité a pour définition le fait d'utiliser plusieurs encodeurs vidéos dans la compression d'une seule et même vidéo, le but étant d'utiliser un encodeur adapté à chaque région de l'image à encoder. Une vidéo peut alors être encodée en utilisant différentes modalités. En d'autres termes, on définit une modalité comme étant la compression d'une classe par un encodeur. On définit une classe comme étant un ensemble de zones. Une classe est compressée par un encodeur spécifique. L'utilisation d'une ou plusieurs modalités n'est pas connue à l'avance, mais dépend du contenu même de la vidéo. La décision d'utiliser 1, 2 ou n modalités est prise en temps réel durant l'encodage multimodal, ce qui implique que les formes et les tailles des zones correspondantes sont potentiellement différentes pour chaque image. Un exemple est donné dans la figure 8.1. Cette image est composée de 5 zones qui sont regroupées selon 3 classes : il y a donc 3 modalités.

On considère qu'une image I (assimilée à un ensemble) peut-être découpée en un sous-ensemble non vide Z de p éléments, tel que :

$$Z = \{z_1, \ldots, z_p\} \tag{8.1}$$

avec :

$\forall k \in P = \{1, p\}, z_k \subseteq I$;

$\forall i, j \in P \times P, \text{avec } i \neq j, z_i \cap z_j = \emptyset$;

$Z - I = \emptyset$

Chaque z_k est une zone de l'image I. Une zone donnée est composée d'un ensemble contigu de pixels (voisinage V4) tel que défini ci-dessous :

$$\texttt{contigu}(i, j; k, l) = \delta_{ik}\delta_{jl-1} \oplus \delta_{ik}\delta_{jl+1} \oplus \delta_{jl}\delta_{ik-1} \oplus \delta_{jl}\delta_{ik+1} \oplus \delta_{ik}\delta_{jl} \tag{8.2}$$

avec :

$\texttt{contigu}(i,j;k,l) = \{i,j\} \times \{k,l\} \rightarrow [0,1]$

$i,k \in \{1,m\}, j,l \in \{1,n\}$

On définit un second sous-ensemble de I, nommé C, non vide et de taille r tel que :

$$C = \{c_1, \ldots, c_r\} \tag{8.3}$$

avec :

$r \leq p$;

$c_r = \{z_1, \ldots, z_n\}, 1 \leq n \leq p$;

$C - I = \emptyset$

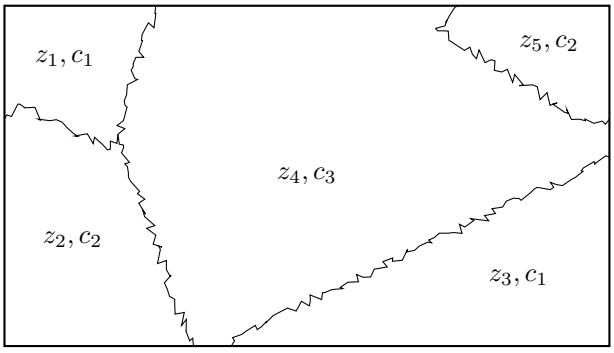


FIGURE 8.1 – Zones et modalités dans l'image.

À partir de l'équation 8.3 il est possible de définir le résultat I' de la compression multimodale d'une image I. Cette opération est qualifiée de transmodage. Dans l'équation 8.4 qui exprime l'opération de transmodage, f_i représente la fonction en charge de la compression de la classe c_i.

$$I'_{transmodé} = \bigcup_{i=1}^{n} f_i\,(c_i) \tag{8.4}$$

La décompression d'une image I' est notée I''. Pour obtenir I'', il suffit d'utiliser l'inverse de f_i qui n'est pas égal à f_i^{-1} du fait des pertes engendrées par la compression. Les erreurs introduites sont représentées par $\hat{f}$. L'équation 8.5 exprime mathématiquement ces mots.

$$I''_{transmodée\,inverse} = \bigcup_{i=1}^{n} g_i^{-1}\,(f_i\,(c_i)) \tag{8.5}$$

avec :

$g_i^{-1} = f_i^{-1} + \hat{f}$

Ainsi, la différence entre l'image I non compressée et sa représentation I'' après son rendu est donnée par l'équation 8.6, $\hat{I}$ étant l'erreur introduite par le processus de compression (et éventuellement de décompression) avec pertes (les erreurs introduites par toutes les modalités sont regroupées dans $\hat{I}$).

$$I'' = I + \hat{I} \tag{8.6}$$

8.2 Transmodeur *vs* transcodeur

Un transcodeur est un élément logiciel ou matériel qui a pour but de modifier la manière dont un fichier ou plus généralement un flux vidéo est compressé. De manière conventionnelle, un flux vidéo contient de fait au moins un flux vidéo et un flux audio, qui sont tous deux décompressés à l'aide d'un décodeur adapté. Dans la suite, on se focalisera uniquement sur la partie vidéo. Comme le montre la figure 8.2 un flux vidéo est présent en entrée comme en sortie.

FIGURE 8.2 – Vue d'ensemble d'un transcodeur.

Dans la figure 8.3 le principe de fonctionnement d'un transcodeur est mis en avant. Le flux vidéo en entrée est décodé grâce à l'utilisation d'un décodeur D approprié, puis réencodé en utilisant un nouvel encodeur E utilisant de nouveaux paramètres vidéos, mis à jour entre temps.

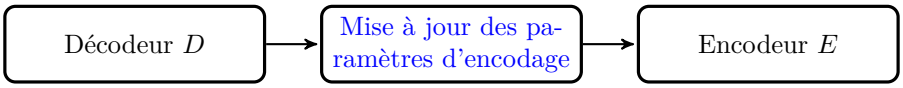


FIGURE 8.3 – Étapes principales réalisées par un transcodeur.

Outre le choix du nouvel encodeur à utiliser, on peut considérer que les principaux paramètres pouvant être modifiés entre le décodage et l'encodage sont les suivants :

a. La résolution spatiale S ;
b. La résolution temporelle T ;
c. La qualité Q, à travers la modification du paramètre de quantification la plupart du temps noté Qp.

Les valeurs de ces paramètres (S, T et Q) sont tout d'abord obtenues auprès du décodeur D. Elles sont – au gré de l'utilisateur – mises à jour vers leurs nouvelles valeurs respectives notées S', T' et Q' (voir la figure 8.4). L'encodeur utilise alors ces nouveaux paramètres pour compresser la vidéo. Il est à noter que les paramètres sont totalement indépendants les uns des autres. Il est donc possible de modifier tous les paramètres en même temps, ou de simplement agir sur l'un d'entre eux.

$$\begin{cases} T \to T', T' \leq T \\ S \to S', S' \leq S \\ Q \to Q', 0 \leq Qp \leq Qp_{max} \end{cases}$$

FIGURE 8.4 – Mise à jour des paramètres d'encodage.

Entre le décodage et l'encodage, les images sont temporairement stockées en mémoire dans un format non compressé permettant des opérations de traitement sur l'image. Dans ce travail, nous introduisons un nouveau paramètre ajustable que l'on nomme modalité. Nous définissons alors une nouvelle application appelée transmodeur. Il s'agit d'un transcodeur capable de gérer différentes modalités, comme décrit dans 8.1. Comparé à un transcodeur, notre transmodeur peut jouer sur un nouveau paramètre pour l'adaptation. Ceci va permettre d'améliorer la plage des débits binaires et ainsi rendre possible la distribution de vidéos à un nombre plus grand de personnes. Le transmodeur est aussi capable de réaliser des opérations de transcodage. À partir de notre définition, un transcodeur est donc un cas particulier d'un transmodeur. Si on se base sur la définition mathématique d'un transmodeur donnée par l'équation 8.4, un transcodeur est un transmodeur avec une seule classe, et par conséquent un seul encodeur (en résumé, une seule modalité). Ceci implique évidemment qu'une seule zone z_1 est considérée, qui couvre l'ensemble de l'image I. Le transcodeur est défini dans l'équation 8.7.

$$I'_{transcodé} = \bigcup_{i=1}^{1} f_i\left(c_i\right) = f\left(c\right) \tag{8.7}$$

Dans ce travail, nous nous sommes focalisés sur l'implémentation d'un transmodeur capable de gérer deux modalités. La suite de ce document présente donc une approche avec un transmodeur bimodal.

En utilisant les définitions d'un transmodeur et d'un transcodeur (respectivement les équations 8.4 et 8.7), l'hypothèse qu'un partitionnement dynamique des images constituant une vidéo en un certain nombre de zones va permettre une réduction du débit binaire peut être exprimée par l'inégalité présentée dans l'inéquation 8.8.

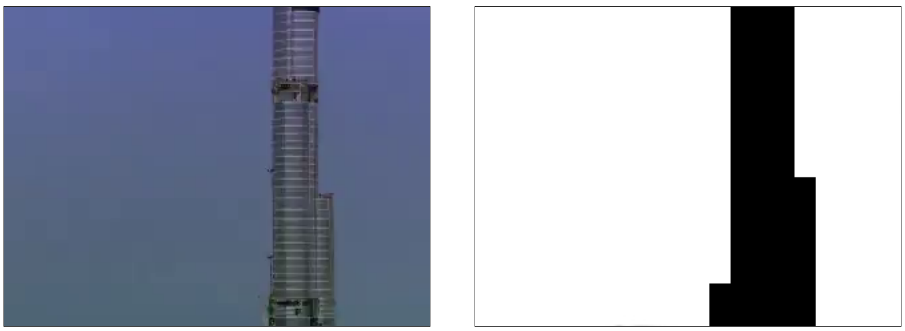
$$|I'_{transmodée}| \leq |I'_{transcodée}| \tag{8.8}$$

L'inégalité présentée dans l'inéquation 8.8 n'est pas forcément tout le temps vérifiée. En effet on essaie de démontrer que la taille d'une vidéo transmodée sera inférieure à son équivalent transcodé. C'est l'inéquation 8.9 qui reste donc à démontrer.

$$|V'_{transmodée}| \leq |V'_{transcodée}| \tag{8.9}$$

8.3 Exemple

Le processus complet de séparation des modalités est détaillé dans le chapitre suivant (voir 9). Pour illustrer le principe de fonctionnement d'un transmodeur, un exemple de découpage appliqué à une image simple est présenté dans la figure 8.5, qui est extraite du film Home [1].



(a) Image source. (b) Masque binaire possible.

FIGURE 8.5 – Masque binaire obtenu après un découpage basé sur un critère de détail (le liseré ne fait pas partie du masque).

Sur cette image, on peut immédiatement extraire deux types de modalités. En effet, si on choisit la quantité de détails comme critère prépondérant (beaucoup de contours et de couleurs), on peut considérer les deux modalités suivantes : une plutôt uniforme, lisse (c'est à dire la partie couverte par le ciel), et une autre qui comporte plus de détails (le gratte-ciel lui-même). L'image de droite (8.5b) dans la figure 8.5 présente un découpage possible, sous la forme d'un masque binaire.

L'idée de base est d'utiliser plusieurs (deux dans ce cas) encodeurs pour encoder une seule et même vidéo dans le but de démontrer que le poids (le débit binaire) du flux vidéo multimodal est réduit, tout en maintenant une qualité visuelle correcte. Le temps de traitement (décodage éventuel ou extraction, découpage & encodage) doit rester aussi faible que possible. Cette approche soulève plusieurs problèmes, notamment dans le choix des encodeurs, la précision du processus de séparation des modalités ou encore la façon avec laquelle sera écrit le flux vidéo en résultant. Dans les parties suivantes, nous nous focalisons sur le fonctionnement du transmodeur, et notamment sur les processus de découpage et d'encodage du flux vidéo multimodal.

8.4 Conclusion

Le but de ce chapitre est de présenter le concept de base servant de support au transmodeur. Les définitions décrivant son fonctionnement sont génériques et permettent l'utilisation de n modalités. Néanmoins, l'implémentation qui en a été faite – décrite ultérieurement au moyen d'un chapitre dédié – se limite dans une première version à l'utilisation de deux modalités. Le transmodeur

1. `http://home-2009.com/fr`

utilisera donc un maximum de deux encodeurs pour compresser une vidéo : un encodeur pixel, et un encodeur utilisant des primitives graphiques assorties de profils couleurs. Il est toutefois possible d'ajouter dans un second temps la gestion de nouvelles modalités. L'état de l'art a permis de recenser les différents encodeurs existants, tant dans le domaine classique du pixel que dans le domaine de la représentation vectorielle. Pour ce qui est de la compression pixel, le choix du H.264 a été fait. Il s'agit en effet du meilleur encodeur actuel, et des implémentations puissantes sont disponibles dans la communauté. Le choix de l'encodeur à base de primitives graphiques est plus délicat, une section relative à cette problématique est présente dans le chapitre 9.

Chapitre 9

Fonctionnement du transmodeur

Sommaire

Ce chapitre décrit en détail les différentes étapes nécessaires pour la création d'un flux vidéo multimodal ainsi que les différentes optimisations utilisées tout au long du processus. La vue d'ensemble du transmodeur présentée dans la figure 9.1 est presque identique à celle du transcodeur (voir la figure 8.2). L'image à traiter peut être obtenue grâce à l'utilisation d'un décodeur (première étape du transcodeur) ou de toute autre solution, à condition que l'image soit accessible dans un format connu. Peu importe la manière dont l'image brute a été obtenue, dans la suite de ce chapitre on considère posséder une image sous forme non compressée. Pour faciliter la compréhension des différentes étapes, des images issues d'un exemple réel illustrent chacune d'entre elles.

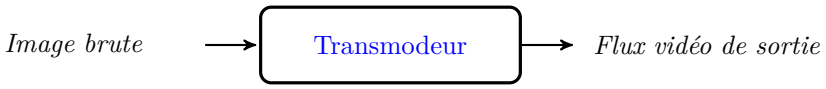


FIGURE 9.1 – Vue d'ensemble du transmodeur. Le schéma synoptique global est présenté dans la figure C.1.

Comme le montre la figure 9.2 le transmodeur est constitué de trois étapes distinctes : la construction des espaces, le codage de chaque espace par son encodeur associé et enfin la production d'un flux vidéo multimodal compressé grâce à un empaquetage adapté.

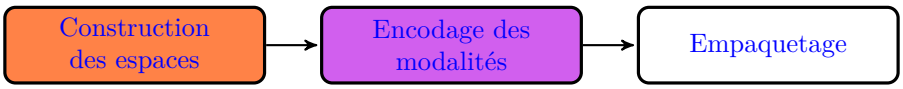


FIGURE 9.2 – Les trois étapes majeures du fonctionnement du transmodeur. Le schéma synoptique global est présenté dans la figure C.1.

Chacune de ces étapes fait l'objet d'une description détaillée dans les paragraphes suivants.

9.1 Construction des espaces

Cette étape a pour but de séparer chaque image en n modalités. Une image non compressée (dans le format RGB) est fournie à l'entrée du processus. La figure 9.3 montre que l'on peut diviser son fonctionnement en deux sous parties : la première opère au niveau de l'image et utilise des outils de traitement permettant d'obtenir au final un masque binaire. La seconde utilise des structures de données issues de l'analyse des images pour nettoyer et affiner les différentes zones constituant les modalités. On retrouve donc en sortie une combinaison de n modalités, dénommées m_1 à m_n.

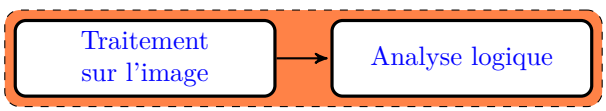


FIGURE 9.3 – Les deux sous étapes permettant la construction des espaces. Le schéma synoptique global est présenté dans la figure C.1.

Chacune de ces deux sous parties fait l'objet d'un paragraphe détaillant leur fonctionnement. Dans la suite du chapitre, on considérera uniquement deux modalités.

9.1.1 Traitement sur l'image

Cette étape est celle responsable de la séparation de l'image en modalités. Les différentes opérations successives de traitement de l'image sont décrites dans la figure 9.5. La figure 9.4 présente une image extraite d'un jeu, juste après qu'elle a été rendue. Celle-ci sera utilisée comme support pour la description du fonctionnement des étapes suivantes.

FIGURE 9.4 – Exemple d'une image extraite du jeu Doom3.

FIGURE 9.5 – Traitements au niveau de l'image. Le schéma synoptique global est présenté dans la figure C.1.

Chaque étape mentionnée dans la figure 9.5 fait l'objet d'un paragraphe qui détaille son fonctionnement.

Détection des détails

L'état de l'art a permis de montrer l'importance des détails constituant les images. Les expérimentations avec l'oculomètre ont quant a elles permis de mettre en avant les zones importantes aux yeux de l'utilisateur. Nous avons ainsi décidé d'utiliser un découpage de l'image en régions utilisant comme critères le niveau de détail, mais aussi un profil généré en fonction de chaque jeu. Dans une image, la détection des détails s'apparente à la recherche des contours, ceux-ci jouant le rôle de frontière entre les différents objets. Ce type de traitement est réalisé à l'aide de filtres. Il en existe plusieurs, les plus connus étant

les filtres de Canny, Laplace et Sobel. Pour des raisons de scalabilité, un algorithme offrant de bonnes performances de détection tout en minimisant le temps de calcul est nécessaire. À partir de nos besoins, et en prenant en compte les conclusions apportées par [94] et plus récemment [27], le filtre de Canny [8] semble la meilleure option. Ce choix est d'ailleurs confirmé par [49]. Leur approche de vectorisation utilise là encore ce filtre pour le traitement de vidéos. Prenons comme point de départ l'image présentée au début de ce chapitre (voir la figure 9.4). Après application du filtre de Canny, on obtient une image binaire, présentée dans la figure 9.6. Comme on peut le voir, les détails de l'image apparaissent en blanc alors que les zones plus uniformes (avec moins de détail donc) sont majoritairement composées de noir.

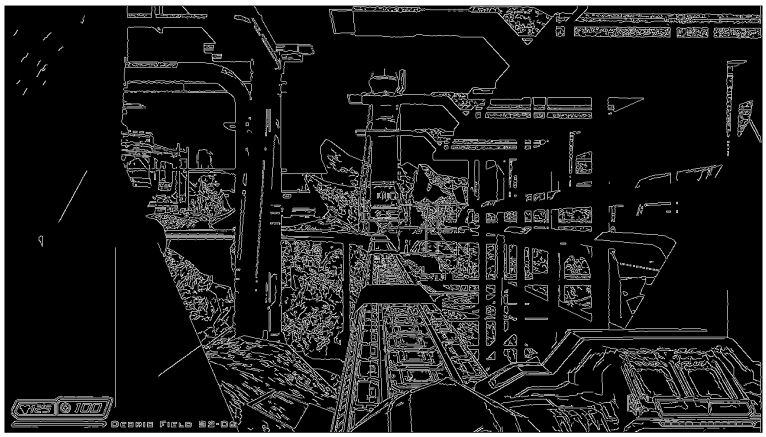


FIGURE 9.6 – Image après application du filtre de Canny.

Filtrage morphologique

Après l'application du filtre de Canny, quelques opérations morphologiques sont nécessaires pour nettoyer cette image binaire (suppression des points esseulés, des trous...). Diverses opérations comme des ouvertures (et des fermetures) sont réalisées pour pouvoir exploiter cette image brute. La conclusion de l'état de l'art à mis en avant notre choix d'utiliser la norme MPEG-4 AVC pour la compression d'une modalité. Cette décision implique de réaliser quelques adaptations sur le masque binaire, en effet MPEG-4 AVC utilise comme unité de base non pas le pixel, mais le macrobloc. Pour une efficacité maximale, il est donc nécessaire d'utiliser cette granularité lors du processus de séparation des modalités. Ceci implique le choix et l'application d'une règle pour associer chaque macrobloc à l'une ou l'autre des modalités. Cette opération est simplement réalisée de la manière suivante : si et seulement si tous les pixels appartenant à un macrobloc donné sont noirs, le macrobloc est noir, sinon il est blanc. En définitive, un masque comme celui présenté dans la figure 9.7 est généré.

Aucune opération d'encodage n'est réalisée dans cette étape. Les zones blanches et noires sont simplement candidates à la compression avec l'un ou l'autre des encodeurs. L'appartenance de chaque zone à une modalité n'est à cette étape pas définitive. Elle peut être remise en question par les étapes suivantes, notamment pour des raisons de qualité de représentation ou encore de compacité.

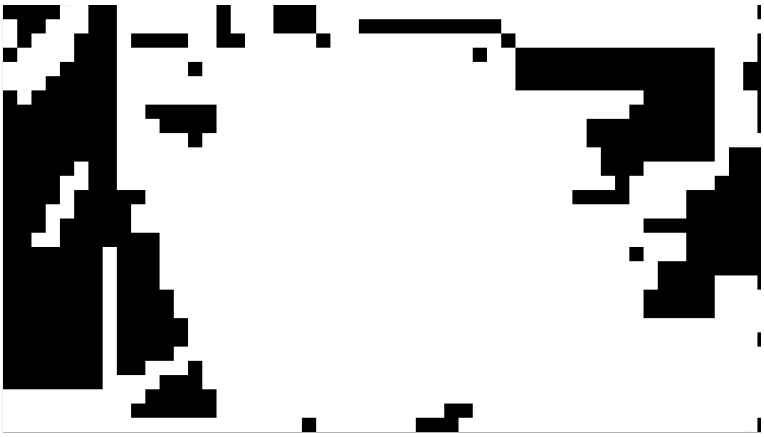


FIGURE 9.7 – Masque binaire correspondant (à la figure 9.6) utilisant le macro-bloc comme unité de base. Le liseré ne fait pas partie du masque et est juste présent pour délimiter les bords de l'image.

Étiquetage

La dernière opération de traitement de l'image est un étiquetage des zones candidates à une représentation paramétrique. Ce dernier est réalisé grâce à l'utilisation des voisins V4. Le nombre de zones, leur contour ainsi que leur surface sont alors connus. Une structure de données est renseignée, celle-ci permettra de manipuler facilement les zones dans la suite de l'algorithme. La figure 9.8 présente le résultat d'un tel étiquetage réalisé à partir de la figure 9.7.

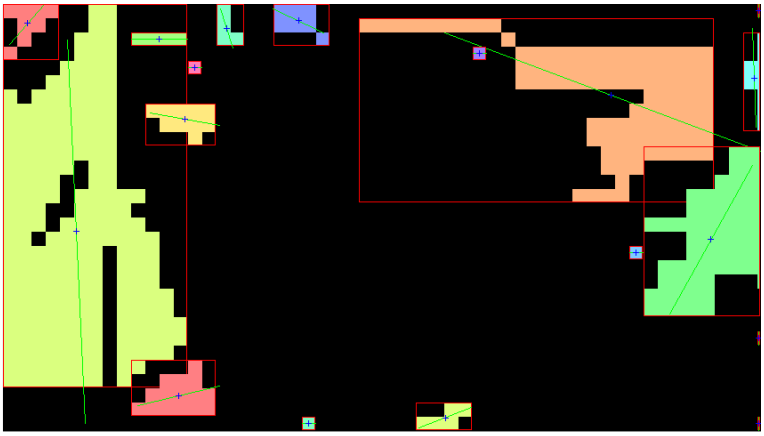


FIGURE 9.8 – Étiquetage des zones candidates à une représentation vectorielle : chaque étiquette est représentée par une couleur différente.

9.1.2 Analyse logique

Cette étape est importante dans le processus de transmodage car elle permet d'affiner le masque binaire précédemment créé. L'analyse logique se décompose

elle-même en trois sous étapes, présentées par la figure 9.9.

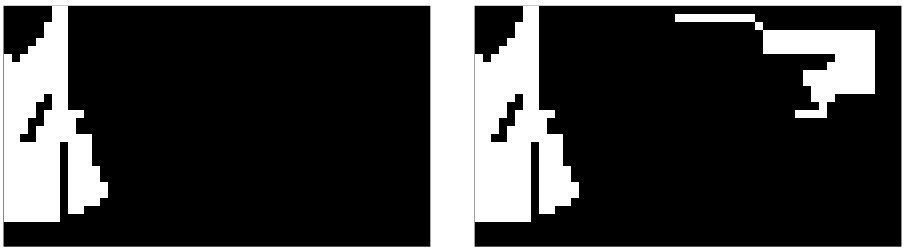
FIGURE 9.9 – Logique appliquée à la suite des opérations sur l'image. Le schéma synoptique global est présenté dans la figure C.1.

Réduction du nombre d'étiquettes

Cette première opération a pour but de supprimer les étiquettes les plus petites. En effet, la représentation paramétrique de petites zones n'est pas optimale, l'encodeur pixel étant dans ce cas bien plus performant. La décision menant à la suppression de chaque étiquette se base sur deux conditions :

1. La surface est inférieure à une surface de référence ;
2. La surface est en proportion (par rapport à la surface totale de l'image) inférieure à un seuil donné.

Les paramètres manipulant ces conditions possèdent des valeurs par défaut qui peuvent être modifiées par l'utilisateur. Évidemment, supprimer une étiquette qui était destinée à être représentée de manière paramétrique revient à redéfinir la modalité de la zone couverte, qui repasse donc dans l'espace pixel.



(a) Limitation à une étiquette.

(b) Limitation à deux étiquettes.

FIGURE 9.10 – Réduction du nombre d'étiquettes à traiter (en fonction de leur surface). Le liseré ne fait pas partie du masque et est juste présent pour délimiter les bords de l'image.

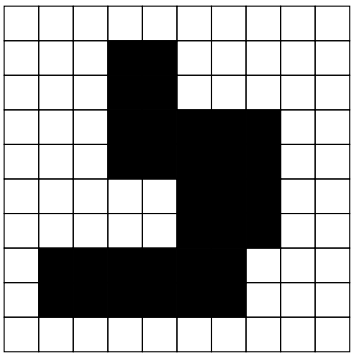
Décomposition en rectangles

Comme on peut le voir sur la figure 9.10, les paramètres utilisés ont permis de supprimer un certain nombre de zones candidates à une représentation alternative : seules deux subsistent. Les zones définies par les étiquettes sont de forme quelconque et peuvent contenir des trous. Au moment du décodage, lors de la reconstruction de l'image, il sera nécessaire de connaître deux choses :

1. La définition de chaque zone pour la seconde modalité ;

2. Les macroblocs sur lesquels s'appliquent le rendu de cette zone.

Il s'agit donc de trouver une solution efficace pour stocker un masque binaire, en utilisant une compression sans perte. À cela, il faut ajouter les choix que nous avons faits. Le premier exprime le fait que les deux encodeurs doivent pouvoir fonctionner indépendamment l'un par rapport à l'autre. Le deuxième que le flux vidéo produit soit compatible avec la norme MPEG-4 AVC et donc avec les implémentations courantes. Plusieurs solutions sont possibles pour répondre à ce problème, mais une solution assurant un maximum de compacité est à privilégier. Pour répondre à ce besoin, nous avons choisi de décomposer chaque étiquette en un ensemble composé de la liste des plus grands rectangles. La décomposition en rectangles d'une matrice binaire ou DBMR [1] est un problème à priori simple, mais qui ne l'est pas du tout en réalité. La figure 9.11 permet d'appréhender visuellement le problème.



(a) Polygone rectilinéaire dans une image binaire.

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(b) Décompostion minimale de la matrice binaire représentant l'image.

FIGURE 9.11 – Illustration d'une solution (9.11b) au problème DBMR à partir de l'image binaire 9.11a.

La solution optimale à ce problème possède une complexité en $\mathcal{O}(n^{\frac{3}{2}} \times log(n))$ lorsque l'objectif de l'optimisation est de minimiser le nombre de rectangles présents dans la décomposition [14]. Malheureusement, cet algorithme n'est pas approprié pour des applications requérant du quasi-temps réel tout en manipulant des matrices de taille moyenne à grande. C'est notamment le cas pour les applications de traitement vidéo qui n'ont que quelques dizaines de millisecondes pour traiter chaque image (images qui sont de plus en plus grandes : 2K [2], 4K [3]...).

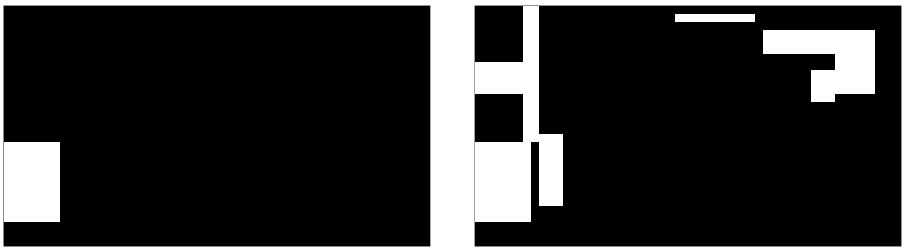
Une première approche a été de réaliser un algorithme testant toutes les possibilités (une solution *bruteforce*), pour que ce problème ne soit pas bloquant dans la suite du développement du prototype logiciel. Cette solution n'était pas du tout satisfaisante, le temps nécessaire à la décomposition était bien trop long (plusieurs centaines de millisecondes). Ainsi, dans un second temps, une

1. *Decomposing Binary Matrices into Rectangles*
2. Résolution spatiale de 2880 × 1620 pixels.
3. Résolution spatiale de 3840 × 2160 pixels.

version beaucoup plus efficace, basée sur la méthode IBR [4] [39] a été mise en œuvre. Néanmoins, cette deuxième approche n'apportant par entière satisfaction, une collaboration avec un chercheur de l'équipe, Julien Subercaze, a permis de mettre au point une nouvelle solution bien plus efficace. Il s'agit d'une heuristique linéaire en temps et en espace, qui permet d'améliorer encore les résultats. Cet algorithme dépasse le cadre de cette thèse, l'annexe D détaille néanmoins de façon concise le fonctionnement de cette nouvelle méthode qui a été nommée WSRM [5]. De plus, un article expliquant le fonctionnement de ce nouvel algorithme a été soumis [138].

Filtrage des rectangles

L'étape précédente a permis l'obtention d'une liste de rectangles permettant de couvrir l'ensemble d'une zone. Ce nouveau filtrage permet de réduire le nombre de rectangles à utiliser dans une zone particulière. Il permet de faire en sorte qu'un petit nombre de rectangles permette de couvrir une grande zone, et ce toujours dans le but d'améliorer la compacité. En effet, dans la plupart des cas et pour une zone donnée, une petite fraction de la liste des plus grands rectangles permet de représenter un pourcentage (en terme de surface) important de la zone à couvrir. Le choix des rectangles à garder est basé sur la même logique que lors du filtrage des étiquettes. Les parties de l'image couvertes par les rectangles supprimés sont alors repositionnées dans l'espace pixel. Les paramètres par défaut sont là encore modifiables par l'utilisateur. La figure 9.12 offre un aperçu du résultat obtenu en imposant un maximum de deux rectangles par zone.



(a) Limitation à un rectangle (une seule étiquette).

(b) Limitation à quatre rectangles (deux étiquettes).

FIGURE 9.12 – Réduction du nombre de rectangles définissant une zone. Le liseré ne fait pas partie du masque et est juste présent pour délimiter les bords de l'image.

Cette étape met un terme aux traitements permettant de construire un masque binaire définissant les zones à encoder pour chaque modalité. Néanmoins, l'appartenance de chaque rectangle à la modalité vecteur peut encore être remise en question lors le l'encodage (voir la fin de la section 9.3). En effet si lors de la compression le logiciel de paramétrisation ne parvient pas à produire un résultat satisfaisant pour une classe, la modalité d'encodage peut encore être modifiée.

4. *Image Block Representation*
5. *Walk Stack Remove Merge*

9.2 Mise en œuvre des cartes d'attention

Le chapitre 7 a permis de mettre en évidence que pour un jeu donné, toutes les zones de l'écran ne sont pas regardées uniformément au cours du temps. À partir de la figure 7.7, on peut considérer trois zones distinctes : une première que l'on qualifiera de très importante, au centre de l'image (dans le cas de Doom3), une deuxième bien moins importante (les zones les plus éloignées du centre) et enfin une troisième jouant le rôle de zone tampon entre les deux premières. La figure 9.13 expose de manière visuelle ces trois zones, respectivement colorées de rouge, de gris et d'orange.

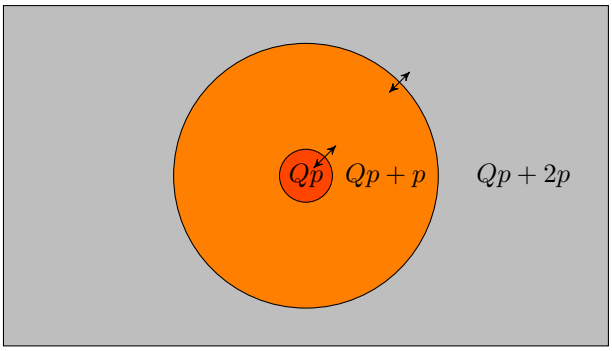


FIGURE 9.13 – Représentation des trois zones d'importance pour le jeu Doom3.

Comme observé dans le chapitre 7, la carte d'attention visuelle obtenue dans le cas du jeu Doom3 ressemble énormément à une fonction bien connue, en l'occurrence une fonction gaussienne en deux dimensions. Sa définition formelle est rappelée par l'équation 9.1 ;

$$f(x,y) = Ae^{-\left(\frac{(x-x_0)^2}{2\sigma_x^2}+\frac{(y-y_0)^2}{2\sigma_y^2}\right)} \tag{9.1}$$

Dans notre cas d'usage, il suffit d'adapter cette définition à la surface définie par une image. Si l'on considère la largeur w et la hauteur h, pour centrer la fonction gaussienne au centre de l'image, il s'agit de modifier les valeurs des variables x_0 et y_0 : il suffit d'utiliser $x_0 = w/2$ et $y_0 = h/2$. On considère l'amplitude maximale A comme égale à la valeur minimale de Qp utilisée pour la quantification de chaque macrobloc. L'étalement selon x et y peut être paramétré en jouant sur les valeurs σ_x et σ_y. L'analyse des résultats obtenus grâce à l'*eye-tracker* (présentés chapitre 7) permettent d'identifier les valeurs suivantes : $\sigma_x = 5$ et $\sigma_y = 5$. En résumé, la courbe gaussienne couvre maintenant l'ensemble de l'image, la valeur maximale étant située au centre de l'image, et les valeurs les plus faibles étant dans les coins (ils sont en effet les points de l'image les plus éloignés du centre de l'image).

Une fois la carte d'attention visuelle générique modélisée, il est alors nécessaire de créer son pendant représentant cette fois les trois niveaux de quantification à utiliser. Pour cela deux seuils sont nécessaires, ils permettent de créer les trois classes. Pour réaliser des tests, tant sur la bande passante que sur la qualité, le tableau 9.1 résume les différentes possibilités explorées.

TABLE 9.1 – Résumé des modèles de cartes d'attention visuelle utilisées pour Doom3.

N° de profil	Qp **minimal**	**Niveau** N_1	**Niveau** N_2	**Pas** p
0	25	10	20	1, 2 et 4
1	25	12	20	1, 2 et 4
2	25	12	22	1, 2 et 4
3	25	12	24	1, 2 et 4
4	25	15	20	1, 2 et 4

Chaque ligne du tableau 9.1 doit être interprétée de la manière suivante :

Si $25 \geq f(x,y) > N_2$, $Qp = 25$;
Si $N_2 \geq f(x,y) > N_1$, $Qp = 25 + p$;
Si $N_1 \geq f(x,y) \geq 0$, $Qp = 25 + 2p$.

Ainsi, une zone moins importante sera compressée en utilisant un degré de quantification plus élevé, réduisant ainsi la qualité d'encodage et du même coup le débit binaire. Les cinq modèles présentés dans le tableau 9.1 sont représentés de manière visuelle au moyen de l'annexe E. La figure E.1 expose les cinq modèles, à la résolution spatiale utilisée pour les tests, à savoir le 480p [6]. Évidemment, le raffinement du paramètre de quantification est réalisé au niveau du macrobloc, ce qui explique l'aspect crénelé des cartes ainsi crées. La mise en œuvre est réalisée au moyen d'une modification de l'option d'optimisation L. Au lieu de positionner la valeur du paramètre de quantification à $Qp = Qp_{max}$ pour les macroblocs voulus, une nouvelle valeur leur est assignée (les trois valeurs de Qp sont Qp, $Qp + 4$ et $Qp + 8$).

Pour appliquer ces différents profils lors du processus de transmodage, il est nécessaire de déterminer dans quelles zones s'appliqueront les différentes modalités. En toute logique, il a été décidé que la zone de moindre importance peut faire l'objet d'une compression multimodale, alors que les deux autres zones sont limitées à un encodage pixel. Ces deux dernières zones peuvent néanmoins utiliser une valeur de quantification différente. Ainsi, après application du profil 1 représenté par la figure 9.15 sur le masque binaire obtenu par le filtrage de Canny (figure 9.6) un nouveau masque est déterminé, présenté par la figure 9.14.

9.3 Encodage des modalités

L'étape permettant la séparation des modalités a été détaillée dans les paragraphes précédents. Les opérations d'encodage respectif des deux modalités peuvent maintenant être réalisées. L'encodage pixel ne sera pas décrit ici, puisqu'il a déjà fait l'objet d'une explication détaillée dans l'état de l'art (voir le chapitre 4). Seul le processus de paramétrisation est décrit dans la suite de ce paragraphe.

Comme nous l'avons précédemment évoqué, la vectorisation n'est pas une tâche évidente, tout particulièrement quand l'utilisateur veut l'utiliser sur des

6. Résolution spatiale de 720×480 pixels.

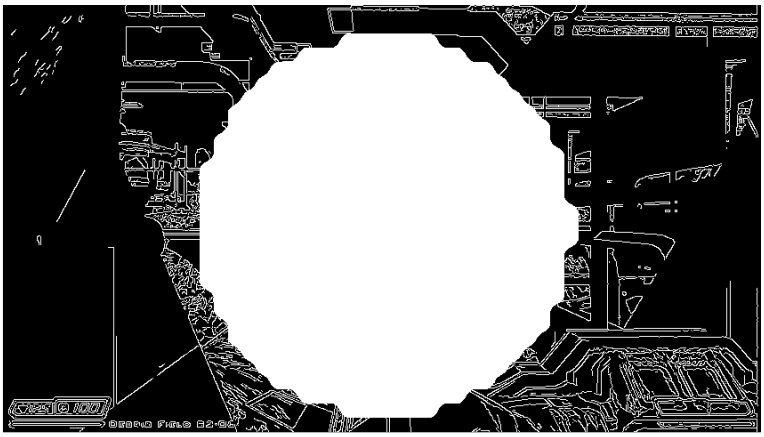


FIGURE 9.14 – Masque binaire après application du profil 1 et du filtre de Canny.

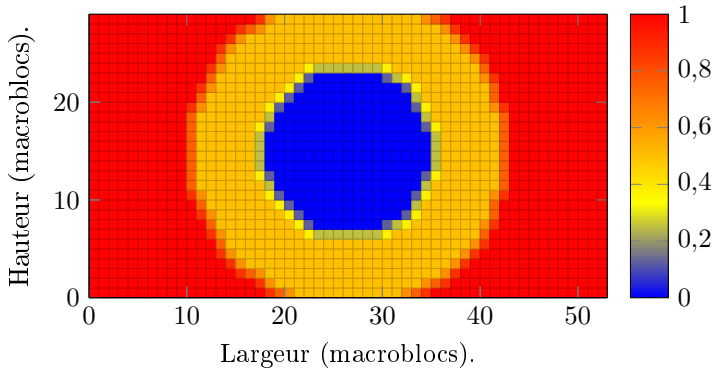


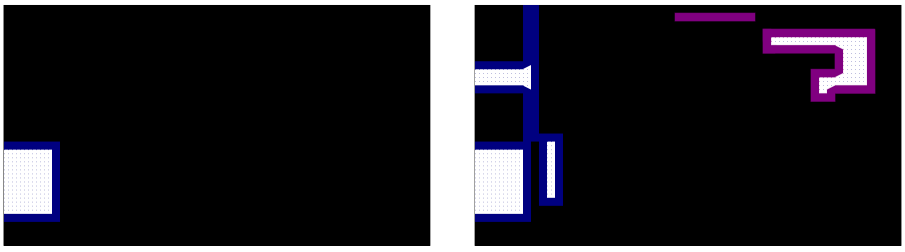
FIGURE 9.15 – Carte d'attention selon le profil 1.

images naturelles en espérant un bon résultat en qualité. Les deux problèmes les plus importants dans le processus de vectorisation sont la complexité du procédé et le temps nécessaire à son achèvement. Ces deux aspects sont de plus intimement liés au contenu même de l'image. Dans ce travail, nous voulons un logiciel de vectorisation qui réponde à nos besoins : un processus rapide et une qualité correcte (en fonction du PSNR). Toutes les étapes (précédentes) de traitement permettent de s'assurer que les textures qui seront fournies au module de vectorisation seront d'un certain type, à savoir plutôt uniforme. À partir de ce fait, il n'est pas nécessaire de concevoir un logiciel de vectorisation complexe. Une approche simple, de fait peu gourmande en ressources matérielles est possible. En outre, il faut prendre en compte le temps nécessaire à cette implémentation. Toutes ces remarques ont influencé notre choix pour l'implémentation d'une première version ; celle-ci est inspirée de la méthode utilisée dans [49] mais a été simplifiée. En résumé, les primitives graphiques utilisées sont des rectangles auxquels il faut adjoindre un profil couleur, dont la détermination est présentée ci-dessous. L'approche se décompose en plusieurs étapes :

a. Décomposer la zone de l'image concernée dans ces trois composantes Y,

Cr et Cb ;
b. Pour chaque composante, générer un ensemble de points d'échantillonnage ;
c. Calculer pour chaque composante l'interpolation polynomiale 3D.

La décomposition d'une zone à vectoriser dans ces trois composantes est simple. Pour chaque composante les valeurs sont stockées sur 8 bits, elles s'échelonnent donc de 0 à 255. L'échantillonnage n'est pas réalisé de manière uniforme dans la classe. Le nombre d'échantillons est plus important sur les contours de la classe, et est plus faible à l'intérieur. Cette approche a pour but de limiter les effets de bords lors de la reconstruction entre les zones compressées avec le premier encodeur et celles ayant été traitées par le second. Les paramètres d'échantillonnage sont modifiables et ont une influence non négligeable sur le temps d'exécution total, mais aussi sur la qualité obtenue. La figure 9.16 illustre visuellement les deux zones d'échantillonnage utilisées.



(a) Deux zones d'échantillonnage : la zone bleue et la zone blanche (une seule étiquette).

(b) Deux zones d'échantillonnage par étiquette : la zone violette et la zone blanche (deux étiquettes).

FIGURE 9.16 – Mise en évidence des différentes zones d'échantillonnage. Le liseré ne fait pas partie du masque et est juste présent pour délimiter les bords de l'image.

L'interpolation polynomiale 3D est calculée de manière matricielle, grâce à la méthode des moindres carrés. Le polynôme de base est celui proposé par [49]. Il est de la forme suivante : $z = a + bx + cy + dx^2 + exy + fy^2$. Les tailles n des matrices colonnes z, x et y sont donc identiques. M et Z sont connus et il s'agit de calculer K. De manière matricielle, les différents éléments s'expriment donc de la manière suivante :

$$M = \begin{pmatrix} 1 & x_0 & y_0 & {x_0}^2 & x_0 y_0 & {y_0}^2 \\ 1 & x_1 & y_1 & {x_1}^2 & x_1 y_1 & {y_1}^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n-1} & y_{n-1} & {x_{n-1}}^2 & x_{n-1} y_{n-1} & {y_{n-1}}^2 \end{pmatrix}, K = \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix}, Z = \begin{pmatrix} z_0 \\ z_1 \\ \vdots \\ z_{n-1} \end{pmatrix}$$

Pour calculer les coefficients a, b, c, d, e et f, il faut donc résoudre l'équa-

tion (9.2a). La solution est donnée par l'équation (9.2d).

$$M \times K = Z \tag{9.2a}$$
$$M' \times M \times K = M' \times Z \tag{9.2b}$$
$$(M' \times M)^{-1} \times M' \times M \times K = (M' \times M)^{-1} \times M' \times Z \tag{9.2c}$$
$$K = (M' \times M)^{-1} \times M' \times Z \tag{9.2d}$$

Pour permettre d'apprécier la différence entre l'image originale et son équivalent reconstruit, le coefficient de corrélation de Pearson est calculé. Pour cela, il est nécessaire de calculer l'estimation de Z, notée Z_e qui s'obtient simplement de la façon suivante : $Z_e = M \times K$. Le coefficient de corrélation r_p est ensuite évalué à partir de variance d'origine $V_{origine}$ et de la variance expliquée $V_{expliquée}$. Le détail du calcul est donné dans l'équation 9.3.

$$r_p = \frac{V_{expliquée}}{V_{origine}} \tag{9.3}$$

avec

$$V_{origine} = Var(Z) \quad \text{et} \quad V_{expliquée} = Var(Z_e)$$

Chaque zone à représenter de manière vectorielle est donc constituée de trois polynômes correspondant aux trois composantes ainsi que d'une suite de rectangles définissant la zone dans laquelle les appliquer. Leurs coefficients de corrélation respectifs permettent de contrôler la qualité de la modélisation. Si pour une zone donnée la qualité de la représentation vectorielle est trop faible, la décision de remettre cette aire dans l'espace pixel peut être prise. Cet arbitrage dépend de la valeur du coefficient de corrélation de la composante Y.

9.4 Optimisation de l'encodage du flux multimodal

Encoder une vidéo en utilisant une compression optimisée en fonction de régions d'intérêt est une chose. Utiliser les différentes modalités de concert correctement en est une autre. Il s'agit d'éviter de transmettre des informations redondantes, mais aussi de minimiser le temps d'encodage. Néanmoins, le seul fait d'ajouter de nouvelles lignes de code va inévitablement conduire à une augmentation de la puissance de calcul nécessaire. En partant de ce constat, nous avons essayé de minimiser l'impact de la construction des espaces et de l'encodage bimodal en optimisant la façon dont l'encodeur travaille. Les deux encodeurs utilisés lors de la compression ont tous deux fait l'objet d'optimisations.

La vectorisation inclut un module permettant d'estimer s'il est possible de représenter plusieurs zones avec un même polynôme. La décomposition dans le système de couleurs YCrCb fait que la majorité de l'information est contenue dans la composante Y. Pour refléter ce fait et minimiser la quantité d'informations à transmettre, le nombre de chiffres significatifs des coefficients est variable : la composante Y utilise un plus grand nombre de chiffres significatifs que

les deux autres (Cr et Cb). Enfin, l'ensemble des données vecteur (polynômes et rectangles) est compressé avant d'être empaqueté.

L'algorithme permettant la construction des espaces de représentation permet de connaître certaines informations importantes, et notamment la liste des macroblocs constituant chaque modalité. Par défaut, l'encodeur vidéo MPEG-4 AVC traite l'ensemble des macroblocs de l'image de la même manière. Dans notre cas, certains sont inutiles puisqu'ils sont déjà représentés dans un autre espace, et de manière plus compacte. Dans le but de réduire le temps nécessaire à la phase d'analyse, mais aussi pour minimiser l'information portée par ces macroblocs inutiles, deux modifications ont été apportées au code source de l'encodeur. Ainsi, pour chaque image, la liste des macroblocs non utilisés est transmise à l'encodeur pixel. Une optimisation différente est ensuite appliquée, en fonction du type d'image (I ou P, voir 4.1.1) en question. La première, notée O1 (ou dans l'implémentation $L = 1$), agit uniquement sur les images P. Elle s'applique durant la phase d'analyse et force les macroblocs inutiles à être ignorés (directive *skip*). De ce fait aucune estimation de mouvement ne sera réalisée sur ces macroblocs, ce qui permet de gagner du temps et de réduire le débit binaire du flux vidéo produit. La deuxième optimisation, notée O2 (ou $L = 2$) a une influence directe sur les images I, mais aussi un effet indirect sur les images P. Il s'agit cette fois-ci d'agir sur le codage « intra » image. Aucune directive *skip* n'existant sur ce type d'image, la façon la plus simple est de manipuler le coefficient de quantification Qp. Chaque macrobloc inutile est alors quantifié en utilisant la valeur maximale possible. Dans le cas d'usage du *cloud gaming*, le groupe d'images (voir figure 4.2) ne contient que des images de type I et P. En effet le but est de minimiser la latence et le fait d'utiliser des images B en introduit forcément.

En fin de compte, chaque image du flux vidéo fait l'objet d'une optimisation en fonction du masque binaire fourni par le module de construction des espaces. Ces deux optimisations peuvent être utilisées indépendamment l'une de l'autre ($L = 1$ ou $L = 2$), mais aussi de concert ($L = 3$). Différentes expérimentations sont présentées dans le chapitre 11 à ce sujet.

9.5 Empaquetage

Pour éviter des problèmes de synchronisation, chaque paquet contenant une image contient à la fois les données pixel, mais aussi les données vecteur. Comme la séparation des modalités est faite de façon automatique, certains paquets peuvent n'avoir qu'une partie pixel, d'autres qu'une partie vecteur. La majorité des paquets seront constitués d'une partie pixel et d'une partie vecteur. Le flux vidéo est écrit de telle manière que n'importe quel décodeur compatible avec la norme MPEG-4 AVC est en mesure de le lire, il sera néanmoins incapable d'interpréter la modalité vecteur. Seule la partie pixel de l'image sera affichée à l'écran. L'affichage complet d'une image bimodale n'est possible que par l'utilisation d'un décodeur adapté qui décodera la partie pixel, rendra la partie vecteur, fusionnera les deux images et affichera enfin le résultat final.

9.6 Conclusion

Le fonctionnement global de l'algorithme a été présenté de manière détaillée. La majeure partie de ce chapitre est dévolue à la description de la création des espaces. Les traitements sur l'image ainsi que toute la logique utilisée sont présentés : découverte des zones candidates à une compression par un encodeur alternatif, calcul des rectangles les constituant, filtrage, compression respective et enfin empaquetage. Cette vue d'ensemble permet au lecteur une compréhension globale du cheminement utilisé. Une section s'intéresse à la mise en œuvre d'un modèle représentant l'attention visuelle grâce aux données collectées lors de l'analyse de l'AVU d'un panel de joueurs. La dernière section présente quant a elle les optimisations apportées aux deux encodeurs utilisés ainsi qu'à la problématique d'empaquetage.

Chapitre 10

Implémentation

Sommaire

Ce chapitre traite des différents détails relatifs à l'implémentation d'un transmodeur. Le langage choisi, les différentes librairies externes utilisées ainsi que l'architecture adoptée y sont détaillés.

10.1 Langage et librairies

L'encodage vidéo est une tâche qui est relativement lourde à exécuter, même sur du matériel récent. Le choix d'un langage natif s'est donc naturellement imposé. Pour des raisons liées à la fois à la portabilité du code source, mais aussi à la qualité des librairies externes disponibles, le choix du `C++`, dans sa version `C++11` a été fait. L'implémentation actuelle est sous deux formes distinctes : un fichier exécutable qui permet des opérations de transmodage complètes, mais aussi sous la forme d'une librairie pour une intégration facile dans des projets existants (Kusanagi, jeux...). Pour accélérer le développement et réduire la taille du code à maintenir, différentes librairies externes ont été utilisées, elles sont citées ci-dessous :

a. OpenCV [1] : cette librairie permet la gestion des images à partir de structures simples. Les opérations telles que l'accès à chaque pixel, le filtrage morphologique ou l'application de filtres plus complexes, tel celui de Canny sont réalisés à l'aide de cette librairie ;
b. FFmpeg [2] : la gestion du codage et du décodage vidéo est prise en charge par cette brique logicielle. Le nombre de codecs supportés dépend notamment des options de compilation et des librairies présentes sur le système hôte ;
c. x264 [3] : Il s'agit de la meilleure implémentation du codeur MPEG-4 AVC. Cette librairie est utilisée directement par FFmpeg pour le codage d'une vidéo en MPEG-4 AVC ;
d. GSL [4] [5] : ensemble de fonctions mathématiques, permettant notamment de résoudre les problèmes d'algèbre linéaire. L'implémentation est relativement bas niveau, ce qui assure à la librairie des performances intéressantes ;
e. zlib [6] : permet la compression de données à l'aide de différents algorithmes ;
f. TinyXml [7] : librairie utilisée pour la lecture et l'écriture de fichiers XML [8] ;
g. cvblob [9] : utilisé en complément d'OpenCV pour les opérations d'étiquetage ;
h. WSRM : librairie contenant trois méthodes de recherche de la liste des plus grands rectangles dans une image binaire (voir le paragraphe 9.1.2 pour plus de détail).

1. `http://opencv.org`
2. `http://www.ffmpeg.org`
3. `https://www.videolan.org/developers/x264.html`
4. *Gnu Scientific Library*
5. `https://www.gnu.org/software/gsl`
6. `http://www.zlib.net`
7. `http://www.grinninglizard.com/tinyxml`
8. *eXtended Markup Language*
9. `https://code.google.com/p/cvblob`

i. TCLAP [10] [11] : cette libraire permet de manière très simple la gestion des options en ligne de commande.

Le tableau 10.1 détaille les versions utilisées ainsi que les licences sous lesquelles ces logiciels sont utilisables. La colonne « modification » indique simplement si les librairies ont été utilisées telles quelles, ou si des modifications ont été introduites dans leur code source.

TABLE 10.1 – Synthèse des versions et des licences des librairies utilisées.

Nom	**Version**	**Modification**	**License**
OpenCV	2.4.6	non	BSD
FFmpeg	0.11.1	oui	GNU LGPL
x264	0.125	oui	GNU GPL
GSL	1.15.1	non	GNU GPL
zlib	1.2.8	non	zlib
TinyXml	2.6.2.2	non	zlib
cvblob	0.10.4	non	GNU GPL
WSRM	1.0	crée par besoin	Apache
TCLAP	1.2.1	non	MIT

Sur l'ensemble des libraires utilisées, une a été entièrement créée pour répondre à nos besoins, deux ont été légèrement modifiées. La librairie WSRM a pour but de répondre au problème DBMR qui a été explicité dans la section 9.1.2. L'algorithme détaillé est présenté dans l'annexe D. FFmpeg a été modifié pour autoriser une mise à jour de certains paramètres d'encodage utilisés par x264 à n'importe quel moment lors de la compression. La librairie x264 a subi de plus nombreuses modifications, notamment pour gérer les options de compression présentées dans la partie 9.4. Ces modifications permettent une communication directe entre la librairie x264 et un programme externe l'utilisant. Des pointeurs vers des tableaux permettent de marquer chaque macrobloc pour que chacun d'entre eux soit directement considéré comme inutile (*skip* pour les images P) mais aussi pour raffiner le paramètre de quantification Qp. La dernière modification, qui tient plus de l'ajout permet de superposer à l'image originale les décisions prises par l'encodeur lors du traitement de cette même image (partitionnement 16×16, 8×8 ou 4×4, vecteurs de mouvement, décision de *skip*), chaque image étant ensuite enregistrée sur le disque dur.

10.2 Architecture logicielle

L'implémentation actuelle est constituée de trois projets logiciels, aboutissant à la génération de trois fichiers exécutables :

1. Le projet **transmodeur**, qui permet de transmoder une vidéo ;
2. Le projet **lecteur**, qui permet de lire un flux transmodé ;

10. *Templatized C++ Command Line Parser*
11. `http://tclap.sourceforge.net`

3. Le projet **psnr**, qui permet de faire une étude sur la qualité du flux transmodé produit, en fonction du flux original et du flux transcodé.

Le code source est en outre constitué de onze classes principales, leurs noms et fonctions sont cités ci-dessous :

1. *binaryDecoding* : décode le flux de données de la partie vecteur ;
2. *binaryEncoding* : encode le flux de données de la partie vecteur ;
3. *compressData* : compresse des données (sans perte) ;
4. *decodeRawVideo* : décode un flux vidéo non compressé ;
5. *decodeVideo* : décode un flux vidéo compressé ;
6. *encodeVideo* : encode un flux vidéo (encodage uni ou multimodal) ;
7. *polynomialInterpolation* : représente de manière paramétrique une zone de l'image ;
8. *sunder* : découpe une image en modalités ;
9. *transmodeVideo* : classe principale du projet transmodeur ;
10. *decodeTransmodedVideo* : classe principale du projet lecteur ;
11. *psnr* : classe principale du projet psnr.

L'architecture logicielle des trois projets est détaillée dans les trois sections suivantes.

10.2.1 Le transmodeur

L'implémentation actuelle présentée par la figure 10.1 du transmodeur permet la lecture de n'importe quel flux vidéo compressé, pourvu qu'il soit pris en charge par les librairies constituant FFmpeg. En sortie, on retrouve soit un fichier vidéo, soit un flux vidéo si l'utilisateur souhaite diffuser la vidéo sur le réseau. Le fichier de sortie est sous forme d'un flux vidéo encodé en MPEG-4 AVC, encapsulé dans le conteneur standard mp4 [12]. Lors d'une diffusion en temps réel, le flux vidéo (toujours encodé en MPEG-4 AVC) est encapsulé par la suite de protocole RTP/RTCP.

Différentes options permettent d'enregistrer toutes les données intéressantes lors d'un transmodage. Les décisions prises par l'algorithme, la surface des différentes zones ou encore les temps de calcul sont ainsi consignés. Ces données sont utilisées dans le chapitre suivant pour permettre l'évaluation de la solution proposée.

10.2.2 Le lecteur

Le lecteur permet la lecture de tous les flux vidéos compressés. Il permet le décodage d'un flux multimodal, c'est-à-dire la décompression du flux vidéo MPEG-4 AVC, le rendu de la partie paramétrique et la recomposition de l'image finale. Le diagramme de classe simplifié est présenté par la figure 10.2.

12. Conteneur pour encapsuler des données de type multimédia (audio ou vidéo essentiellement).

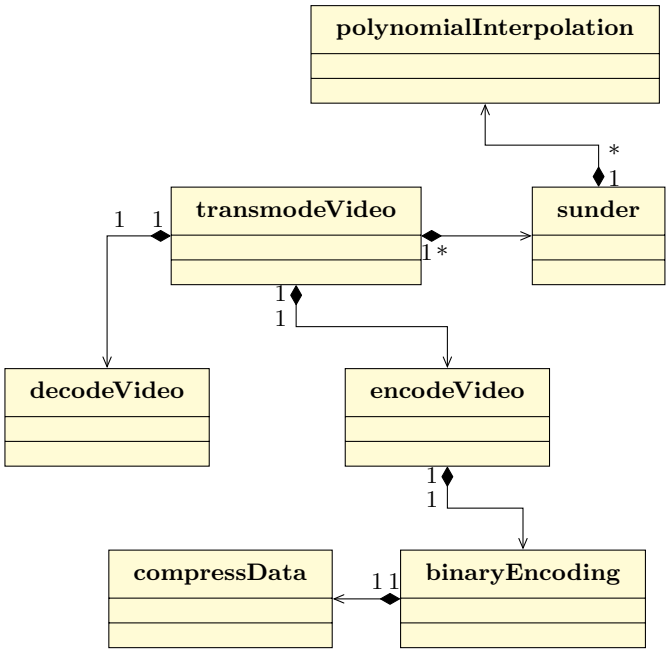


FIGURE 10.1 – Diagramme de classe simplifié pour l'implémentation du transmodeur.

10.2.3 Le projet psnr

Ce programme permet l'évaluation de la qualité d'un flux multimodal par rapport à son équivalent transcodé, mais aussi face à la vidéo originale, non compressée. La métrique de qualité utilisée ici est le PSNR. Pour permettre une analyse fine des résultats, le PSNR est calculé pour l'image entière, mais aussi pour chaque modalité. Il est ainsi possible de comparer la version pixel et la version vecteur d'une même zone de l'image. Les données sont là aussi enregistrées pour faciliter leur analyse. La figure 10.3 présente l'architecture de ce projet.

10.2.4 Librairies

Pour faciliter l'ajout du support d'un encodage multimodal dans la chaîne de rendu de Kusanagi, deux librairies ont été crées. La première est dévolue à l'encodage en lui même, la deuxième au décodage. Toutes deux possèdent deux interfaces : une en `C` et une en `C++`. La librairie de transmodage est intégrée directement dans la partie serveur de la chaîne de rendu de Kusanagi. Quant à la librairie de décodage, elle est présente dans le lecteur vidéo utilisé par la plate-forme, à savoir GPAC. Ces mêmes librairies sont intégrées dans la chaîne de rendu du projet XLcloud.

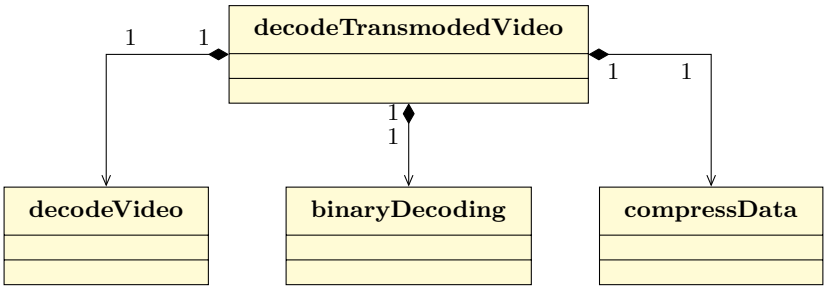


FIGURE 10.2 – Diagramme de classe simplifié pour l'implémentation du lecteur.

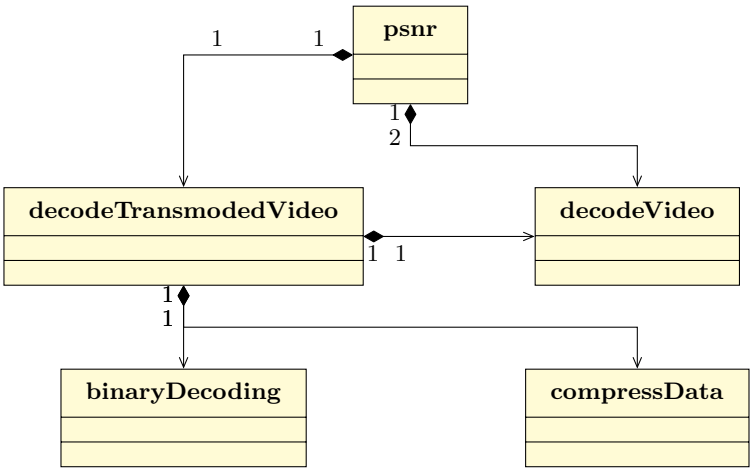


FIGURE 10.3 – Diagramme de classe simplifié pour l'implémentation du projet psnr.

10.3 Configuration

À la compilation, l'utilisateur peut choisir entre une version se basant sur des fichiers de configuration, ou sur une autre utilisant des options en lignes de commande. Quoi qu'il arrive, l'utilisation du transmodeur sous l'une ou l'autre de ses formes implique une configuration préalable. Les sections suivantes décrivent l'utilisation de ces deux solutions.

10.3.1 Fichiers de configuration

Ce sont de simples fichiers texte, écrits en XML et qui permettent de configurer le transmodeur, le lecteur et le projet psnr. Chaque exécutable attend un certain nombre de fichiers XML avec des noms prédéfinis. Ils doivent se trouver dans le même répertoire que l'exécutable. Le tableau 10.2 résume la situation. Pour le transmodeur on distingue trois fichiers de configurations, chacun est dévolu à une opération spécifique. Dans ce cas là, on retrouve un fichier de configuration pour le décodage, un pour la séparation des modalités et un pour l'encodage. Le lecteur n'utilise qu'un seul fichier de configuration, relatif au décodage. Quant au projet psnr il s'appuie sur trois fichiers de configuration, tous

trois dévolus au décodage des différents fichiers vidéos. Il en est de même pour la librairie de transmodage qui utilise un seul fichier de configuration relatif à la séparation des modalités.

TABLE 10.2 – Fichiers de configuration nécessaires pour chaque fichier exécutable.

Projet	**Fichiers de configuration**
transmodeur	decodeVideo.cfg.xml encodeVector.cfg.xml encodeVideo.cfg.xml
lecteur	decodeMixteVideo.cfg.xml
psnr	decodeOriginalVideo.cfg.xml decodeTransmodedVideo.cfg.xml decodeTranscodedVideo.cfg.xml
Librairie de transmodage	encodeVector.cfg.xml

10.3.2 Options en ligne de commandes

Cette alternative n'utilise pas de fichiers XML pour la configuration, mais simplement des options qui seront à préciser dans l'interface de la ligne de commande. Les fichiers exécutables générés de telle sorte sont utilisés par un langage de script, notamment lors des tests.

10.3.3 Liste et description des options

Les options relatives au décodage, à la séparation des modalités et à l'encodage sont respectivement présentées et détaillées dans les annexes G.1, G.2 et G.3. Le nom de toutes les options figurant dans les fichiers de configurations XML, ainsi que leurs équivalents pour une utilisation en ligne de commande sont précisés. Chaque option fait l'objet d'une courte description ; les valeurs par défaut ainsi que les valeurs possibles sont détaillées. Pour une utilisation basique, seuls les chemins vers les fichiers vidéos à lire et/ou à écrire sont nécessaires. Tous les autres paramètres possèdent des valeurs par défaut.

10.4 Passage à l'échelle

Le but de la première implémentation était avant tout de fournir un logiciel fonctionnel permettant d'expérimenter la solution proposée. Celle-ci était capable de tenir le temps réel (25 à 30 images par seconde) sur des résolutions spatiales faibles (320×240). Dans un second temps, une version largement améliorée a été mise au point. Pour cela, les points faibles de la première implémentation ont été revus. Ainsi l'algorithme DBMR proposé en annexe D a été implémenté et intégré. Pour séparer les tâches pouvant être réalisées parallèlement, l'architecture logicielle a été légèrement modifiée ; rendant possible l'implémentation de solutions *multithread*. Différents niveaux de parallélisation sont

proposés, permettant d'optimiser au mieux le fonctionnement du programme. L'utilisateur garde tout de même le contrôle grâce à des options qui lui permettent d'activer ou non la gestion *multithread* dans les différentes portions du programme. Ces options sont notamment mises en avant dans la section 11.5 du chapitre suivant, lors de l'évaluation des performances. Le transmodeur étant une brique du projet XLcloud, il serait astucieux d'utiliser les avantages du *cloud* pour améliorer de manière significative les performances.

Des travaux dans ce sens ont été entrepris, notamment grâce à l'utilisation d'une approche *map/reduce* au moyen de la librairie Gearman [13]. Une telle implémentation est en outre rendue plus aisée du fait que les tâches concurrentes ont déjà été découplées les unes des autres lors de la modification apportant le support du *multithreading*. Le transcodage dans le *cloud* est d'ailleurs une activité qui tend à se développer rapidement, de grands acteurs comme Microsoft ou Amazon ont récemment proposé de telles solutions : elles sont respectivement connues sous les dénominations Windows Azure Media Services [14] et AWS Elastic Transcoder [15]. L'élasticité permise par ces solutions permet de répondre rapidement aux fluctuations d'affluence pour un service donné en adaptant le nombre de machines (virtuelles ou non) qui lui sont dévolues. Ce type d'approche est donc particulièrement adapté aux services de *cloud gaming*. L'implémentation *map/reduce* du transmodeur, tant en version *standalone* que sous forme d'API est bien avancée, mais n'est pas encore fonctionnelle. Le code source des différentes implémentations décrites dans ce chapitre est disponible en ligne [16] sur la forge Bitbucket.

10.5 Conclusion

Ce chapitre a présenté l'architecture logicielle et l'implémentation existante de notre transmodeur. Différentes utilisations sont possibles que ce soit grâce aux exécutables ou via l'utilisation des librairies. Seuls les paramètres importants ont été présentés, néanmoins, toutes les options existantes sont précisées dans l'annexe G.

13. `http://gearman.org`
14. `https://azure.microsoft.com/en-us/services/media-services`
15. `https://aws.amazon.com/fr/elastictranscoder`
16. `https://bitbucket.org/trynitron/transmoder`

Chapitre 11

Analyse de l'espace de représentation hybride

Sommaire

Ce chapitre a pour but d'évaluer les performances offertes par l'utilisation d'un espace de représentation hybride par rapport à l'utilisation d'un espace unique qui serait entièrement constitué de pixels. L'étude a été menée dans le cadre du *cloud gaming*. Un nombre important de paramètres pouvant influer sur le résultat final, un protocole de test strict a été utilisé. Dans un premier temps, c'est ce protocole de test qui est présenté. L'ensemble des tests ont été réalisés sur une machine virtuelle opérant sous Windows 8.1, constituée de quatre processeurs Intel® Xeon® E5430 à 2,66 GHz et de 8 Go de mémoire vive.

11.1 Protocole de test

L'implémentation du transmodeur utilisé dans ce chapitre a été détaillée dans le chapitre 9. L'ensemble des options utilisables y a notamment été présenté. Leur nombre étant relativement important, il a été décidé de focaliser les tests sur un nombre limité d'entre elles. Les noms des paramètres dont l'influence a été étudiée sont énumérés ci-dessous :

a. Le *preset* p utilisé ;
b. La valeur du paramètre de quantification Qp ;
c. Le nombre d'étiquettes v ;
d. Le nombre de rectangles par étiquette b ;
e. Le degré d'optimisation de l'encodage vidéo L.

Pour évaluer l'influence de ces différents paramètres, on considère les quatre métriques suivantes :

1. La taille du flux vidéo ;
2. Le temps nécessaire au traitement ;
3. Le pourcentage d'images traitées de manière bimodale ;
4. La valeur du PSNR, toutes modalités confondues.

Pour permettre une comparaison des résultats, l'encodeur vidéo utilise une stratégie de quantification à Qp constant. De plus, le *cloud gaming* impose une configuration spécifique du système de compression vidéo : celle-ci doit permettre un traitement temps réel, mais aussi une latence la plus proche de 0. La configuration de la librairie x264 se fait principalement en deux étapes : le choix d'un *preset*, suivi de l'application d'un *tune* (ce dernier a pour effet d'affiner la configuration). Il est donc nécessaire d'utiliser un *preset* offrant un temps de traitement court (*veryfast*, *superfast* ou *ultrafast*) et de choisir l'option de raffinement des paramètres permettant de minimiser la latence en utilisant le *tune* nommé *zerolatency*. Au niveau de l'encodeur, la latence est réduite au minimum en supprimant l'utilisation des images B ; le flux vidéo sera en conséquence uniquement constitué d'images I et P.

Les différentes valeurs utilisées durant les tests sont présentées dans le tableau 11.1. Toutes les options dont il n'est pas question ici restent à leur valeur par défaut, telle que définie dans l'annexe G. Tous les tests présentés ci-dessous ont été réalisés sur trois vidéos différentes, toutes issues du jeu Doom3. Ces vidéos représentent divers environnements du jeu, mêlant scènes intérieures, extérieures avec ou sans action. Après extraction du moteur de rendu, chaque

Nom qualifié	Nom de l'option	Valeurs utilisées
preset	*preset*	*veryfast*, *superfast* et *ultrafast*
valeur du paramètre de quantification	*options*	de 18 à 36, par pas de 2
nombres d'étiquettes	*maxVectorAreas*	1 et 2
nombres de rectangles	*maxVectorLRAreas*	1, 2 et 4

TABLE 11.1 – Valeurs utilisées pour les options étudiées.

image est stockée dans un fichier au format YCrCb, ce qui permet de préserver tous les détails nécessaires. Les vidéos ont toute une résolution spatiale de 480p. Tout au long de ce chapitre, toutes les illustrations présentées portent sur la séquence Doom3 – 1, même si certains graphiques présentent aussi des résultats pour les deux autres séquences étudiées.

11.2 Transcodage simple

Cette section s'intéresse au choix du *preset* à utiliser lors des sessions de *cloud gaming*. La figure 11.1 représente la compression d'une même vidéo avec l'utilisation de trois *preset* différents, pour des valeurs de quantification comprises entre 18 et 36.

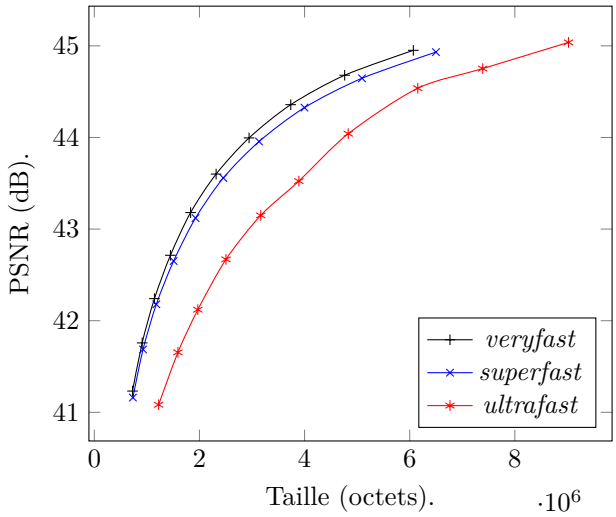


FIGURE 11.1 – Transcodage en fonction de trois *preset* différents (séquence vidéo Doom3 – 1).

Les trois courbes représentent la compression vidéo de référence, pour les trois *preset* les plus rapides. De la même manière, les temps de traitements nécessaires sont présentés au moyen de le figure 11.2.

On remarque bien que le *preset ultrafast* est effectivement le plus rapide. Dans cette configuration, tous les *preset* sont utilisables tels quels pour la compression d'un flux vidéo en temps réel. Néanmoins, la résolution spatiale utilisée

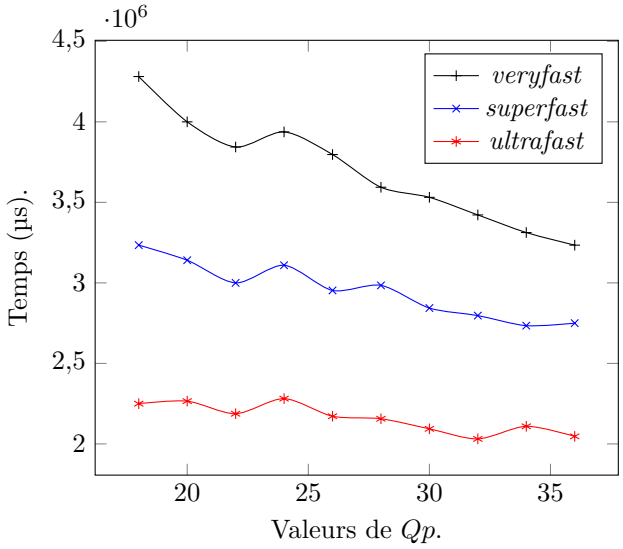


FIGURE 11.2 – Temps nécessaire au transcodage en fonction de trois *preset* différents (séquence vidéo Doom3 – 1).

est relativement faible et l'utilisation du 720p [1] ou du 1080p nécessite l'utilisation du *preset ultrafast*. Dans cette optique, tous les tests suivants sont réalisés avec l'option *ultrafast*. On privilégie la vitesse de compression face au volume du débit binaire produit.

11.3 Transmodage

Après avoir obtenu les données des fichiers vidéo transcodés, on s'intéresse maintenant aux résultats obtenus grâce à l'utilisation d'un encodeur multimodal.

11.3.1 Détection des zones utilisables

Ce paragraphe a pour but d'évaluer la surface utilisable pour une compression par une seconde modalité. Les tests réalisés s'intéressent ici aux données suivantes :

a. La surface utilisable, juste après l'application du filtre de Canny ;
b. La surface utilisée, en fonction des paramètres *maxVectorAreas* et *maxVectorLRAreas* ;
c. Le nombre d'images impactées.

La figure 11.3 représente pour chaque image de la vidéo Doom3 – 1 le pourcentage de la surface utilisable par une seconde modalité. Ces valeurs sont obtenues après l'application du filtre de Canny et avant la réduction du nombre d'étiquettes et du nombre de rectangles par étiquette. Selon la figure 11.3 et

1. Résolution spatiale de 1280 × 720 pixels.

dans ce cas précis, la part de l'image utilisable par une seconde modalité varie de 20 à plus de 70%.

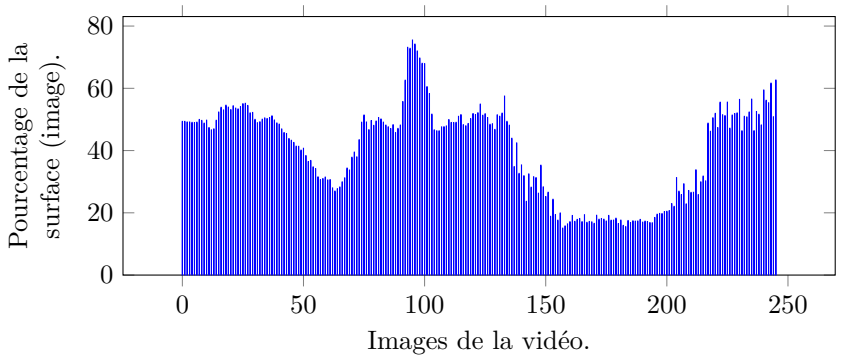


FIGURE 11.3 – Évolution du pourcentage de la surface utilisable (sur chaque image de la vidéo) par la seconde modalité, avant l'application des paramètres *maxVectorAreas* et *maxVectorLRAreas* (séquence vidéo Doom3 − 1).

À ce stade-là, une partie importante de chaque image est considérée comme pouvant être compressée de manière alternative. Cette part va être significativement réduite par l'application des paramètres *maxVectorAreas* et *maxVectorLRAreas*, respectivement contrôlés par les options v et b. À ce propos, la figure 11.4 expose le pourcentage total de la surface dédié à la compression par la seconde modalité, en se limitant à une étiquette. Qaunt à la figure 11.5 elle fait état de la même métrique, mais pour deux étiquettes cette fois-ci. Les valeurs présentées par les figures 11.4 et 11.5 ne sont pas des moyennes puisque chaque image n'est pas forcément composée de deux modalités. Par exemple, si l'on considère le cas de la séquence Doom3 − 1 avec les paramètres $v = 1$ et $b = 1$, $1,92\%$ de la surface du flux vidéo multimodal est traité par la seconde modalité ; ce qui implique que $98,08\%$ est compressé par la première modalité.

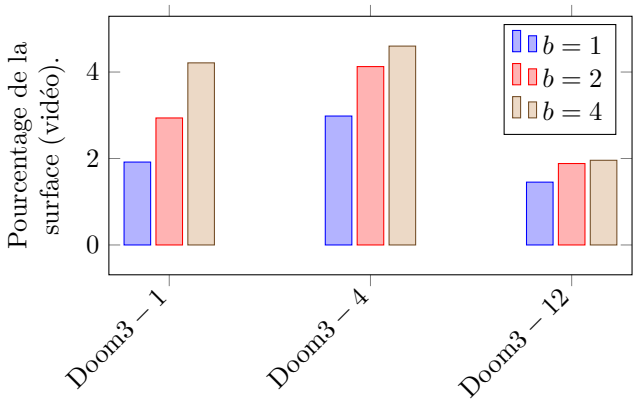


FIGURE 11.4 – Évolution du pourcentage de surface utilisé (sur l'ensemble de la vidéo) avec une étiquette ($v = 1$) et en fonction du nombre de rectangles.

Évidemment, ce taux dépend de la vidéo elle-même et du nombre de rec-

tangles autorisés pour la représentation d'une étiquette. Néanmoins, et comme on pouvait s'y attendre, plus le nombre de rectangles par étiquette augmente (c'est à dire quand b croît), plus la surface globale utilisée par la seconde modalité augmente. De manière similaire, le nombre d'étiquettes retenu influe lui aussi sur la surface utilisée par la seconde modalité : plus v est grand, plus la surface dévolue à la seconde modalité croît.

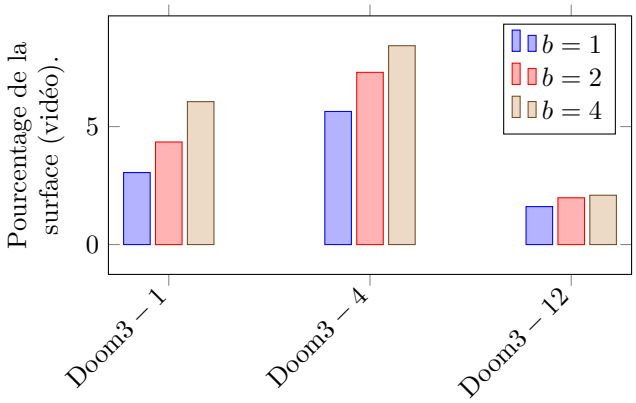


FIGURE 11.5 – Évolution du pourcentage de surface utilisé (sur l'ensemble de la vidéo) avec deux étiquettes ($v = 2$) et en fonction du nombre de rectangles.

Une dernière information est nécessaire dans ce paragraphe, il s'agit du nombre d'images compressées par les deux modalités. La figure 11.6 présente une frise chronologique dans laquelle les images compressées de manière bimodale sont représentées sous la forme de traits bleus. Les images compressées uniquement avec l'encodeur MPEG-4 AVC sont quant à elles représentées par un trait blanc. Dans ce cas de figure, seulement 86 images sur un total de 246 seront représentées par deux modalités. Le tableau 11.2 fait le point sur le pourcentage d'images traitées de manière bimodale pour les trois séquences de test. Ce pourcentage évolue de 8 à 59% lors des tests avec une seule étiquette ($v = 1$) et de 10 à 79% avec deux étiquettes (voir tableau F.1). Il est intéressant de noter que dans la plupart des cas le fait d'augmenter la surface utilisable par la seconde modalité n'implique pas forcément une augmentation du nombre d'images considérées de manière bimodale. Par exemple, pour la séquence Doom3 – 1 la variation du paramètre b de 1 à 4 entraîne un doublement de la surface utilisable, alors que le nombre d'images traitées de manière bimodale baisse dans le même temps de 3%.

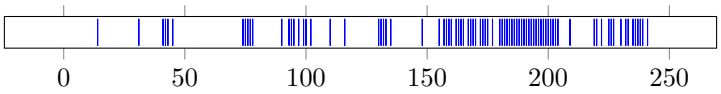


FIGURE 11.6 – Chronologie des images considérées comme pouvant être utilisées par une seconde modalité de compression (séquence vidéo Doom3 – 1 avec $v = 1$ et $b = 1$).

Le nombre d'images traitées de manière bimodale évolue en fonction des paramètres choisis lors du processus de compression. À ce titre, la figure F.1

faisant partie de l'annexe F.1 permet de mettre en évidence cette évolution par rapport aux paramètres v et b, même si d'autres paramètres entrent en action, notamment ceux responsables du contrôle du mécanisme de regroupement de zones (au sein de la seconde modalité). Pour cette séquence vidéo (Doom3 − 1), le nombre maximal d'images traitées grâce à l'utilisation de deux modalités distinctes est atteint avec les paramètres $v = 2$ et $b = 1$ (deux étiquettes et un rectangle par étiquette) pour lesquels la valeur de 119 est atteinte.

TABLE 11.2 – Récapitulatif du nombre d'images traitées de manière bimodale pour les trois séquences de test (avec $v = 1$).

		Images bimodales		
Séquence	**Nombre d'images**	$b = 1$	$b = 2$	$b = 4$
Doom3 − 1	246	86 (35%)	72 (29%)	79 (32%)
Doom3 − 4	246	146 (59%)	123 (50%)	127 (52%)
Doom3 − 12	496	55 (11%)	46 (9%)	38 (8%)

11.3.2 Influence du nombre d'étiquettes & du nombre de rectangles les constituant sur le débit binaire

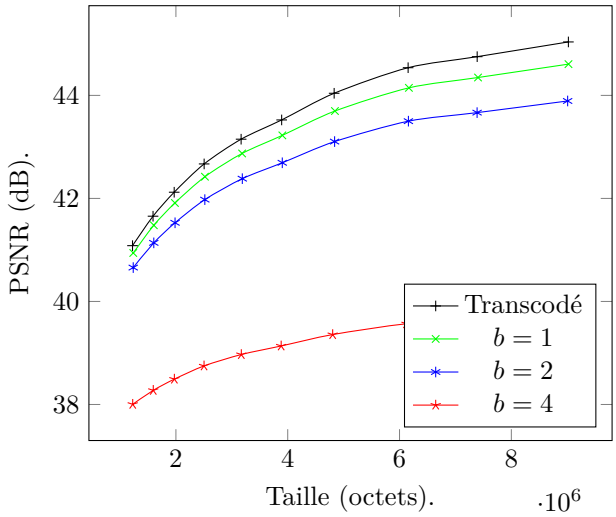


FIGURE 11.7 – Influence du nombre de rectangles par étiquette sur le débit binaire (séquence vidéo Doom3 − 1, avec $v = 1$).

Après s'être intéressé aux zones détectées et aux sous-ensembles utilisables pour un encodage multimodal, il s'agit maintenant dévaluer l'impact des paramètres utilisés d'une part lors de la détection, mais aussi lors du raffinement des zones exploitées par la seconde modalité d'encodage. Dans ce but, les figures 11.7 et 11.8 seront exploitées. Toutes deux s'intéressent à l'influence du nombre de rectangles utilisés, à cela près que la première le fait avec une seule étiquette et la seconde avec deux.

Dans le premier cas de figure (figure 11.7), la mise en œuvre d'un ou deux rectangles permet de rester très proche de l'équivalent transcodé en terme de PSNR. L'essai effectué avec quatre rectangles se détache plus nettement, la valeur de la métrique objective diminuant d'environ 2,4 dB.

Les tests réalisés en conservant deux étiquettes et en faisant varier le nombre de rectangles de 1 à 4 pour chacune d'entre elles (c'est-à-dire un nombre maximal de 8 rectangles par image) sont présentés par la figure 11.8. Les résultats sont plus serrés qu'avec une seule étiquette, en effet l'ensemble des courbes n'est séparé que d'environ 0,5 dB.

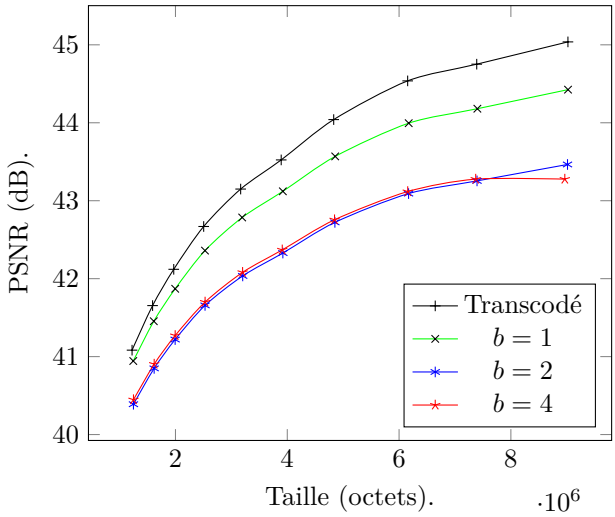


FIGURE 11.8 – Influence du nombre de rectangles par étiquette sur le débit binaire (séquence vidéo Doom3 − 1, avec $v = 2$).

Il est intéressant de remarquer que la valeur du PSNR est presque identique, que ce soit avec une ou deux étiquettes elle se situe aux alentours de 41 dB. La figure 11.9 se focalise sur l'évolution de la différence relative du débit binaire entre équivalents transcodés et transmodés. Les valeurs positives retranscrivent un débit binaire à l'avantage de la version transmodée, c'est à dire plus faible que son équivalent transcodé.

Ce graphique (voir figure 11.9) permet de mettre en avant deux choses. Tout d'abord, l'efficacité (en terme de débit binaire) du transmodeur est liée à la valeur du paramètre de quantification Qp : plus celui-ci est faible, plus le transmodeur devient efficace. Le second point est la mise en évidence de l'impact du paramètre b sur le débit binaire. En effet, plus b est grand, plus la seconde modalité devient importante (en surface) : ceci a pour conséquence dans la plupart des cas de faire baisser le débit binaire (et donc de faire augmenter la valeur de différence relative). La figure F.4 présente dans l'annexe F confirme ces dires en proposant la même approche, mais cette fois avec l'utilisation de deux étiquettes ($v = 2$). De plus, on remarquera que la différence relative de taille est plus importante lorsque $v = 2$ (figure 11.9) que lorsque $v = 1$ (figure F.4). Là encore, on peut conclure que lorsque la surface dédiée à la seconde modalité augmente, le débit binaire du flux transmodé diminue (par rapport à son équivalent transcodé).

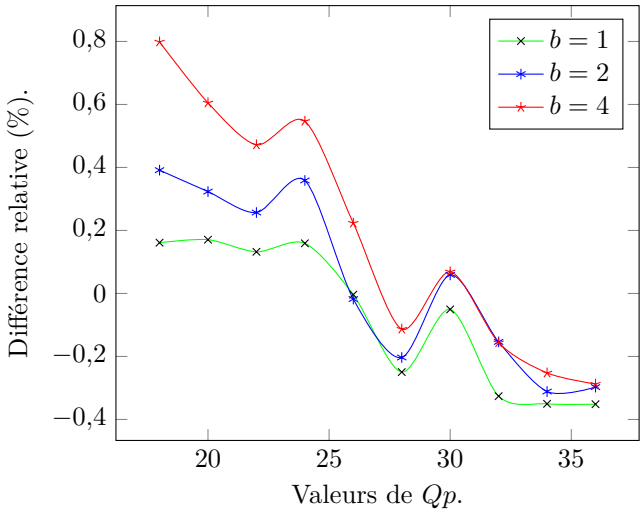


FIGURE 11.9 – Différence relative du débit binaire en fonction de la valeur de Qp (séquence vidéo Doom3 − 1, avec $v = 1, L = 3$ et $p = ultrafast$).

11.3.3 Impact des optimisations apportées à l'encodeur vidéo

Au vu du pourcentage d'images traitées par les deux encodeurs (les images bimodales) et de la surface utilisée par la seconde modalité, il faut s'attendre à de faibles variations du débit binaire. Les optimisations dont l'impact est étudié ici ont été présentées dans la section 9.4. Pour rappel la première agit sur les images P en forçant les macroblocs utilisés par la seconde modalité à être traités dans le mode *skip* par l'encodeur vidéo, la deuxième agit directement sur les images I en modifiant le paramètre de quantification Qp de ces même macroblocs (elle agit donc aussi, mais de manière indirecte sur les images P). Ces deux optimisations sont utilisables séparément comme de concert, c'est pour cela que le paramètre L – en charge de la gestion de cette optimisation – peut prendre quatre valeurs différentes ($L = \{0, 1, 2, 3\}$). La section 9.4 et l'annexe G.3 apportent plus de détails à ce sujet. La figure 11.10 permet d'apprécier l'influence du paramètre L sur la séquence Doom3 − 4. Il est possible de déduire plusieurs informations de ce graphique. Tout d'abord, quand $L = 1$, on voit une nette baisse du débit binaire, baisse qui est bien moins flagrante avec $L = 2$ et $L = 3$. Ceci peut s'expliquer par le fait que lorsque $L = 1$ l'optimisation agit sur les images P, bien plus représentées statistiquement dans le flux vidéo que les images I. La deuxième remarque que l'on peut faire porte sur l'évolution du gain en fonction de la valeur du paramètre de quantification : plus la valeur de Qp augmente, plus le gain devient faible.

11.4 Mise en œuvre des cartes d'attention

Cette section s'intéresse à l'impact engendré par l'utilisation des cartes d'attention précédemment évaluées et modélisées. Dans une première expérimentation, leur impact est évalué lors d'un transcodage. Une seconde évaluation

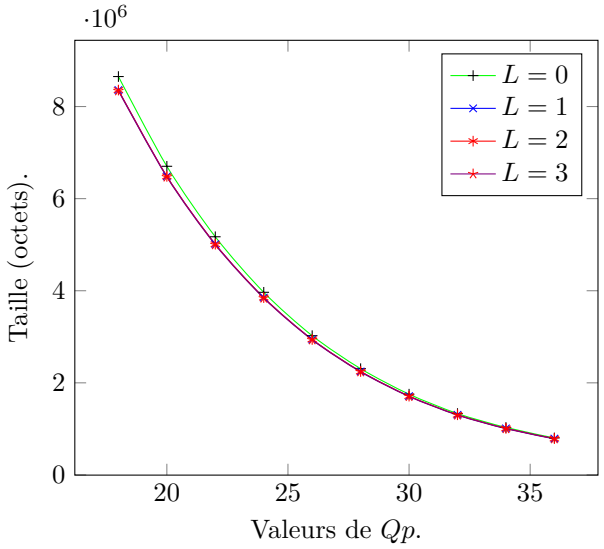


FIGURE 11.10 – Influence de l'option L sur le débit binaire (séquence vidéo Doom3 − 4 avec $v = 1, b = 1$ et $p = superfast$).

expose les résultats obtenus lors de leur utilisation avec notre transmodeur.

11.4.1 Transcodage seul

La figure 11.11 montre pour chaque profil utilisé, son influence sur le débit binaire. Pour rappel, la gestion des cartes d'attention et les valeurs des paramètres associés ont étés décrits dans la section 9.2. À elle seule, l'utilisation de $p = 1$ entraîne une baisse d'environ 15% du débit binaire. Si $p = 2$ la réduction atteint 25% ; la valeur de 45% de réduction est obtenue avec $p = 4$. Le profil 0 est le moins agressif alors que c'est le profil 3 qui permet la plus forte baisse du débit binaire.

L'utilisation du profil le moins impactant d'un point de vue de l'utilisateur permet à lui seul de réduire le débit binaire de façon importante. Il est toutefois possible de réduire encore un peu plus la quantité d'informations en mettant en œuvre notre encodeur multimodal.

11.4.2 Transmodage associé à la mise en œuvre des cartes d'attention

L'utilisation des modèles d'AVU lors du processus de transmodage va forcément entraîner des différences par rapport à l'utilisation seule du transmodeur. Dans le but de réduire le débit binaire tout en proposant au joueur une qualité de jeu optimale, seule la zone de moindre importance (telle que décrite dans la figure 9.13) est considérée. En résumé, chaque image est composée de trois zones, chacune utilisant une valeur de quantification différente. Seule la surface définie par la zone périphérique (ou de moindre importance) peut être traitée grâce à l'utilisation des modalités disponibles.

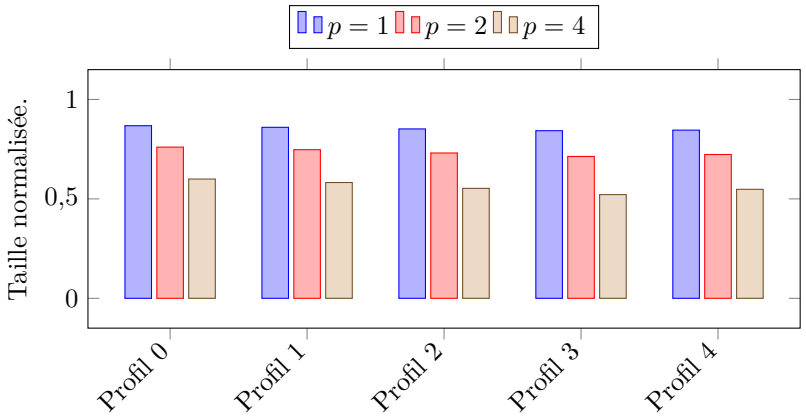


FIGURE 11.11 – Impact de la mise en œuvre des modélisations de cartes d'attention visuelle sur le débit binaire lors d'un transcodage (séquence vidéo Doom3−1 avec $v = 1, b = 1$ et $p = superfast$).

Les modèles de cartes d'AVU ont une influence directe sur le processus de transmodage, et notamment sur l'évaluation des zones utilisables par une seconde modalité. En d'autres termes, il faut s'attendre à ce que les statistiques (chronologie, nombre d'images traitées de manière bimodale) à ce sujet évoluent au gré des modèles utilisés. La figure F.3 permet d'illustrer ces propos. Elle présente les différentes chronologies obtenues après application des différents modèles de profils d'AVU. Les cinq chronologies présentées dans la figure F.3 sont à comparer avec la première de la figure F.1.

Comme dans le cas d'un transcodage, la figure 11.12 présente l'évolution des débits binaires en fonction des différents profils utilisés.

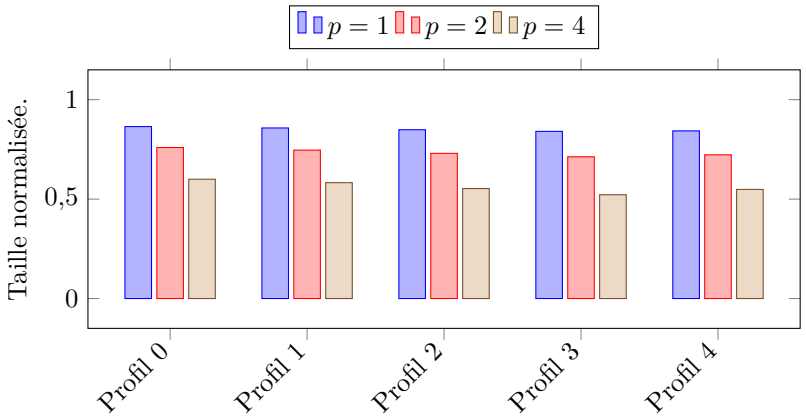


FIGURE 11.12 – Impact de la mise en œuvre des modélisations de cartes d'attention visuelle sur le débit binaire lors d'un transmodage (séquence vidéo Doom3 − 1 avec $v = 1, b = 1$ et $p = superfast$).

Les différences entre les figures 11.11 et 11.12 sont faibles, mais existent. La figure 11.13 met en valeur ces différences au moyen de la séquence vidéo

Doom3 − 12. La différence présente entre les versions transcodées et transmodées (avec utilisation des cartes d'attention visuelle) est toujours à mettre au profit de l'encodeur multimodal. On remarque aussi que la différence relative des tailles est presque proportionnelle au paramètre p. Dans cette expérimentation, la différence relative maximale est de $0,69\%$, pour $Qp = 25$. Comme l'a montré la figure 11.10, plus la valeur de Qp est faible, plus l'impact de l'optimisation L devient important. Ainsi, cette même expérience réalisée avec un Qp plus faible devrait donner de meilleurs résultats.

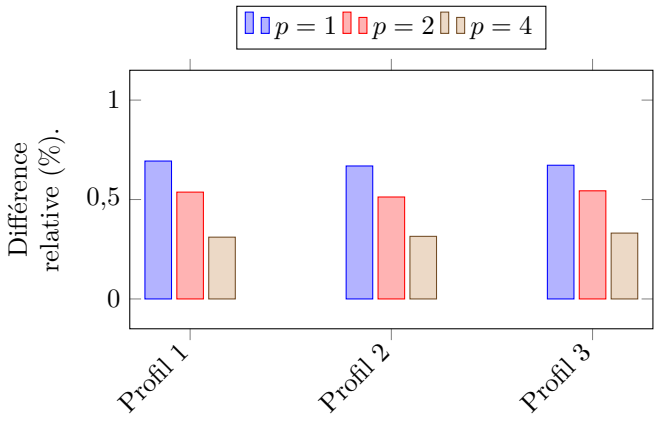


FIGURE 11.13 – Comparaison des débits binaires (transcodage vs transmodage) avec l'utilisation des profils d'attention visuelle (séquence vidéo Doom3 − 12 avec $v = 2, b = 2$, $p = superfast$ et $Qp = 25$).

11.5 Performances & limites de fonctionnement

Après une étude détaillée des résultats obtenus par notre transmodeur, cette section s'intéresse à son efficacité non pas de manière qualitative, mais de façon temporelle. Comme le chapitre dédié à l'implémentation l'a détaillé, le fait même d'ajouter de nouvelles lignes de code pour permettre la gestion de plusieurs modalités va forcément entraîner une baisse de la rapidité de traitement. Dans le but de minimiser ce surcoût temporel, diverses tactiques ont été mises en œuvre et implémentées, notamment une optimisation de l'encodage vidéo, mais aussi une parallélisation des tâches à travers l'utilisation du *multithreading*. Dans cette optique-là la figure 11.14 présente le traitement des séquences vidéo en fonction de l'option T qui contrôle le nombre de *threads* utilisés lors du processus.

11.5.1 Autres cas d'usage

L'intégralité de ce manuscrit se focalise sur le cas d'usage du *cloud gaming*. Néanmoins, tels que présentés, la formalisation (chapitre 8), le fonctionnement (chapitre 9) ainsi que l'implémentation (chapitre 10) permettent l'utilisation du transmodeur dans un grand nombre d'applications. Il est ainsi possible d'étudier l'impact de l'utilisation de notre encodeur multimodal sur des vidéos « classiques ». C'est ce type d'expérimentation qui a été réalisé dans notre article [133].

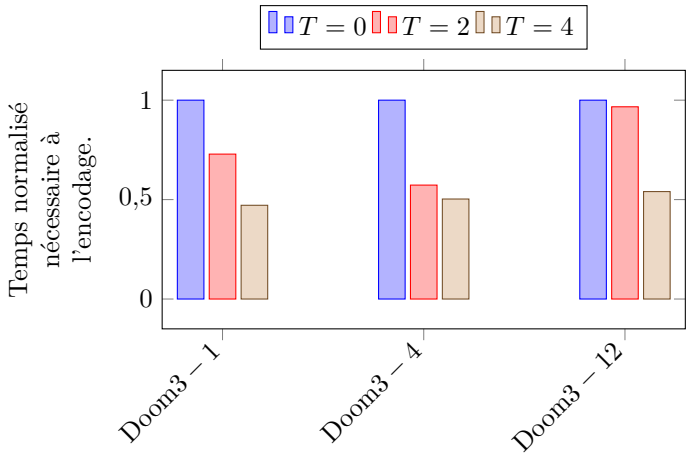


FIGURE 11.14 – Évolution du temps nécessaire à la compression bimodale en fonction du nombre de *threads* utilisé (avec $v = 1, b = 1, p = ultrafast, Qp = 24$).

Le fonctionnement du transmodeur étant dépendant de la nature des images à traiter, des vidéos de différentes natures ont été utilisées. Les résultats sont intéressants puisque selon les vidéos et les paramètres utilisés, tous les cas de figure se sont présentés. Dans trois cas sur quatre, le débit binaire a été réduit, et ce jusqu'à 8.6% par rapport aux versions transcodées équivalentes. Plus intéressant, trois vidéos transmodées (sur quatre) présentent un PSNR plus important dans leur version multimodale, toujours face à leur équivalent transcodé.

11.6 Conclusion

Les tests réalisés dans ce chapitre ont permis d'évaluer notre transmodeur au moyen de trois séquences vidéo extraites du jeu Doom3. La première partie de l'analyse s'est focalisée sur un aspect statistique et a permis de montrer qu'une proportion relativement faible de chaque vidéo peut être compressée de manière bimodale. Par ailleurs, lorsqu'une image peut être traitée de manière bimodale, seule une petite partie de sa surface l'est. Cette conclusion est évidemment liée aux vidéos elles-mêmes, mais aussi au fonctionnement et au paramétrage de notre algorithme.

La deuxième partie s'est focalisée sur une analyse des vidéos transmodées tout en observant l'impact des paramètres principaux. Dans la plupart des cas, la taille des fichiers ainsi traités est plus faible que leur équivalent transcodé. L'analyse objective à l'aide du PSNR donne dans notre cas des résultats décevants, ce qui n'est pas vraiment une surprise puisque la compression par la seconde modalité a un effet de lissage. Néanmoins, la qualité des zones importantes n'est pas dégradée puisque seules des zones faisant partie de la région périphérique du système de vision humain sont concernées.

La seconde expérimentation a mis en avant la possibilité de réduire de manière significative le débit binaire grâce à l'utilisation de cartes d'attention spécifiques à chaque jeu (au moyen d'une modélisation de l'attention visuelle). Une

modification faible du paramètre de quantification, imperceptible de manière subjective peut permettre une diminution importante de la bande passante nécessaire.

Enfin une analyse des performances a montré qu'il était possible de rendre le processus de traitement compatible avec des applications temps réel. L'utilisation d'une version multicœur permet de réduire très fortement le temps de traitement nécessaire. Cette approche peut facilement être transposée dans une version adaptée à une utilisation sur le *cloud*.

Conclusion de la partie III

Ce chapitre conclut la troisième partie de ce manuscrit. Celle-ci a eu pour rôle de détailler les différentes contributions apportées. Les tests réalisés ont permis de mettre en évidence le bienfait d'un encodage sélectif, tant d'un point de vue de la bande passante nécessaire que du point de vue des utilisateurs du service. Le cas d'usage du *cloud gaming* est important, car il implique une carte d'attention visuelle spécifique. Comme la démonstration en a été faite, ces cartes d'attention visuelle sont uniquement dépendantes du jeu utilisé. Cette approche peut toutefois être généralisée à d'autres types de flux vidéos voués à être utilisés dans des solutions de rendu à distance à condition qu'il soit possible d'extraire une carte d'attention. La notion d'encodage multimodal, à travers l'implémentation d'un transmodeur a aussi montré tout son potentiel. Cette implémentation – qu'il faut plus considérer comme une preuve de concept – montre déjà qu'il est possible de réduire le débit binaire. En améliorant les points clés de l'algorithme, il est certain que des gains bien plus importants sont atteignables, tant du point de vue de la compacité que du temps nécessaire à la compression. Le surcoût temporel peut être nettement réduit d'une part en limitant le nombre de zones à traiter, d'autre part en utilisant une approche *multithreading*.

Quatrième partie

Conclusion

Chapitre 12

Contributions & perspectives

Dans cette thèse, nous proposons un système de compression vidéo hybride, basé sur la notion de modalité d'encodage. Chaque modalité, elle même constituée de plusieurs zones est traitée par un encodeur spécialisé, ce qui permet une réduction du débit binaire. Les contributions apportées par ce travail sont présentées de manière concise ci-dessous. Un paragraphe évoque les perspectives possibles et clôture ce manuscrit.

12.1 Contributions

L'objectif de compresser un flux vidéo en utilisant plusieurs encodeurs de différente nature nous a amenés à deux questions principales :

1. Comment gérer les zones sur lesquelles vont respectivement s'appliquer des encodeurs différents ?
2. Dans le cas d'usage du *cloud gaming*, le joueur perçoit-il l'intégralité des images avec la même attention ?

La réponse à la première question passe par l'utilisation de rectangles. Plus précisément par la décomposition d'une forme en une suite de rectangles ; dans ce cadre, une nouvelle heuristique a été proposée. À cette première question, l'intégralité du transmodeur est aussi à prendre en compte. Des tests objectifs ont permis de mettre en avant la capacité de cette approche à réduire le débit binaire.

La seconde question trouve sa réponse par le biais d'une expérimentation réalisée avec plusieurs joueurs, sur plusieurs jeux. À partir de ces tests, il est possible de tirer les conclusions suivantes :

a. Pour un joueur et un jeu donné, les cartes d'AVU (issues de différentes sessions de test) sont corrélées ;
b. Pour un jeu donné, toutes (pour tous les joueurs et tous les tests) les cartes d'AVU sont corrélées.

Ces deux résultats permettent d'affirmer que la carte d'attention visuelle d'un jeu est indépendante des joueurs. Il est donc possible de déterminer et de discriminer les zones importantes des zones bien moins importantes aux yeux du joueur. Cet aspect a notamment été mis en œuvre dans le transmodeur au moyen de différents profils, modélisant la carte d'attention pour un jeu donné.

En outre – même si des tests supplémentaires seraient certainement nécessaires – la modification des modalités de compression dans les zones de moindre importance (et selon les cartes d'attention visuelle) permet de réduire de manière importante le débit binaire nécessaire au transfert du flux vidéo sans pour autant impacter de manière notable la qualité perçue par le joueur. Au final, on peut dire que notre système de compression déplace l'erreur dans les zones qui importent peu à l'utilisateur du service en impactant très faiblement sa perception.

Les travaux présentés dans cette thèse ont fait l'objet d'une présentation à l'INSA [1] de Lyon [132]. Un article [133] dans une conférence internationale à comité de lecture fait état de ces travaux, il a obtenu le *best student paper award* dans sa catégorie. Une version plus détaillée a été acceptée pour publication dans un journal international [134]. La nouvelle heuristique proposée permettant la décomposition d'une matrice binaire en un ensemble de rectangles a fait l'objet d'un article [138]. Les détails des présentations et publications sont fournis dans l'annexe H.

12.2 Perspectives

La définition du transmodeur est générique, son implémentation actuelle souffre cependant de quelques lacunes. Celle-ci a néanmoins permis la réalisation d'une série de tests mettant en avant les avantages d'une compression multimodale. De nombreux axes d'améliorations sont possibles et en premier lieu par rapport à la gestion des différentes modalités. Il est envisageable d'utiliser bien plus de 2 modalités, ce qui implique une refonte du module de construction des espaces. La qualité des encodeurs utilisés est aussi un aspect important et il faut s'assurer que chaque encodeur est bien en charge d'une partie de l'image qui lui convient particulièrement bien. Les performances du transmodeur peuvent largement être améliorées en adaptant son architecture au *cloud*, comme l'a montré sa version *multithreading*.

Comme le chapitre 11 l'a mis en avant, les valeurs des paramètres à utiliser lors de l'encodage multimodal ont une influence forte sur les résultats. Une gestion plus poussée de leurs valeurs aurait sans doute pour conséquence une diminution plus prononcée du débit binaire tout en limitant au maximum l'augmentation du débit binaire engendré par la compression d'une vidéo peu adaptée à notre système de compression.

L'évaluation de la qualité a été réalisée grâce à une métrique objective, à savoir le PSNR. Les contributions (partie III) apportées par ce travail décrivent et utilisent la notion d'attention. Cet aspect subjectif se doit d'être pris en compte grâce à l'utilisation d'une métrique subjective permettant d'évaluer la qualité perçue. L'utilisation des deux métriques précédemment citées était pré-

1. *Institut National des Sciences Appliquées*

vue, tout comme leur confrontation. Cette partie n'a malheureusement pas pu être réalisée.

De manière générale, plus de tests sont sans doute nécessaires pour permettre une évaluation complète du système proposé. L'analyse des résultats s'est cantonnée au jeu Doom3, mais la création d'un modèle d'attention pour 0 A.D. est tout à fait envisageable. Évidemment, il serait intéressant de voir si notre système se comporte de la même manière pour ces deux jeux, intrinsèquement différents.

Du côté client, il est possible d'ajouter un post-traitement agissant juste après le décodage et la fusion des différentes modalités aboutissant à la reconstitution d'une image complète.

Cinquième partie

Annexes

Annexe A

Détection et suivi des mouvements oculaires

L'oculométrie regroupe l'ensemble des techniques permettant d'apprécier, de suivre et d'enregistrer les mouvements oculaires. Pour comprendre les raisonnements utilisés par l'étude du mouvement des yeux, quelques rappels au sujet du système de vision humain sont nécessaires. Les sections suivantes proposent une courte explication des termes importants et des caractéristiques de la vision humaine permettant aux systèmes d'*eye-tracker* de fonctionner. Les informations présentes dans les paragraphes suivants sont en partie issues du site Web de Tobii [1] qui est l'un des principaux fournisseurs de solutions d'*eye-tracking*.

A.1 Fonctionnement de l'œil

Nos yeux ont beaucoup de similarités avec le fonctionnement des caméras : la lumière reflétée par un objet ou une scène parvient à nos yeux en traversant une lentille. Cette lentille concentre et projette la lumière sur une surface sensitive positionnée au fond d'une chambre close. Cependant, et contrairement à une caméra, la surface sensible à la lumière (la rétine dans l'œil) ne dispose pas d'une sensibilité uniforme. Au fil de l'évolution, nos yeux ont été adaptés pour fonctionner à la fois dans des environnements sombres et lumineux. Ils sont capables de fournir un maximum de détails et de changer rapidement de cible. Ceci a conduit à certains compromis, parmi lesquels le fait de ne pouvoir appréhender un maximum de détails que dans une zone limitée du champ de vision, la zone fovéale. La plus grande partie de notre champ de vision (la zone périphérique) est bien mieux adaptée à la vision dans des conditions de faible luminosité. Elle permet de détecter les mouvements, les contrastes et les formes. L'image produite dans cette zone est plus floue et moins colorée (par rapport à une image produite dans la zone fovéale). Entre ces deux zones se trouve une région de transition (la zone parafovéale), dans laquelle l'image devient graduellement plus floue, en allant de la zone fovéale à la zone périphérique.

Les différences de perception de notre champ de vision sont dues à la nature des récepteurs photosensibles. Un œil en possède deux différents : les bâtonnets

1. `http://www.tobii.com`

et les cônes. Approximativement 94% des récepteurs sont des bâtonnets. Comme précédemment évoqué, la zone périphérique de la rétine n'est pas adaptée à une restitution précise des couleurs et des détails. L'explication vient du fait que cette zone est majoritairement couverte de bâtonnets. Ceux-ci n'ont pas besoin de beaucoup de lumière pour fonctionner, mais fournissent seulement une image peu précise et faiblement colorée. Pour obtenir plus de détail et une image nette, nos yeux sont aussi équipés de cônes, qui représentent seulement 6% du nombre total des cellules photosensibles. Les cônes sont – dans l'œil humain – présents sous trois formes différentes : certains sont sensibles au bleu, d'autres au vert et d'autres encore au rouge. S'ils sont performants en offrant une image nette et précise, ils nécessitent beaucoup plus de lumière pour fonctionner. Ainsi, lorsque l'on regarde un objet dans un endroit sombre, ce sont les bâtonnets qui sont principalement mis en œuvre, expliquant de fait que l'image soit plutôt peu colorée, voire en niveaux de gris. Les cônes sont surtout présents dans la fovéa où leur densité est importante permettant de restituer une image aussi colorée et précise que possible.

Le système de vision humain couvre environ 220° et est, comme précédemment mentionné, composé de trois zones : la zone fovéale, la zone parafovéale et la zone périphérique. Nous enregistrons principalement les données visuelles à partir de la région fovéale, qui représente environ 6% du champ visuel. Malgré le fait qu'elle ne représente qu'une petite partie de notre champ de vision, les informations en provenance de la région fovéale constituent 50% de ce qui est envoyé au cerveau grâce au nerf optique. La vision périphérique possède une très mauvaise acuité et est seulement utilisée pour détecter les mouvements et les changements de contrastes. Ainsi, lorsque l'on se déplace et que nos yeux se concentrent sur une région spécifique d'une scène, les mouvements de l'œil permettent de positionner cette région dans la zone fovéale. Cela signifie que les ressources de traitement sont adaptées en fonction du type de capteur utilisé par la rétine. En laissant la région fovéale capturer l'image, le cerveau obtient un maximum d'informations à traiter sur la zone en question, lui permettant d'interpréter une image de qualité optimale.

Dans le processus de visualisation, on considère que les mouvements de l'œil répondent à trois fonctions principales :

1. Placer la zone ciblée d'une scène dans la fovéa ;
2. Garder l'image stationnaire dans la rétine, en dépit des mouvements de l'objet suivi ou de l'observateur lui-même ;
3. Empêcher les objets fixes de ternir la perception.

A.2 Fonctionnement de l'*eye-tracker*

L'oculométrie est connue et utilisée depuis longtemps en tant que méthode permettant d'étudier l'attention des personnes. Il existe plusieurs techniques différentes permettant de détecter et de suivre les mouvements des yeux. Cependant, lorsque l'on souhaite un capteur à distance non intrusif, la technique le plus souvent utilisée est la PCCR [2]. Le concept de base est d'utiliser une source de lumière pour illuminer les yeux, ce qui provoque une réflexion nettement

2. *Pupil Centre Corneal Reflection*

visible. Une caméra capture alors une image des yeux montrant ces réflexions. L'image capturée par cette caméra est ensuite utilisée pour identifier la réflexion de cette source de lumière sur la cornée (reflet) et sur la pupille. Il est ensuite possible de calculer le vecteur formé par l'angle entre les réflexions de la cornée et de la pupille. La direction de ce vecteur, combinée avec d'autres caractéristiques géométriques, permet de calculer la direction du regard. Les *eye-tracker* Tobii utilisent une version améliorée de la technologie PCCR, faisant l'objet d'un brevet [57]. Une lumière dans le proche infrarouge est utilisée pour créer les réflexions sur la cornée et la pupille de l'œil de l'utilisateur. Deux capteurs permettent ensuite de capturer l'image des yeux et les réflexions associées. Des traitements de l'image, appuyés par une modélisation physiologique de l'œil en 3D sont mis en œuvre pour estimer avec précision la position dans l'espace ainsi que le point visé par le regard.

FIGURE A.1 – L'*eye-tracker* utilisé pour les tests d'attention sélective, modèle X1 de Tobii.

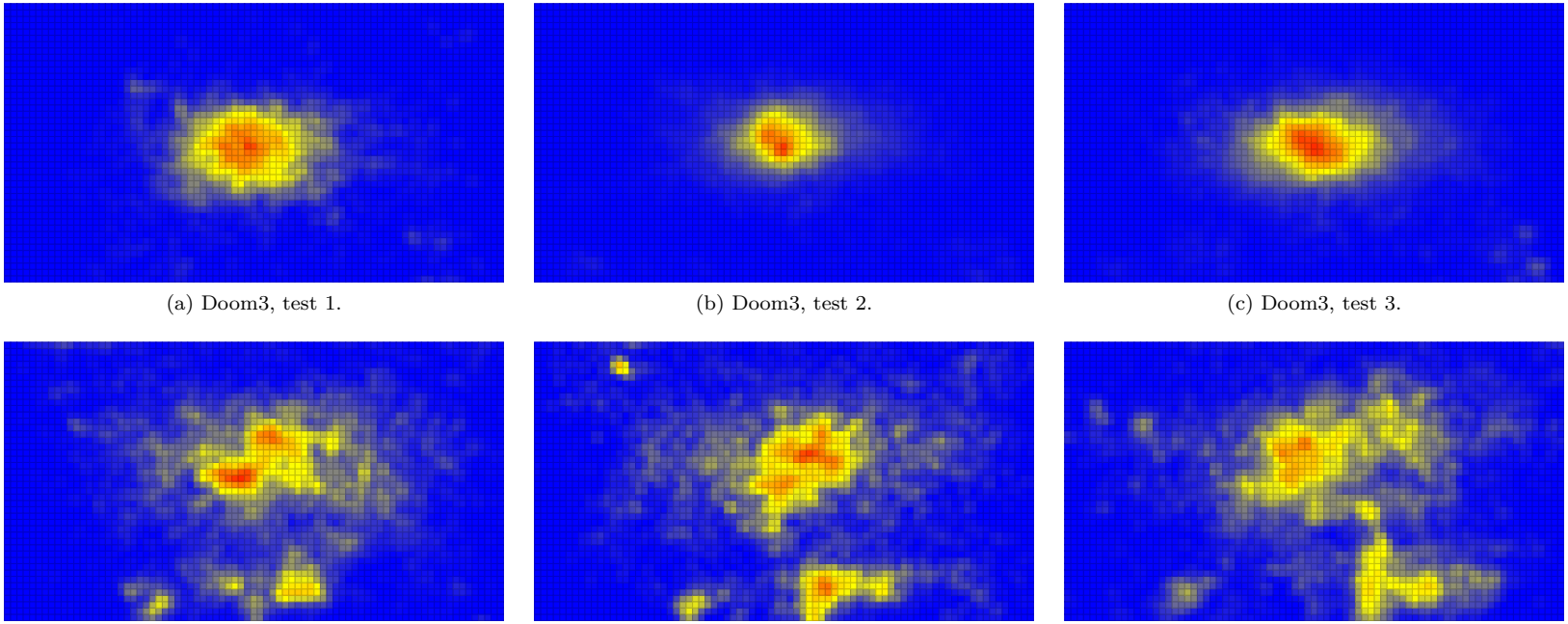
Avant toute acquisition de données grâce à l'*eye-tracker*, chaque utilisateur doit réaliser une procédure de calibration. Durant cette procédure l'*eye-tracker* mesure les caractéristiques des yeux de l'utilisateur et les utilise en complément du modèle physiologique pour calculer les points regardés. Ce modèle prend en compte les formes, les propriétés de réfraction et de réflexion des différentes parties de l'œil. Durant la calibration, il est demandé à l'utilisateur de regarder des points spécifiques à l'écran, nommés points de calibration. Durant le processus, plusieurs images des yeux sont capturées et analysées. Les informations en résultant sont alors intégrées au modèle et les points regardés sont calculés pour chaque image de test. Une fois la procédure terminée, la qualité de calibration peut être appréciée grâce à des lignes vertes de longueur variable. La longueur de ces lignes représente la distance entre le point regardé (calculé par le modèle) et le centre du point de calibration (référence).

Annexe B

Données des tests utilisateurs

Cette annexe contient les données de l'ensemble des tests réalisés lors de la phase d'expérimentation avec l'*eye-tracker* et les deux jeux. Les cartes de chaleur générées dans le but d'évaluer les résultats ainsi que les différentes métriques calculées sont présentées.

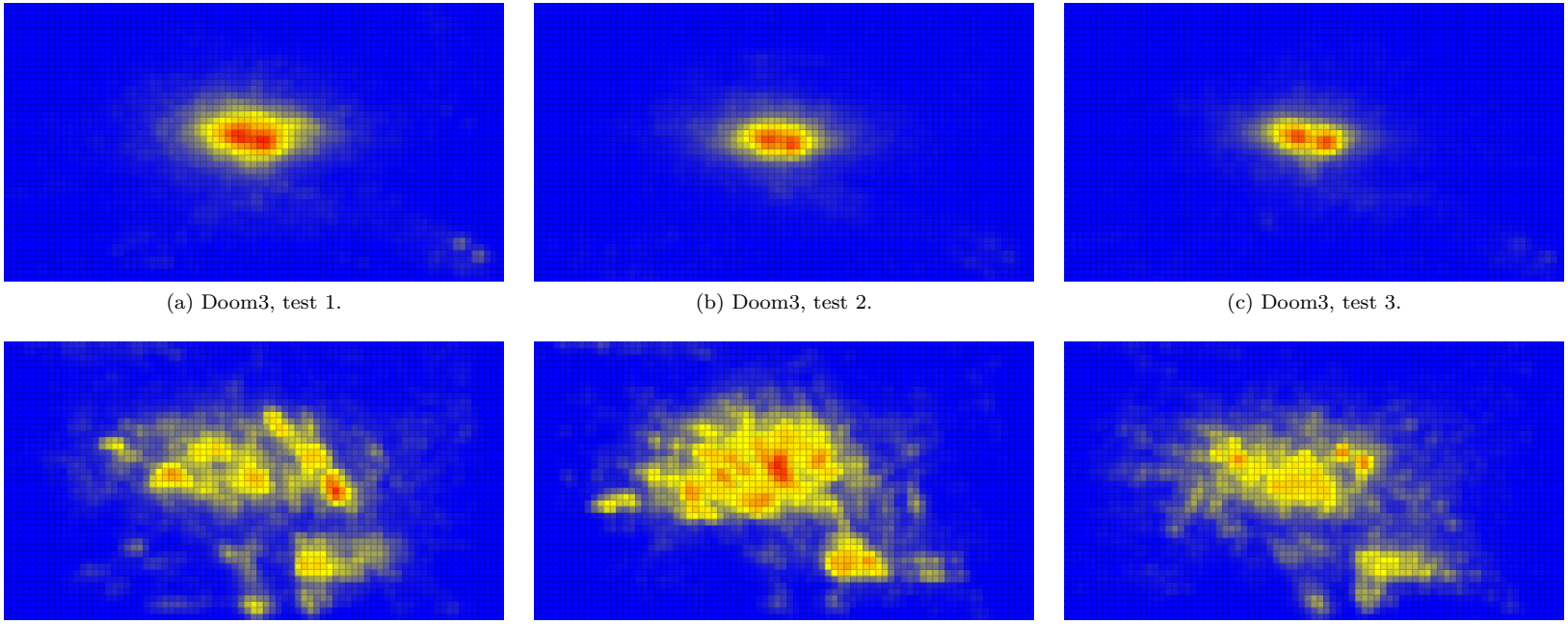
B.1 Cartes de chaleur pour chaque session de jeu



(a) Doom3, test 1. (b) Doom3, test 2. (c) Doom3, test 3.

(d) 0 A.D., test 1. (e) 0 A.D., test 2. (f) 0 A.D., test 3.

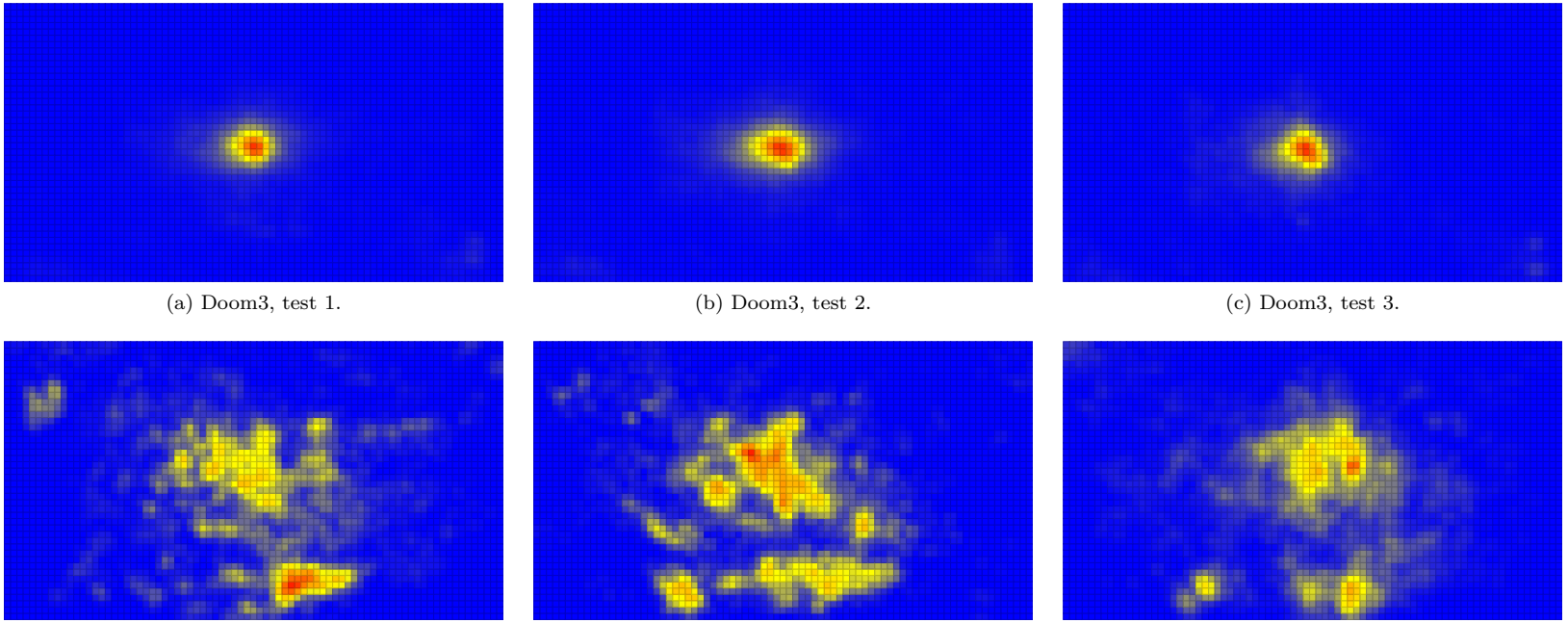
FIGURE B.1 – Cartes de chaleur obtenues par le testeur 0.



(a) Doom3, test 1. (b) Doom3, test 2. (c) Doom3, test 3.

(d) 0 A.D., test 1. (e) 0 A.D., test 2. (f) 0 A.D., test 3.

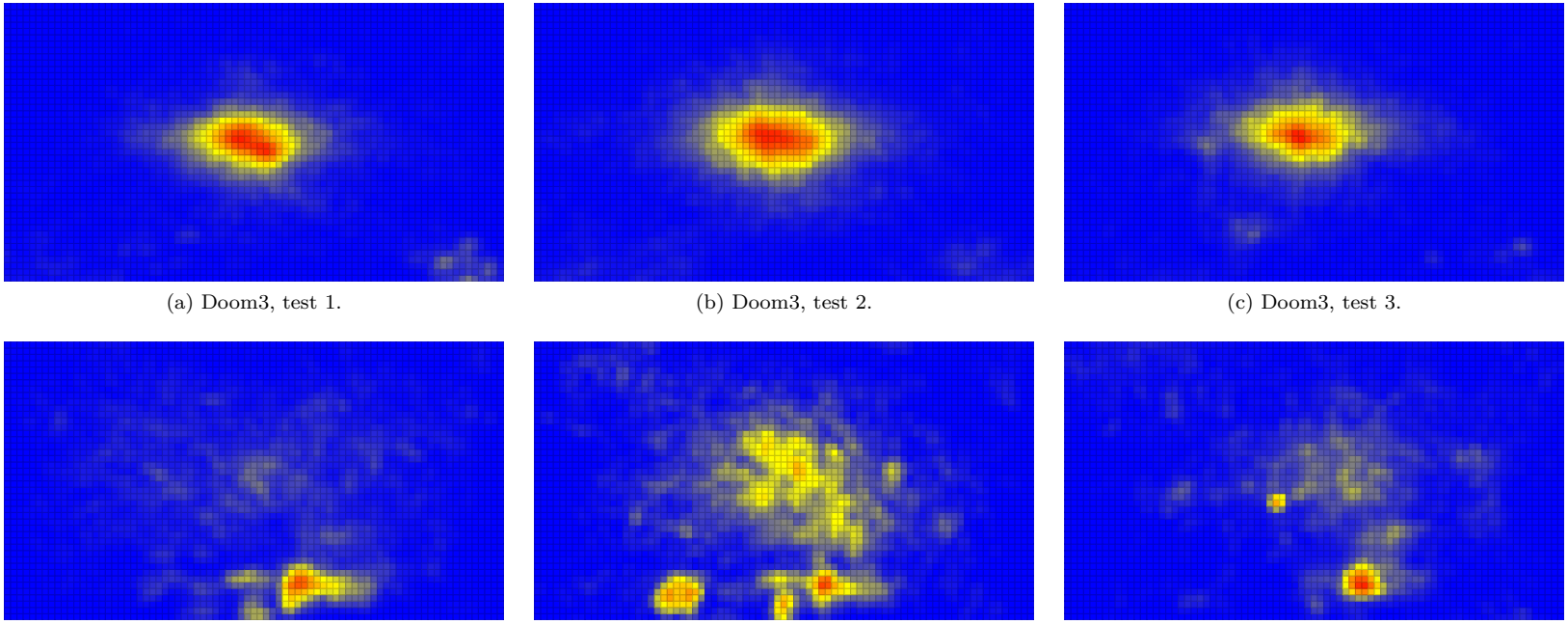
FIGURE B.2 – Cartes de chaleur obtenues par le testeur 1.



(a) Doom3, test 1. (b) Doom3, test 2. (c) Doom3, test 3.

(d) 0 A.D., test 1. (e) 0 A.D., test 2. (f) 0 A.D., test 3.

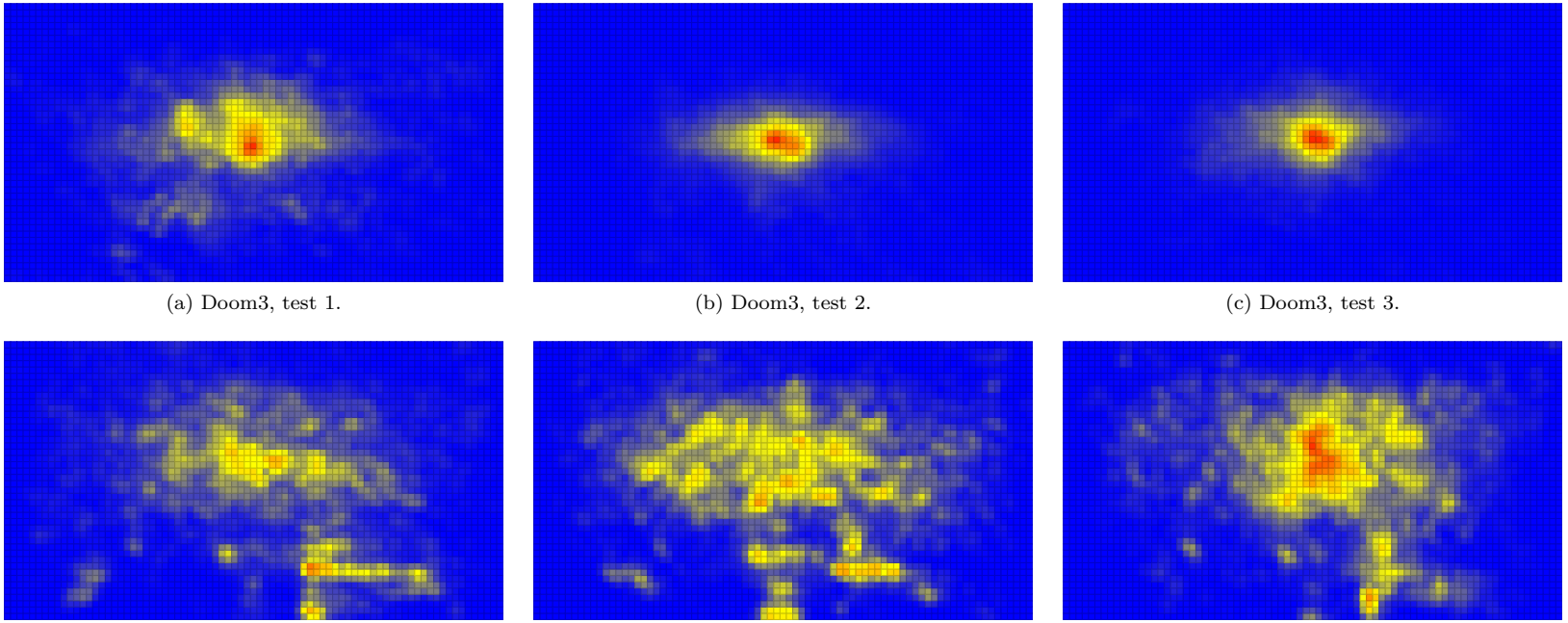
FIGURE B.3 – Cartes de chaleur obtenues par le testeur 2.



(a) Doom3, test 1. (b) Doom3, test 2. (c) Doom3, test 3.

(d) 0 A.D., test 1. (e) 0 A.D., test 2. (f) 0 A.D., test 3.

FIGURE B.4 – Cartes de chaleur obtenues par le testeur 3.

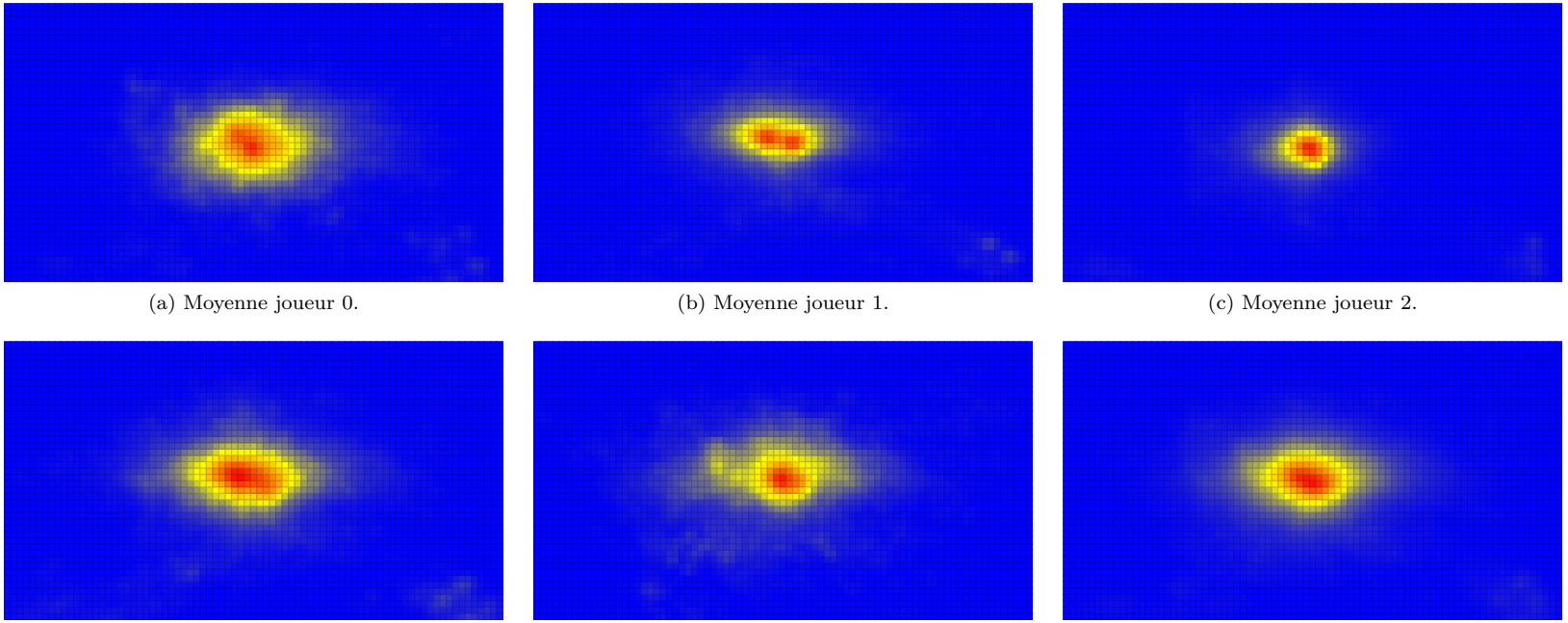


(a) Doom3, test 1. (b) Doom3, test 2. (c) Doom3, test 3.

(d) 0 A.D., test 1. (e) 0 A.D., test 2. (f) 0 A.D., test 3.

FIGURE B.5 – Cartes de chaleur obtenues par le testeur 4.

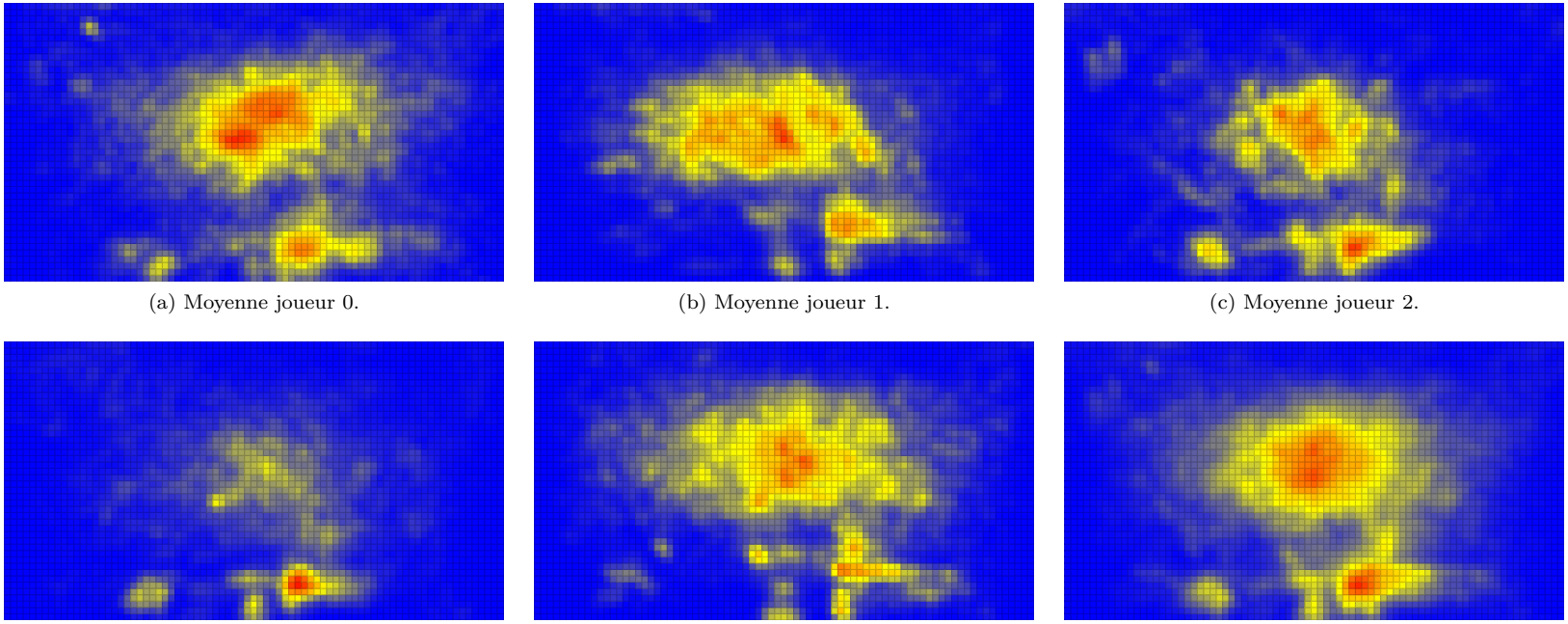
B.2 Cartes de chaleur par joueur et par jeu



(a) Moyenne joueur 0. (b) Moyenne joueur 1. (c) Moyenne joueur 2.

(d) Moyenne joueur 3. (e) Moyenne joueur 4. (f) Moyenne englobant tous les joueurs.

FIGURE B.6 – Cartes de chaleur moyennes obtenues par les joueurs sur le jeu Doom3.



(a) Moyenne joueur 0. (b) Moyenne joueur 1. (c) Moyenne joueur 2.

(d) Moyenne joueur 3. (e) Moyenne joueur 4. (f) Moyenne englobant tous les joueurs.

FIGURE B.7 – Cartes de chaleur moyennes obtenues par les joueurs sur le jeu 0 A.D..

B.3 Corrélation entre tests et utilisateurs

TABLE B.1 – Coefficients de corrélation (en rouge) et distances de Jensen-Shannon obtenus pour tous les tests réalisés avec Doom3.

		Testeur 0			**Testeur** 1			**Testeur** 2			**Testeur** 3			**Testeur** 4		
		S0	S1	S2	S0	S1	S2	S0	S1	S2	S0	S1	S2	S0	S1	S2
Testeur 0	S0	N/A	0,88	0,92												
	S1	0,27	N/A	0,94		0,91			0,87			0,96			0,94	
	S2	0,25	0,20	N/A												
Testeur 1	S0				N/A	0,95	0,95									
	S1		0,21		0,21	N/A	0,96		0,80			0.96			0,93	
	S2				0,22	0,21	N/A									
Testeur 2	S0							N/A	0,97	0,89						
	S1		0,26			0,28		0.19	N/A	0,94		0,80			0,83	
	S2							0.27	0,23	N/A						
Testeur 3	S0										N/A	0,95	0,91			
	S1		0,19			0.16			0,27		0,21	N/A	0,95		0,95	
	S2										0,26	0,21	N/A			
Testeur 4	S0													N/A	0,85	0,84
	S1		0,21			0.21			0.31			0,19		0,31	N/A	0,93
	S2													0,30	0,22	N/A

TABLE B.2 – Coefficients de corrélation (en rouge) et distances de Jensen-Shannon obtenus pour tous les tests réalisés avec 0 A.D..

		Testeur 0			**Testeur** 1			**Testeur** 2			**Testeur** 3			**Testeur** 4		
		S0	S1	S2	S0	S1	S2	S0	S1	S2	S0	S1	S2	S0	S1	S2
Testeur 0	S0	N/A	0,70	0,64	0,73			0,80			0,66			0,77		
	S1	0,35	N/A	0,59												
	S2	0,37	0,38	N/A												
Testeur 1	S0	0,30			N/A	0,69	0,69	0,70			0,49			0,81		
	S1				0,32	N/A	0,77									
	S2				0,30	0,29	N/A									
Testeur 2	S0	0,29			0,32			N/A	0,64	0,62	0,76			0,66		
	S1							0,40	N/A	0,64						
	S2							0,39	0,38	N/A						
Testeur 3	S0	0,28			0,34			0,27			N/A	0,57	0,74	0,48		
	S1										0,38	N/A	0,60			
	S2										0,37	0,37	N/A			
Testeur 4	S0	0,26			0,24			0,35			0,34			N/A	0,66	0,68
	S1													0,32	N/A	0,65
	S2													0,30	0,32	N/A

Annexe C

Schéma global du transmodeur

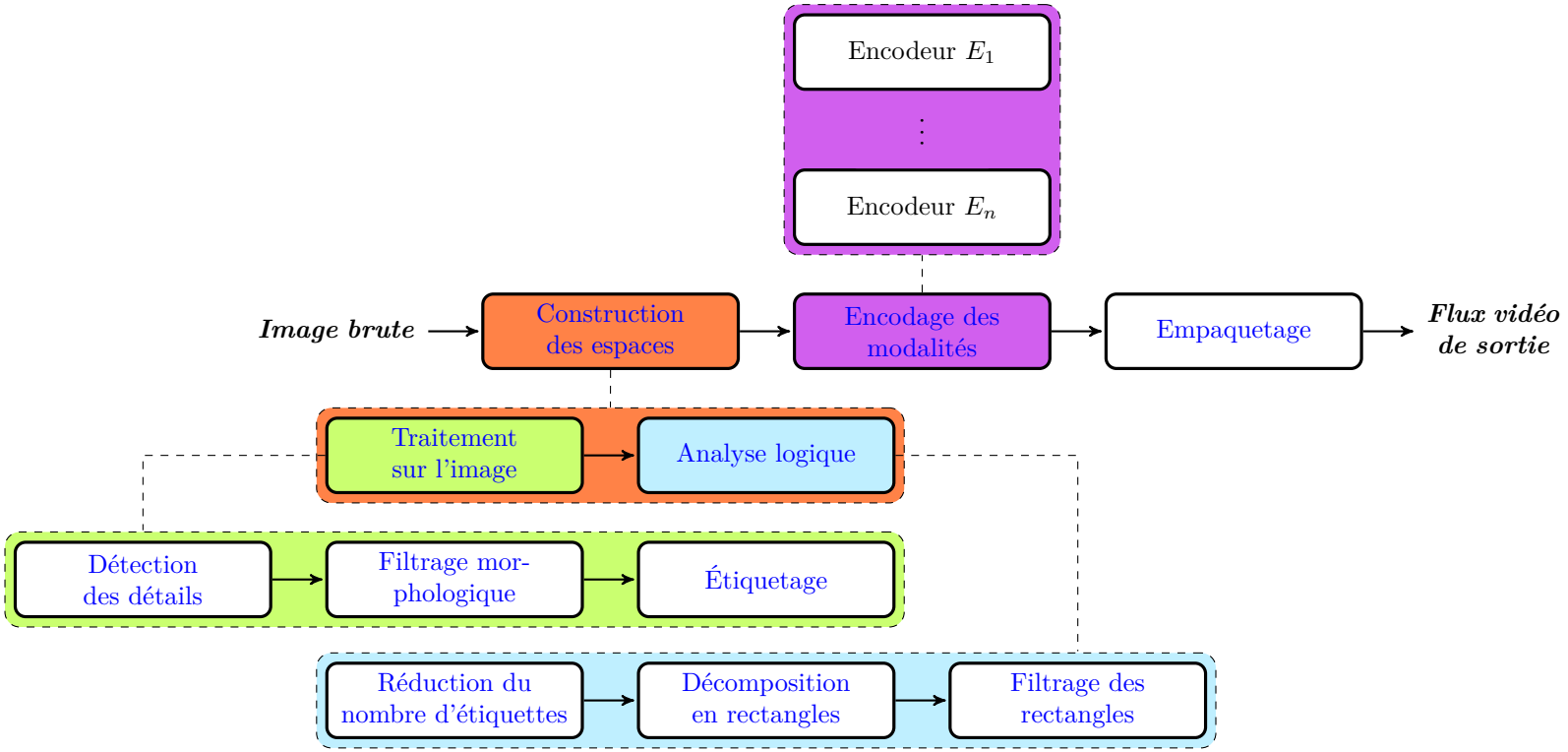


FIGURE C.1 – Vue synoptique globale du fonctionnement du transmodeur.

Annexe D

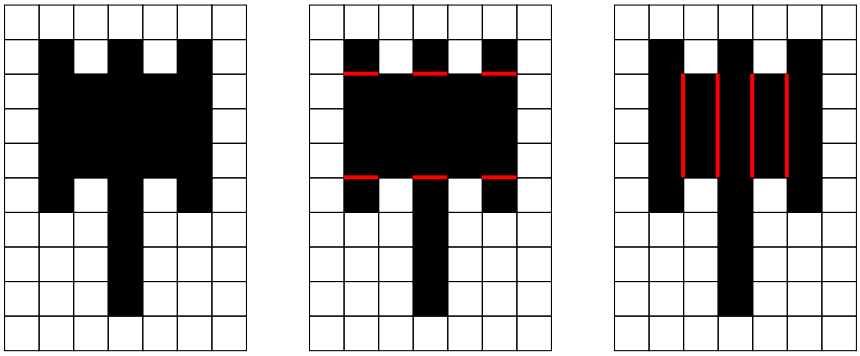
Problème DBMR

Ce problème a été rapidement évoqué dans le chapitre 9. Après une définition formelle du problème et la présentation des heuristiques existantes, cette annexe détaille le fonctionnement de l'algorithme implémenté dans le transmodeur.

D.1 Décomposition de rectangles en polygones rectilinéaires

Le problème de la décomposition des $1-$*entrées* d'une matrice binaire en rectangles (c'est-à-dire le problème DBMR) est le fait de trouver la « meilleure » décomposition d'une matrice binaire en un ensemble de rectangles qui forment une partition de ces $1-$*entrées*. La notion de « meilleure » décomposition dépend de l'application. Il y a deux variantes principales à ce problème. La première considère la meilleure décomposition comme celle qui minimise le nombre de lignes utilisées pour décomposer le polygone rectilinéaire formé par les $1-$*entrées* (problème DBMR-*MinLines*) ou, ce qui est équivalent, comme la décomposition minimisant la longueur des contours des rectangles. La seconde variante consiste à minimiser la cardinalité de l'ensemble des rectangles qui forme la décomposition (problème DBMR-*MinRect*). Le problème DBMR-*MinLines* fut d'abord étudié dans des applications VLSI [1]. L'auteur de [20] fournit un état de l'art complet traitant de la décomposition de matrices binaires. [115] apporte un algorithme en temps $\mathcal{O}(n^4)$ au problème DBMR-*MinLines*. Dans le même article, les auteurs ont montré que si le polygone rectilinéaire contient des trous, le problème devient NP-complet, même en se limitant au cas où les trous sont limités à des points. Plus tard, [52] proposa une approximation en temps $\mathcal{O}(n^2)$, et en temps $\mathcal{O}(n \times log(n))$ dans [113]. L'objectif d'optimisation diffère entre les problèmes DBMR-*MinLines* et DBMR-*MinRect*. Une illustration de ces différences est présentée dans la figure D.1. Dans notre cas d'usage, nous nous focalisons sur une heuristique linéaire faisant partie de la famille DBMR-*MinRect*. La section suivante expose les différentes solutions connues permettant la résolution de ce problème.

1. *Very-Large-Scale Integration*



(a) Le polygone rectilinéaire utilisé comme référence et qui doit être décomposé.

(b) La solution optimale pour le problème DBMR-*MinLines* : 6 unités et 7 rectangles.

(c) La solution optimale pour le problème DBMR-*MinRect* : 5 rectangles et 12 unités.

FIGURE D.1 – Les solutions optimales aux problèmes DBMR-*MinLines* et DBMR-*MinRect* diffèrent dans la plupart des cas.

D.2 Le problème DBMR-*MinRect*

Sont présentés dans cette section la définition du problème, la solution optimale et finalement les heuristiques solvant le problème DBMR-*MinRect*.

D.2.1 Définition

Considérons M comme une matrice binaire ($M_{i,j} = 0$ ou 1) de taille $m \times n$. On pose $\mathcal{H} = \{1, 2, \ldots, n-1, n\}$ et $\mathcal{W} = \{1, 2, \ldots, m-1, m\}$. Un sous ensemble non vide $R_{t,l,b,r}$ de M est un rectangle (un ensemble de $(i,j) \in \mathcal{W} \times \mathcal{H}$) s'il existe $l, r \in \mathcal{W}$ et $b, t \in \mathcal{H}$ tels que :

$$\begin{cases} R_{t,l,b,r} = \{(i,j) : b \leq i \leq t, l \leq j \leq r\} \\ \sum_{i=l}^{r} \sum_{j=b}^{t} R[i][j] = (|r-l|+1) \times (|b-t|+1) \end{cases} \tag{D.1}$$

Pour chaque rectangle $R_{t,l,b,r}$, $R[t][l]$ (respectivement $R[b][r]$) est le coin du rectangle situé en haut à gauche (respectivement en bas à droite). Le problème DBMR-*MinRect* revient donc à trouver un ensemble minimal de rectangles $\mathcal{R}$, tel que $\mathcal{R}$ soit une partition de la matrice binaire M. Ce problème peut être formulé mathématiquement de la manière suivante :

$$argmin(|\mathcal{R}|) \; : \begin{cases} \mathcal{R} = \bigcup_{k=1}^{K} R^k, \text{avec } R^k \text{ un rectangle de } M \\ R_i \cap R_j = \emptyset \text{ pour } i \neq j \\ \sum_{k=1}^{K} \sum_{i=R_l^k l}^{R_r^k} \sum_{j=R_b^k}^{R_t^k} R^k[i][j] = \sum_{i=0}^{m} \sum_{j=0}^{n} M[i][j] \end{cases} \tag{D.2}$$

Une illustration (figure 9.11) du problème a déjà été présentée dans le chapitre 9. Il est important de noter que ce problème se focalise sur la recherche d'une partition minimale d'une matrice binaire en rectangles.

Il est important de faire la différence avec le problème de couverture par des rectangles [114] pour lequel la superposition de rectangles est possible.

La contribution de [12] se concentre sur une généralisation de ce problème. Dans ce travail, les auteurs se concentrent sur la décomposition d'une matrice A constituée de nombres réels non négatifs et de dimension $m \times n$ en une « combinaison non négative de matrices binaires ($M_{i,j} = 0$ ou 1) formant un rectangle de telle sorte que la somme des coefficients est minimale ». Si A est elle même une matrice binaire, les coefficients de chaque matrice dans l'ensemble des matrices correspondent à la décomposition minimale de A et sont égaux à 1. Le problème est alors équivalent à notre problème de décomposition, à savoir celui d'une matrice binaire en rectangles. Les auteurs proposent plusieurs algorithmes et les comparent avec les solutions de l'état de l'art dans le cas général, et notamment [11, 43, 44]. Tous les algorithmes présentés réalisent la décomposition en un temps $O(m^2n^2)$ pour le cas général (valeurs réelles non négatives), aucune étude n'étant fournie quant au cas particulier où A est une matrice binaire.

D.2.2 Solution optimale

La solution optimale pour des polygones rectilinéaires pouvant comporter des trous a été prouvée indépendamment par [14, 90, 24, 123]. Cette solution est donnée par le théorème D.2.1, les preuves sont données dans les travaux sus-cités.

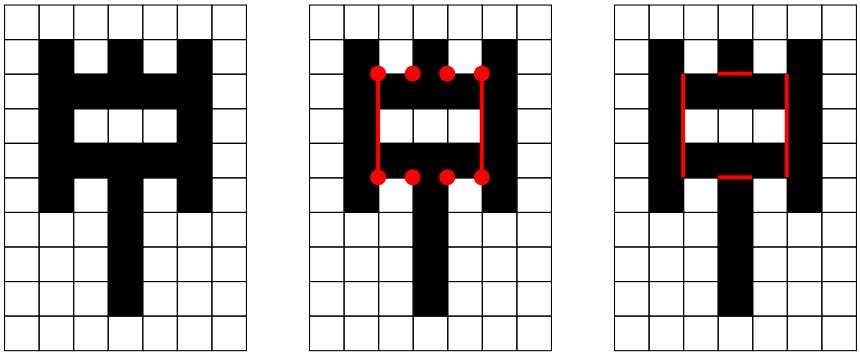
Théorème D.2.1. *Un polygone orthogonal peut être partitionné de manière minimale en $N - L - H + 1$ rectangles, N étant le nombre de sommets avec un angle intérieur supérieur à 180 degrés. H est le nombre de trous et L le nombre maximal de segments ne se coupant pas et pouvant être dessinés horizontalement ou verticalement entre des sommets opposés (théorème extrait et traduit de [90]).*

Corollaire D.2.1. *À partir de l'équation D.2 et du théorème D.2.1, on peut conclure que $argmin(|\mathcal{R}|) = N - L - H + 1$.*

Ce résultat n'est pas trivial et a été établi grâce à l'investigation du problème DBMR-*MinRect* dans le domaine du calcul géométrique [88]. La figure D.2 met en œuvre ce résultat en exposant les valeurs des différents paramètres présentés par le théorème D.2.1. Pour parfaitement refléter le fonctionnement de l'algorithme, un trou sous la forme d'un rectangle de dimension 1×3 a été ajouté à la matrice de référence précédemment utilisée dans la figure D.1.

Cependant, [88] fournit une preuve que l'algorithme optimal fait partie de la classe de complexité P. [14, 24, 123] ont exposé de façon indépendante un algorithme optimal en temps polynomial s'exécutant en $\mathcal{O}(n^{\frac{3}{2}} \times log(n))$. Nous faisons référence à cet algorithme amenant à la solution optimale sous l'acronyme BGD [2]. Le fonctionnement global de cet l'algorithme BGD peut se résumer par les trois étapes présentées ci-dessous :

2. *Bipartite Graph-based Decomposition*



(a) Le polygone rectilinéaire (avec un trou) utilisé comme référence et qui doit être décomposé.

(b) $N = 8$ sommets (cercles), et $L = 2$ segments horizontaux ou verticaux ne se coupant pas au maximum.

(c) La solution optimale pour le problème DBMR-*MinRect* fait état de $8 - 2 - 1 + 1 = 6$ rectangles.

FIGURE D.2 – Comment trouver le nombre optimal de rectangles pour le problème DBMR-*MinRect*.

1. Lister tous les segments pour les sommets concaves, et créer leurs graphes bipartis ;
2. Conserver les segments nécessaires à la décomposition du polygone (ensemble maximal de segments indépendants) ;
3. Choisir un segment dans une direction arbitraire pour les sous-polygones restants.

Malheureusement et comme dans le cas du problème DBMR-*MinLines*, la complexité des algorithmes ne permet pas leur utilisation dans des applications temps réel. C'est particulièrement le cas dans le domaine du traitement d'images, et par extension dans celui de la vidéo. En effet, la dimension des matrices n'est pas négligeable et le temps de traitement est limité tout au plus à une douzaine de millisecondes. La suite de ce chapitre s'efforce de trouver une heuristique permettant de répondre à ces besoins particuliers.

D.2.3 Heuristiques existantes

Depuis 1985, la plupart des efforts pour le développement d'heuristiques ont été réalisés par la communauté du traitement d'images. L'application majeure s'articule autour de la compression d'images binaires : il s'agit d'encoder une image en utilisant en ensemble $\mathcal{R}$ plutôt qu'un ensemble de pixels de dimensions $m \times n$. Pour cela de nouveaux algorithmes portant sur le problème DBMR-*MinRect* ont été développés. En 2012 les auteurs de [139] ont proposé une étude exhaustive basée sur les applications de traitement d'images. Les paragraphes suivants présentent les cinq heuristiques existantes.

RS

La méthode RS [3] est la plus ancienne [48], elle est quelquefois qualifiée de méthode delta (*delta method*) [139]. Elle consiste à segmenter la matrice ligne par ligne. Ainsi, les rectangles formant la décomposition ont une hauteur maximale de 1, qui impacte fortement la qualité de la décomposition. L'avantage principal est sa complexité linéaire. Cette approche est cependant obsolète depuis l'introduction de la méthode IBR qui en est une extension.

IBR

La méthode IBR est une amélioration de la méthode RS. Proposée par [39], elle est aussi connue comme la généralisation de la méthode delta (*generalized delta method*). Le principe suit celui de la méthode RS, à ceci près que les lignes adjacentes identiques sont fusionnées. Ainsi, si deux lignes consécutives contiennent des intervalles avec les mêmes limites, elles sont fusionnées. Bien que la complexité demeure linéaire et que cette méthode soit une amélioration de la méthode RS, elle génère toujours un nombre important de rectangles non nécessaires dans la plupart des cas. Cette approche est considérée par [139] comme la meilleure alternative lorsqu'une faible complexité prévaut sur une décomposition optimale.

QTD

L'approche QTD [4] est une décomposition hiérarchique dans laquelle la décomposition finale est sous la forme d'un *quadtree* (un graphe acyclique dont chaque nœud accepte quatre fils). L'algorithme considère qu'une matrice carrée lui est fournie en entrée. Si ce n'est pas le cas un remplissage avec des 0 est réalisé dans une dimension. Pour obtenir un *quadtree*, l'algorithme divise l'espace en quadrants, c'est-à-dire en quatre carrés de mêmes dimensions ne se chevauchant pas. Le processus est récursivement appliqué pour chaque quadrant qui ne contient pas exclusivement des $1 - entrées$. La représentation de la décomposition sous forme de *quadtree* permet la meilleure compacité. Dans le même temps, cette décomposition comporte seulement des carrés, ce qui est dans la plupart des cas une solution sous optimale.

MED

La méthode MED [5] s'appuie sur un masque habituellement de taille 3×3 qui est appliqué sur tous les pixels. Si les voisins 3×3 d'un pixel donné contiennent uniquement des $1 - entrées$ ce pixel reste à 1, sinon il est positionné à 0. L'algorithme érode un polygone rectilinéaire jusqu'à ce que tous les pixels soient à zéro dans la matrice binaire. Quand cela arrive, les derniers pixels de l'itération précédente sont les centres de carrés inscrits de dimensions $(2n-1) \times (2n-1)$ où n est le nombre d'itérations. Cette approche utilise un algorithme complexe, qui n'est pas adapté aux applications temps réel.

3. *Row Segmentation*
4. *Quad Tree Decomposition*
5. *Morphological Decomposition*

DTD

L'algorithme DTD [6] a été proposé par [139], il calcule pour chaque pixel la distance maximale jusqu'aux frontières horizontales et verticales. On peut le considérer comme une extension de l'approche MED puisque les pixels contenant les valeurs maximales sont potentiellement les centres de carrés inscrits. Il fournit des résultats similaires à la méthode MED tout en présentant une complexité moindre.

D.2.4 Bilan

Le tableau D.1 permet une vue d'ensemble des classes de complexité, mais aussi des avantages et inconvénients de toutes les approches présentées. La figure D.3 illustre quant à elle les résultats obtenus avec chaque méthode.

TABLE D.1 – Résumé des différentes approches permettant de résoudre le problème DBMR-*MinRect*. Toutes les méthodes sont des heuristiques à l'exception de la méthode BGD qui résout le problème de façon optimale.

Méthode	**Complexité**	**Analyse**
RS	$\mathcal{O}(n)$	très rapide, rectangles de hauteur 1
IBR	$\mathcal{O}(n)$	très rapide, sous optimale
QTD	$\mathcal{O}(n \times log_2(n))$	très compacte, uniquement des carrés
MED	$\mathcal{O}(n^2)$	très lente, sous optimale
DTD	$\mathcal{O}(n^2)$	lente, sous optimale
BGD	$\mathcal{O}(n^{\frac{3}{2}} \times log(n))$	complexité importante, solution optimale

D.3 Approximation de la plus grande surface pour chaque point

L'idée sous-jacente de notre algorithme est d'effectuer un parcours de la matrice en supprimant successivement les plus grands rectangles découverts. Pour cela, le *walker* (voir D.4) nécessite une matrice contenant le plus grand rectangle pour chaque point. Cette section présente une solution existante et ces limitations ainsi qu'un algorithme dérivé, parfaitement adapté à nos besoins. En considérant chaque point d'une matrice binaire comme le coin d'un rectangle (situé en haut et à gauche), l'objectif est de déterminer la taille du plus grand rectangle pour chaque point. La figure D.4 présente le résultat obtenu sur la matrice de référence.

D.4 *Walk, Stack, Remove & Merge*

Le principe de notre algorithme est de parcourir la matrice en supprimant successivement les rectangles découverts. Le parcours débute en haut et à gauche de la matrice. Chaque fois qu'un rectangle est découvert par le *walker* (à partir du prétraitement décrit dans la section suivante qui calcule le plus grand

6. *Distance-Transform Decomposition*

rectangle pour chaque point de la matrice), il est parcouru entièrement pour s'assurer que ses points sont présents dans la matrice. Si un rectangle plus grand est rencontré, le rectangle précédent est enregistré dans une pile, ainsi que la position du *walker*. Si le rectangle est finalement validé, l'intégralité des points le constituant est supprimée de la matrice, et le parcours redémarre à la position enregistrée avant l'empilement du rectangle. Le processus se termine quand la matrice est vide. L'algorithme est constitué de quatre opérations principales ; celles-ci sont décrites ci-dessous :

1. *walk* : le parcours s'effectue de gauche à droite et de haut en bas. Le *walker* s'assure de la présence de chaque point du rectangle courant. Si un point est manquant, les dimensions du rectangle courant sont mises à jour en conséquence ;
2. *stack* : si le *walker* atteint un point qui est le début d'un rectangle plus grand que l'actuel, ce dernier est enregistré dans une pile et sa taille mise à jour ;
3. *remove* : lorsque le rectangle courant a été validé par le *walker*, il est supprimé de la matrice ;
4. *merge* : cette étape a lieu uniquement quand les deux derniers rectangles supprimés sont adaptés. Pour assurer la linéarité, les rectangles candidats sont limités aux deux derniers rectangles supprimés.

L'algorithme D.1 présente la boucle principale de l'algorithme WSRM. Le fonctionnement détaillé de l'algorithme WSRM est précisé au moyen d'un exemple dans [138].

```
   Données : Matrice binaire M
   Résultat : Liste L de rectangles r_i ne se chevauchant pas tel que ∪_i r_i = M
 1 début
 2     stack ← ∅, Rectangles ← ∅, aire ← 0
 3     répéter
 4         si stack = ∅ alors
 5             stackAndWalk()
 6         alors
 7             si pointCourant_aire ≤ aire alors
 8                 si le point courant est contigu avec le point précédent alors
 9                     décrémenter la valeur du compteur
10                     si le rectangle courant est validé alors
11                         enlever le rectangle
12                     finsi
13                 alors
14                     resizeOrRemove()
15                 finsi
16             alors
17                 stackAndWalk()
18             finsi
19         finsi
20     jusqu'à M ≠ ∅
21 fin
```

Algorithme D.1 – Aperçu de l'algorithme WSRM.

```
1 début
2 |  stack d'un nouveau rectangle commençant au point courant;
3 |  aire ← aireCourrante;
4 |  walk jusqu'à l'élément suivant;
5 fin
```

Algorithme D.2 – Fonction *stackAndWalk*.

D.5 Implémentation

Une implémentation en Java [7] ainsi qu'une en `C++11` [8] sont disponibles.

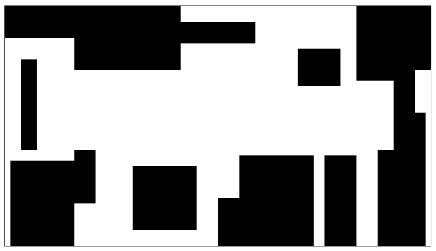
D.6 Évaluation

Nous avons évalué notre algorithme contre son adversaire direct, IBR qui est le seul à s'exécuter en temps linéaire tout en livrant de bons résultats. En effet, RS donne en comparaison d'IBR de mauvais résultats. Une comparaison globale des différents algorithmes est documentée dans [139]. Dans le but d'évaluer les performances de notre algorithme, nous avons généré des polygones rectilinéaires aléatoires avec un nombre de côtés fixé grâce à l'approche proposée dans [144]. En utilisant le code source que les auteurs nous ont procuré, nous avons généré des polygones rectilinéaires aléatoires pour un nombre de côtés allant de 8 à 25. Pour chacune de ces valeurs, 50000 polygones rectilinéaires ont été générés.

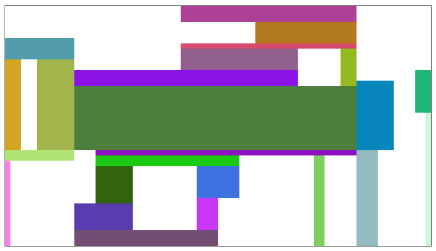
Dans une première expérimentation, nous avons comparé la qualité des deux approches, c'est-à-dire le nombre de rectangles découverts par chaque algorithme (WSRM par rapport à IBR). La figure D.5 montre la qualité de la décomposition en fonction du nombre de côté des polygones aléatoires. Lorsque le nombre de côté est inférieur à 22, la méthode WSRM est la meilleure dans 30% des cas. Les deux algorithmes se comportent de manière équivalente 60% du temps pour de petits polygones. Cette valeur diminue à 30% quand le nombre de côtés augmente, à ce moment-là c'est la méthode IBR qui devient la meilleure. Une distribution équivalente est atteinte pour des polygones de 22 côtés, après quoi c'est l'approche IBR qui fournit la meilleure décomposition.

La seconde expérimentation, présentée par la figure D.6 compare le ratio des côtés des rectangles du plus grand au plus petit pour les deux mêmes algorithmes en plus de la méthode QTD. L'algorithme IBR fonctionne en scannant les lignes ce qui permet de favoriser des valeurs de ratio élevé. La méthode QTD décompose de manière récursive la matrice en carrés, et donne donc toujours des carrés en sortie. Ce test a été réalisé de manière similaire au premier, déjà présenté. Comme attendu, la méthode IBR donne en sortie des rectangles comportant une valeur de ratio importante. La valeur est supérieure à 2 pour un polygone rectilinéaire de 8 côtés et dépasse 4 pour 25 côtés. De manière similaire, notre approche WSRM donne une décomposition dont le ratio des rectangles est plus petit : il évolue de $1,9$ à $2,5$. La méthode proposée propose sur ce critère une meilleure décomposition que l'approche IBR, quel que soit le nombre de côtés.

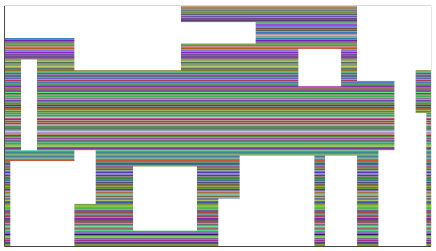
7. `https://github.com/jsubercaze/wsrm`
8. `https://bitbucket.org/trynitron/wasm`



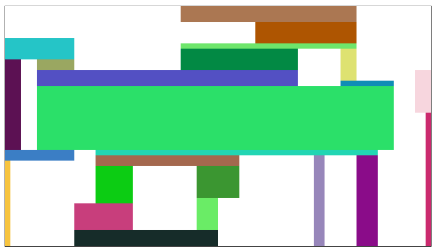
(a) Matrice binaire à traiter.



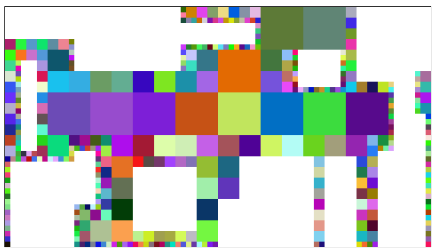
(b) BGD, 24 rectangles.



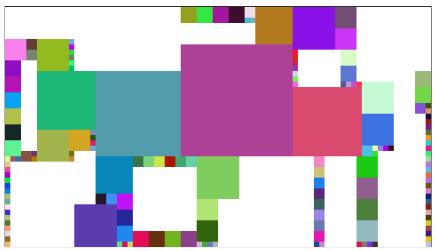
(c) RS, 1870 rectangles.



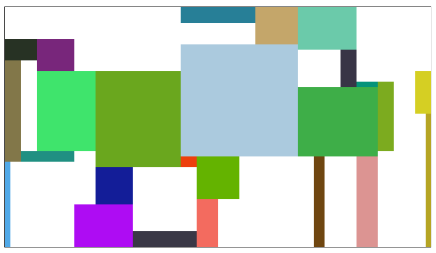
(d) IBR, 24 rectangles.



(e) QTD, 386 rectangles.

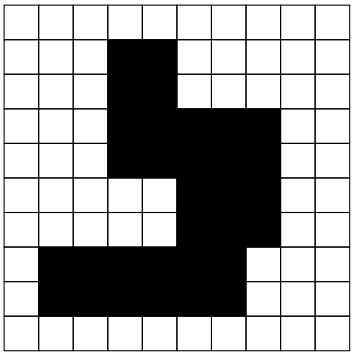


(f) MED, 157 rectangles.



(g) DTD, 25 rectangles.

FIGURE D.3 – Comparaison des décompositions obtenues par les différentes méthodes après traitement de la matrice binaire D.3a.



(a) Polygone rectilinéaire dans une image binaire.

$$\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 8 & 4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 6 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 10 & 8 & 12 & 8 & 4 & 0 & 0 \\
0 & 0 & 0 & 5 & 4 & 10 & 6 & 3 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 8 & 4 & 2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 6 & 3 & 1 & 0 & 0 \\
0 & 12 & 10 & 8 & 6 & 4 & 2 & 0 & 0 & 0 \\
0 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

(b) Matrice contenant pour chaque élément l'aire du plus grand rectangle.

FIGURE D.4 – Illustration du résultat obtenu après la recherche du plus grand rectangle sur chaque point d'une matrice.

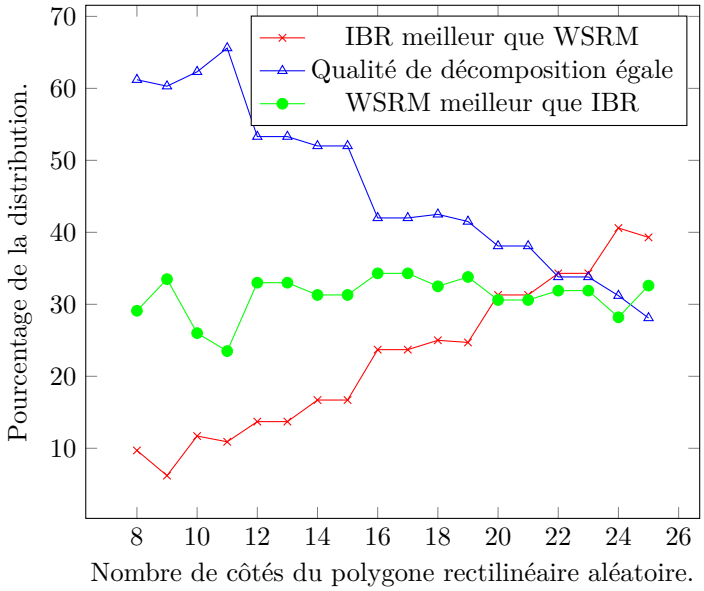


FIGURE D.5 – Comparaison qualitative de la décomposition entre les méthodes WSRM et IBR en fonction du nombre de côtés constituant le polygone rectilinéaire.

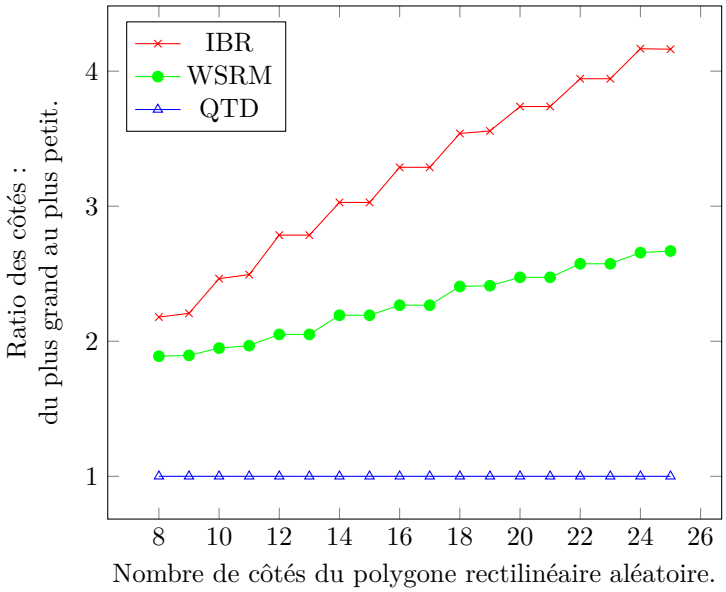
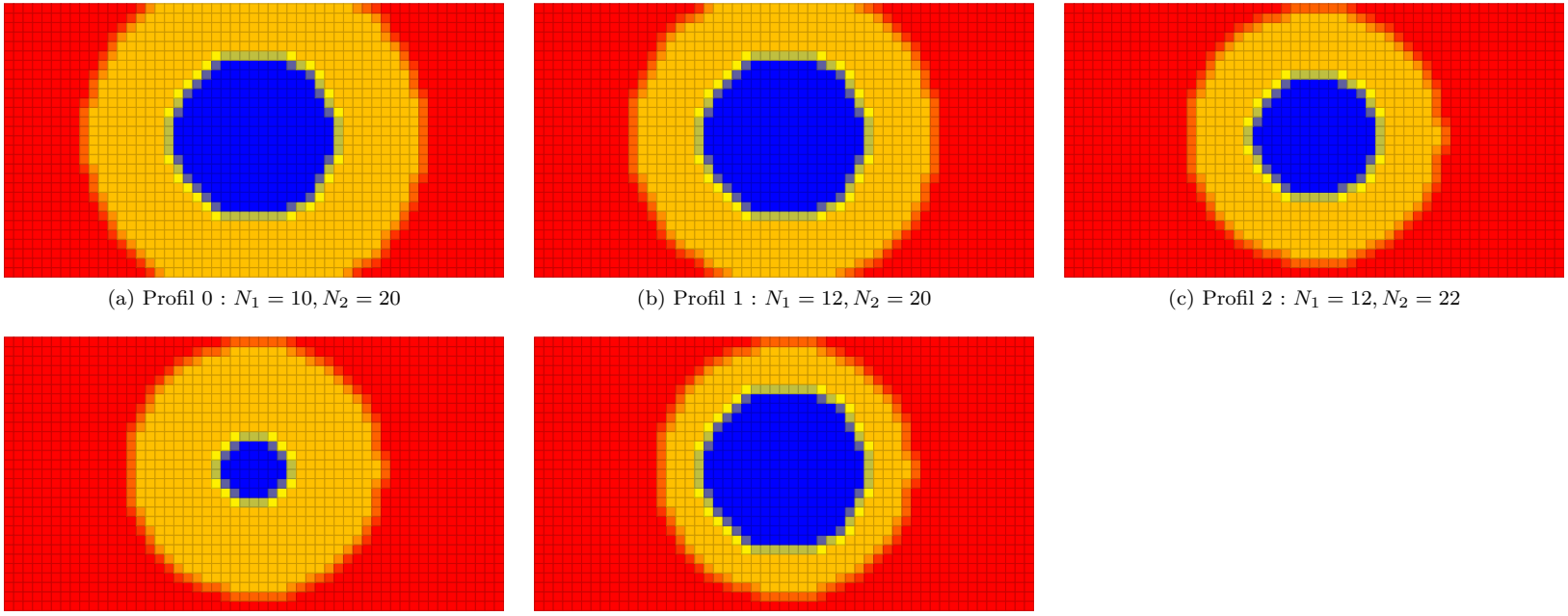


FIGURE D.6 – Ratio des côtés des rectangles décomposés, du plus grand au plus petit.

Annexe E

Profils utilisés pour la mise en œuvre des cartes d'attention

Les cinq profils modélisant la carte d'attention visuelle du jeu Doom3 utilisés lors des tests sont présentés ci-dessous.



(a) Profil 0 : $N_1 = 10, N_2 = 20$ (b) Profil 1 : $N_1 = 12, N_2 = 20$ (c) Profil 2 : $N_1 = 12, N_2 = 22$

(d) Profil 3 : $N_1 = 12, N_2 = 24$ (e) Profil 4 : $N_1 = 15, N_2 = 20$

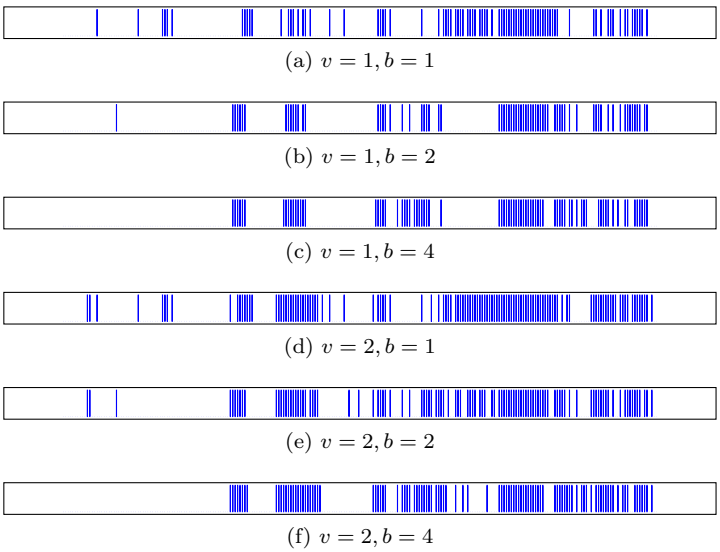
FIGURE E.1 – Profils utilisés pour la modélisation des cartes d'attention visuelle dans le cas du jeu Doom3 (Résolution spatiale de 480p).

Annexe F

Données issues de l'analyse de l'espace de représentation hybride

Les différents graphiques et tableaux présentés ci-dessous permettent l'analyse de l'espace de représentation hybride proposé.

F.1 Chronologie des images bimodales



(a) $v = 1, b = 1$

(b) $v = 1, b = 2$

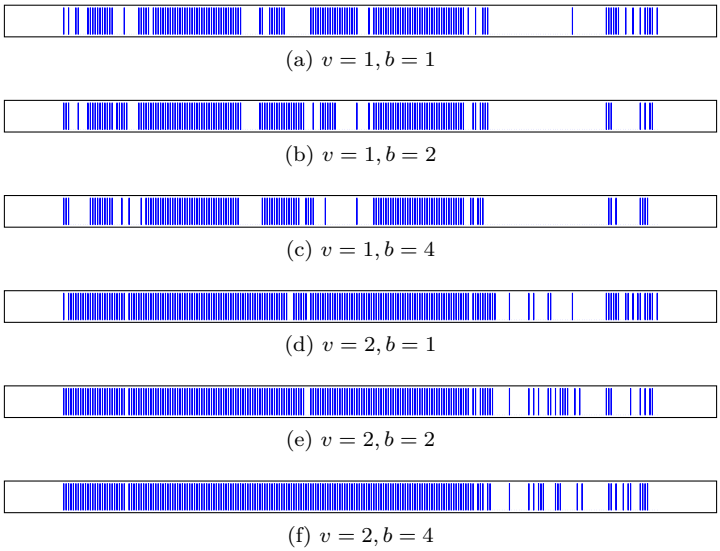
(c) $v = 1, b = 4$

(d) $v = 2, b = 1$

(e) $v = 2, b = 2$

(f) $v = 2, b = 4$

FIGURE F.1 – Chronologie des images compressées de manière bimodale pour tous les tests réalisés avec la séquence Doom3 – 1.



(a) $v = 1, b = 1$

(b) $v = 1, b = 2$

(c) $v = 1, b = 4$

(d) $v = 2, b = 1$

(e) $v = 2, b = 2$

(f) $v = 2, b = 4$

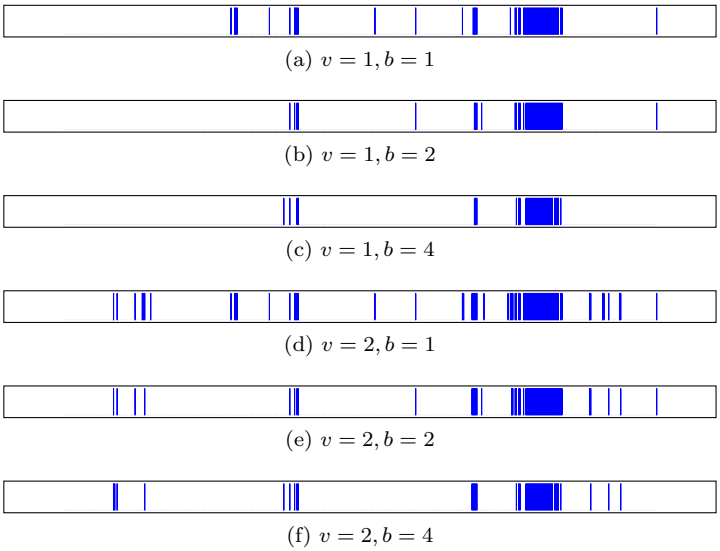
FIGURE F.2 – Chronologie des images compressées de manière bimodale pour tous les tests réalisés avec la séquence Doom3 – 4.

TABLE F.1 – Récapitulatif du nombre d'images traitées de manière bimodale pour les trois séquences de test (avec $v = 2$).

		Images bimodales		
Séquence	**Nombre d'images**	$b = 1$	$b = 2$	$b = 4$
Doom3 – 1	246	119 (48%)	107 (43%)	106 (43%)
Doom3 – 4	246	195 (79%)	194 (79%)	195 (79%)
Doom3 – 12	496	75 (15%)	58 (12%)	48 (10%)

F.2 Impact sur le débit binaire

F.3 Chronologie des images bimodales avec application des modèles de cartes d'attention



(a) $v = 1, b = 1$

(b) $v = 1, b = 2$

(c) $v = 1, b = 4$

(d) $v = 2, b = 1$

(e) $v = 2, b = 2$

(f) $v = 2, b = 4$

FIGURE F.3 – Chronologie des images compressées de manière bimodale pour tous les tests réalisés avec la séquence Doom3 – 12.

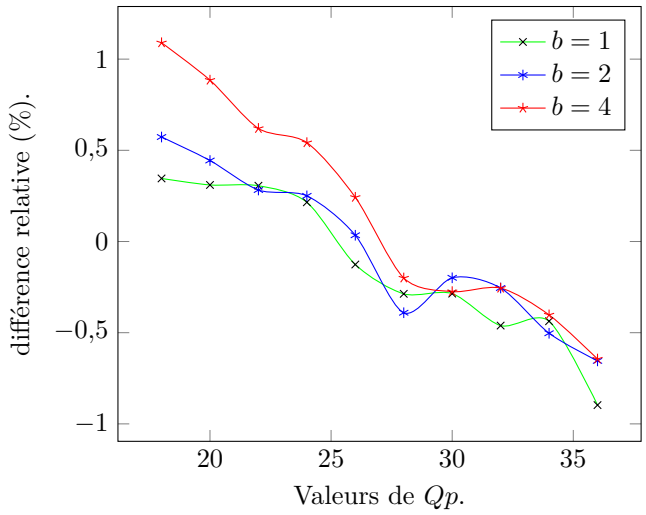
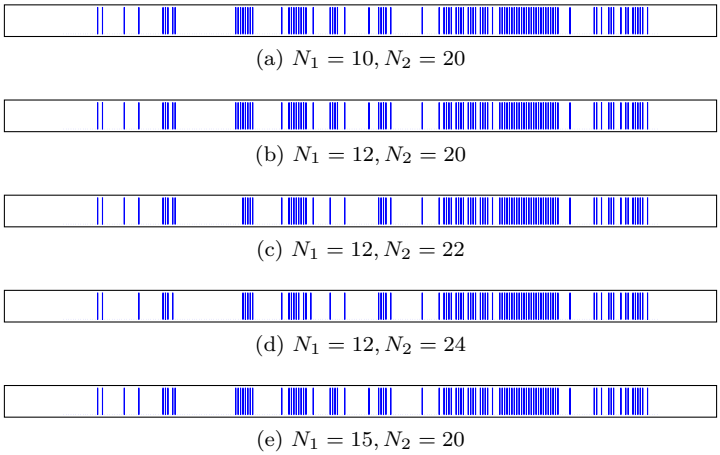


FIGURE F.4 – Différence relative du débit binaire en fonction de la valeur de Qp (séquence vidéo Doom3 – 1, avec $v = 2, L = 3$ et $p = ultrafast$).



(a) $N_1 = 10, N_2 = 20$

(b) $N_1 = 12, N_2 = 20$

(c) $N_1 = 12, N_2 = 22$

(d) $N_1 = 12, N_2 = 24$

(e) $N_1 = 15, N_2 = 20$

FIGURE F.5 – Chronologie des images compressées de manière bimodale après application des profils de cartes d'attention (séquence vidéo Doom3 − 1 avec $v = 1$ et $b = 1$).

Annexe G

Options de configuration

Cette annexe détaille toutes les options utilisables par l'utilisateur lors d'un transmodage, lors de la lecture d'un flux transmodé ou lors de l'évaluation de la qualité d'un flux vidéo multimodal face à son équivalent transcodé. Le type de variable, la valeur par défaut ainsi que les valeurs possibles sont donnés.

G.1 Décodage

Les options utilisables sont résumées dans le tableau G.1. La fonction de chaque option est détaillée ci-dessous.

filePath Chemin vers le fichier à décompresser.

useAsAGenericDecoder Cette variable permet d'informer le décodeur si le fichier à décompresser est une vidéo utilisant notre format (0), c'est à dire transmodée ou dans un format classique 1.

useRawDecoder Utilise la suite de décodeurs incluse dans FFmpeg (0) ou une implémentation d'un décodeur de flux vidéo non compressée (au format RGB si la variable est positionnée à 1.

rawWidth Largeur à utiliser lors du décodage d'un fichier non compressé. Pris en compte uniquement quand la variable *useRawDecoder* est égale à 1.

rawHeight Hauteur à utiliser lors du décodage d'un fichier non compressé.

rawFrameRate Fréquence à laquelle a été enregistré un fichier non compressé.

startAtFrame Démarre le décodage d'un fichier non compressé à partir d'un index particulier.

onlyDecodeNFrames Limite le nombre d'images à décoder dans un flux vidéo non compressé. Cette valeur doit être supérieure ou égale à celle contenue dans la variable *startAtFrame*.

TABLE G.1 – Options de configuration pour le décodage.

Nom de l'option dans les fichiers XML	**Nom de l'option dans la ligne de commande**	**Type de variable**	**Valeurs possibles**	**Valeurs par défaut**
filePath	I	chaîne de caractères	N/A	N/A
useAsAGenericDecoder	g	entier	0 ou 1	1
useRawDecoder	r	entier	0 ou 1	0
rawWidth	W	entier	De 0 à 9999	800
rawHeight	H	entier	De 0 à 9999	600
rawFrameRate	f	entier	De −1 à 99	15
startAtFrame	S	entier	De −1 à 999	−1
onlyDecodeNFrames	o	entier	De −1 à 999	−1

G.2 Séparation des modalités

Le tableau G.2 présente les options modifiables dans le processus de séparation des modalités. La fonction de chaque option est détaillée ci-dessous.

yDecimalPrecicion Nombre de chiffres significatifs à utiliser sur la composante Y. Influe uniquement sur l'encodage de la partie paramétrique de l'information.

crDecimalPrecicion Nombre de chiffres significatifs à utiliser sur la composante Cr. Influe uniquement sur l'encodage de la partie paramétrique de l'information.

cbDecimalPrecicion Nombre de chiffres significatifs à utiliser sur la composante Cb. Influe uniquement sur l'encodage de la partie paramétrique de l'information.

internalQuantizationStep Pas à utiliser pour l'échantillonnage à l'intérieur des zones définies pour être encodées de façon paramétrique.

edgeQuantizationStep Pas à utiliser pour l'échantillonnage des bords des zones définies pour être encodées de façon paramétrique.

cannyFilterRefinement Valeur à utiliser lors de l'application du filtre de Canny. Permet de moduler la finesse des détails détectés.

maxVectorAreas Fixe le nombre maximal d'étiquettes à conserver. La valeur -1 permet de ne pas mettre de limite.

minVectorAreaRatio Permet de supprimer les étiquettes dont la surface est inférieure à une certaine valeur. Cette valeur est calculée en faisant le ratio de la surface de l'étiquette par la surface totale de l'image. La valeur -1 permet de ne pas fixer de limite.

maxVectorLRAreas Fixe le nombre maximal de rectangles à conserver. La valeur -1 permet de ne pas mettre de limite.

minVectorLRAreaRatio Permet de supprimer les rectangles dont la surface est inférieure à une certaine valeur. Cette valeur est calculée en faisant le ratio de la surface du rectangle par la surface totale de l'image. La valeur -1 permet de ne pas fixer de limite.

minAreasCoeff Valeur minimale du coefficient de corrélation pour qu'une représentation paramétrique soit acceptée.

enablePolynomialRegroupement Si la variable est positionnée à 1, une tentative de regroupement des zones destinées à être encodées de façon paramétrique est réalisée.

polynomialRegroupementStep Pas d'échantillonnage utilisé pour tester le potentiel de regroupement des zones. Ce test est uniquement réalisé sur la composante Y des images.

minAreasRegroupementCoeff Valeur minimale du coefficient de corrélation pour autoriser un regroupement.

LRThreading Permet d'activer ou non le support multicœur dans le processus de décomposition d'une image binaire en une suite de plus grands rectangles.

blobThreading Permet d'activer ou non le support multicœur lors de la gestion des étiquettes.

interpoThreading Permet d'activer ou non le support multicœur lors des tâches d'interpolation (au niveau des zones).

regInterpoThreading Permet d'activer ou non le support multicœur lors des tâches d'interpolation (au niveau des canaux de l'image).

largestRectangleMethod Permet de choisir la méthode à utiliser lors du calcul de la suite de plus grands rectangles à partir d'une image binaire.

qpMap Lorsque cette variable est égale à 1, la quantification en fonction d'un modèle de carte d'attention est utilisée.

qpThresholds Seuils utilisés pour créer trois zones distinctes, possédant chacune une valeur de quantification donnée.

qpSteps Pas de quantification utilisé entre les trois zones.

TABLE G.2 – Options pour la séparation des modalités.

Nom de l'option dans les fichiers XML	**Nom de l'option dans la ligne de commande**	**Type de variable**	**Valeurs possibles**	**Valeurs par défaut**
yDecimalPrecicion	x	entier	de 0 à 4	2
crDecimalPrecicion	y	entier	de 0 à 4	2
cbDecimalPrecicion	z	entier	de 0 à 4	2
internalQuantizationStep	Q	entier	de 0 à 16	8
edgeQuantizationStep	q	entier	de 0 à 16	1
cannyFilterRefinement	c	entier	3, 5 ou 7	3
maxVectorAreas	v	entier	de -1 à 99	-1
minVectorAreaRatio	V	entier	de 0 à 100	0
maxVectorLRAreas	b	entier	de -1 à 99	-1
minVectorLRAreaRatio	A	entier	de 0 à 100	0
minAreasCoeff	u	réel	de 0 à 1	$0,50$
enablePolynomialRegroupement	j	entier	0 ou 1	1
polynomialRegroupementStep	w	entier	de 0 à 100	8
minAreasRegroupementCoeff	U	réel	de 0 à 1	$0,70$
LRThreading	X	entier	0 ou 1	0
blobThreading	Y	entier	0 ou 1	0
interpoThreading	Z	entier	0 ou 1	0
regInterpoThreading	J	entier	0 ou 1	0
largestRectangleMethod	K	entier	de 0 à 2	1
qpMap	d	entier	0 ou 1	0
qpThresholds	d	entier	$\leq$ à Qp	1
qpSteps	d	entier	de 0 à 4	1

G.3 Encodage multimodal

Le tableau G.3 présente les options disponibles lors de l'encodage d'un flux vidéo multimodal. La fonction de chaque option est détaillée ci-dessous.

filePath Chemin ou adresse IP pour le fichier ou le flux vidéo multimodal. Par défaut, la diffusion n'est active que lorsque l'option *enableStreaming* est positionnée à 1.

outputFormat Format du conteneur vidéo à utiliser.

enableStreaming Permet d'activer ou non la diffusion de la vidéo multimodale.

sdpFilePath Chemin vers lequel sera enregistré le fichier SDP [1]. Option active uniquement lorsque l'option *enableStreaming* est positionnée à 1.

mixteCoding Active ou non une compression multimodale. Si l'option est positionnée à 0, un transcodage est simplement réalisé. Au contraire si la variable est à 1 un transmodage est mis en œuvre.

optimisationLevel Active ou non les niveaux successifs d'optimisation apportés à la librairie x264 qui implémente la norme H.264. La valeur 0 utilise la librairie sans aucune modification, les valeurs 1 et 2 activent l'une ou l'autre des optimisations implémentées, la valeur 3 les utilise toutes.

addVectorData Lorsque la variable est égale à 1, les données de la seconde modalité sont ajoutées au flux vidéo, qui devient de fait multimodal. Cette option permet de mettre en avant l'impact (en taille) des données à ajouter au flux vidéo.

binaryEncodingMethod Méthode d'encodage des données de la seconde modalité à utiliser lors de l'empaquetage des données. Six méthodes sont disponibles.

compressionMethod Méthode de compression à utiliser pour compacter les données de la seconde modalité.

compressionLevel Niveau de compression à utiliser pour la compression des données de la seconde modalité.

fullFlush Impose à l'entité gérant la compression de faire une *full flush* lors de chaque image de type I. Cette opération rend le flux de données compressé indépendant des erreurs éventuelles survenues lors de la diffusion de la vidéo. En d'autres termes, si une erreur survient, le décodage des données de la seconde modalité sera à nouveau possible lors du décodage de la prochaine image I.

1. *Session Description Protocol*

threadNumber Cette option autorise l'utilisation d'une version multicœur du programme. Si la variable est positionnée à 0, une version non multicœur est utilisée. Lorsque la valeur est supérieure ou égale à 1, cette même valeur correspond au nombre de *thread* à utiliser.

preset Spécifie à la bibliothèque x264 le *preset* à utiliser. Il s'agit d'un ensemble d'options pré-configurées agissant lors de la compression vidéo.

tune Spécifie à la bibliothèque x264 le *tune* à utiliser. Cette option permet de raffiner les options positionnées par l'utilisation d'un *preset* spécifique, notamment pour un cas d'usage spécifique.

options Cette variable permet de modifier n'importe quelle option vidéo reconnue par la librairie x264.

TABLE G.3 – Options utilisables lors de l'encodage multimodal.

Nom de l'option dans les fichiers XML	**Nom de l'option dans la ligne de commande**	**Type de variable**	**Valeurs possibles**	**Valeurs par défaut**
filePath	*O*	chaîne de caractères	N/A	N/A
outputFormat	*E*	chaîne de caractères	h264, mp4, MPEG-TS RTP, RTSP	h264
enableStreaming	*s*	entier	0 ou 1	0
sdpFilePath	N/A	chaîne de caractères	N/A	./
mixteCoding	*M*	entier	0 ou 1	1
optimisationLevel	*L*	entier	de 0 à 3	0
addVectorData	*a*	entier	0 ou 1	1
binaryEncodingMethod	*D*	entier	de 0 à 5	0
compressionMethod	*C*	entier	de 0 à 2	1
compressionLevel	*l*	entier	de 0 à 9	9
fullFlush	*F*	entier	0 ou 1	0
threadNumber	*T*	entier	de 0 à 8	0
preset	*p*	chaîne de caractères	voir x264	*veryfast*
tune	*t*	chaîne de caractères	voir x264	*zerolatency*
options	*B*	chaîne de caractères	voir x264	$Qp = 20$

G.4 Évaluation de la qualité

Le tableau G.4 présente les options disponibles lors de l'évaluation de la qualité d'un flux vidéo multimodal. Celui-ci est comparé à la vidéo originale et à son équivalent transcodé. La fonction de chaque option est détaillée ci-dessous.

Les options *filePath* à *onlyDecodeNFrames* s'appliquent au fichier vidéo original et ont déjà été décrites dans le tableau G.1. Les options surlignées en bleu sont relatives à la version transcodée du fichier original. Enfin les options surlignées en orange sont utilisées pour le décodage de la version transmodée du flux vidéo.

TABLE G.4 – Options utilisables lors de l'évaluation de la qualité.

Nom de l'option dans les fichiers XML	**Nom de l'option dans la ligne de commande**	**Type de la variable**	**Valeurs possibles**	**Valeurs par défaut**
filePath	*I*	chaîne de caractères	N/A	N/A
useAsAGenericDecoder	*g*	entier	0 ou 1	1
useRawDecoder	*r*	entier	0 ou 1	0
rawWidth	*W*	entier	De 0 à 9999	800
rawHeight	*H*	entier	De 0 à 9999	600
rawFrameRate	*f*	entier	De −1 à 99	15
startAtFrame	*S*	entier	De −1 à 999	−1
onlyDecodeNFrames	*o*	entier	De −1 à 999	−1
filePath	*a*	chaîne de caractères	N/A	N/A
useAsAGenericDecoder	*b*	entier	0 ou 1	1
onlyDecodeNFrames	*c*	entier	De −1 à 999	−1
filePath	*A*	chaîne de caractères	N/A	N/A
useAsAGenericDecoder	*B*	entier	0 ou 1	1
onlyDecodeNFrames	*C*	entier	De −1 à 999	−1

Annexe H

Publications

Les publications relatives à cette thèse sont énumérées ci-dessous :

Pierre-Olivier Rocher et al. « Video Stream Transmodality ». In : *ICEIS 2014-Proceedings of the 16th International Conference on Enterprise Information Systems*. 2014, p. 28–37

Pierre-Olivier Rocher et al. « Video Stream Transmodality ». In : *Springer LNBIP Series Book*. Accepted, 2014

Pierre-Olivier Rocher. « Transmodalité des flux vidéos dans les réseaux du futur ». In : *Journée d'étude : le contexte dans tout ses états*. INSA de Lyon, 14 mars 2013

Julien Subercaze et al. « WSRM - A linear-time algorithm for the rectangle decomposition of binary images ». In : Submitted, 2014

Bibliographie

Articles

[1] Darren Abramson et al. « Intel Virtualization Technology for Directed I/O. » In : *Intel technology journal* 10.3 (2006) (cf. p. 79).

[2] Nasir Ahmed, T Natarajan et Kamisetty R Rao. « Discrete cosine transform ». In : *Computers, IEEE Transactions on* 100.1 (1974), p. 90–93 (cf. p. 18).

[3] Ghassan Al-Regib, Yucel Altunbasak et Russell M Mersereau. « Bit allocation for joint source and channel coding of progressively compressed 3-D models ». In : *Circuits and Systems for Video Technology, IEEE Transactions on* 15.2 (2005), p. 256–268 (cf. p. 67).

[4] Chandrajit Bajaj et al. « Error resilient streaming of compressed vrml ». In : *Rapport technique, TICAM, Univ. of Texas* (1998) (cf. p. 66).

[5] Ricardo A Baratto, Leonard N Kim et Jason Nieh. « THINC : a virtual display architecture for thin-client computing ». In : *ACM SIGOPS Operating Systems Review* 39.5 (2005), p. 277–290 (cf. p. 64).

[6] Michel Barlaud et al. « Pyramidal lattice vector quantization for multiscale image coding ». In : *Image Processing, IEEE Transactions on* 3.4 (1994), p. 367–381 (cf. p. 39).

[7] Pierre Bézier. « Définition numérique des courbes et surfaces I ». In : *Automatisme* 11.12 (1966), p. 625–632 (cf. p. 26).

[8] John Canny. « A computational approach to edge detection ». In : *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 6 (1986), p. 679–698 (cf. p. 35, 112).

[9] Jean-Fracois Colonna. « Animation of fractal objects ». In : *Computer Graphics Interface* 89 (1989) (cf. p. 41, 42).

[10] Cyril Concolato, J-C Dufourd et J-C Moissinac. « Creating and encoding of cartoons using MPEG-4 BIFS : methods and results ». In : *Circuits and Systems for Video Technology, IEEE Transactions on* 13.11 (2003), p. 1129–1135 (cf. p. 55).

[11] Jian-Rong Dai et Yi-Min Hu. « Intensity-modulation radiotherapy using independent collimators : an algorithm study ». In : *Medical physics* 26.12 (1999), p. 2562–2570 (cf. p. 177).

[12] Konrad Engel. « Optimal matrix-segmentation by rectangles ». In : *Discrete Applied Mathematics* 157.9 (2009), p. 2015–2030 (cf. p. 177).

[13] Pedro F Felzenszwalb et Daniel P Huttenlocher. « Efficient graph-based image segmentation ». In : *International Journal of Computer Vision* 59.2 (2004), p. 167–181 (cf. p. 58).

[14] L Ferrari, PV Sankar et Jack Sklansky. « Minimal rectangular partitions of digitized blobs ». In : *Computer vision, graphics, and image processing* 28.1 (1984), p. 58–71 (cf. p. 115, 177).

[15] KS Fu. « Syntactic image modeling using stochastic tree grammars ». In : *Computer Graphics and Image Processing* 12.2 (1980), p. 136–152 (cf. p. 42).

[16] H Gharavi et H Reza-Alikhani. « Pel-recursive motion estimation algorithm ». In : *Electronics Letters* 37.21 (2001), p. 1285–1286 (cf. p. 57).

[17] Alexander Grossmann et Jean Morlet. « Decomposition of Hardy functions into square integrable wavelets of constant shape ». In : *SIAM journal on mathematical analysis* 15.4 (1984), p. 723–736 (cf. p. 18).

[18] Arnaud E Jacquin. « Fractal image coding : a review ». In : *Proceedings of the IEEE* 81.10 (1993), p. 1451–1465 (cf. p. 41).

[19] Yeong-An Jeong et Cha-Keon Cheong. « A DCT-based embedded image coder using wavelet structure of DCT for very low bit rate video codec ». In : *Consumer Electronics, IEEE Transactions on* 44.3 (1998), p. 500–508 (cf. p. 56).

[20] J Mark Keil. « Polygon decomposition ». In : *Handbook of Computational Geometry* 2 (2000), p. 491–518 (cf. p. 175).

[21] Haibo Li, Astrid Lundmark et Robert Forchheimer. « Image sequence coding at very low bit rates : A review ». In : *Image Processing, IEEE Transactions on* 3.5 (1994), p. 589–609 (cf. p. 40).

[22] Jianhua Lin. « Divergence measures based on the Shannon entropy ». In : *Information Theory, IEEE Transactions on* 37.1 (1991), p. 145–151 (cf. p. 93).

[23] Yoseph Linde, Andres Buzo et Robert M Gray. « An algorithm for vector quantizer design ». In : *Communications, IEEE Transactions on* 28.1 (1980), p. 84–95 (cf. p. 57).

[24] W Lipski et al. « On two dimensional data organization II ». In : *Fundamenta Informaticae* 2.3 (1979), p. 245–260 (cf. p. 177).

[25] Shin-Yee Lu et King Sun Fu. « A syntactic approach to texture analysis ». In : *Computer Graphics and Image Processing* 7.3 (1978), p. 303–330 (cf. p. 42, 43).

[26] Blake C Lucas et al. « Parametric Images : An Image Representation that Preserves Edge Strength in Registration and Atlasing ». In : () (cf. p. 42).

[27] Raman Maini et Himanshu Aggarwal. « Study and comparison of various image edge detection techniques ». In : *International Journal of Image Processing (IJIP)* 3.1 (2009), p. 1–11 (cf. p. 112).

[28] Arun N Netravali et JD Robbins. « Motion-Compensated Television Coding : Part I ». In : *Bell System Technical Journal* 58.3 (1979), p. 631–670 (cf. p. 57).

[29] Alexandrina Orzan et al. « Diffusion curves : a vector representation for smooth-shaded images ». In : *Communications of the ACM* 56.7 (2013), p. 101–108 (cf. p. 30, 35, 36).

[30] JW Patterson, CD Taylor et Phil J Willis. « Constructing and rendering vectorised photographic images ». In : *The Journal of Virtual Reality and Broadcasting* 9.3 (2012) (cf. p. 55).

[31] Manoranjan Paul, Manzur Murshed et Laurence S Dooley. « Very Low Bit-rate Video Coding ». In : *Video Data Management and Information Retrieval* (2005), p. 100 (cf. p. 56).

[32] Marius Preda et Francoise Preteux. « Virtual character within MPEG-4 animation framework eXtension ». In : *Circuits and Systems for Video Technology, IEEE Transactions on* 14.7 (2004), p. 975–988 (cf. p. 40).

[33] Keith Rayner. « Eye movements in reading and information processing : 20 years of research. » In : *Psychological bulletin* 124.3 (1998), p. 372 (cf. p. 83).

[34] Keith Rayner et al. « Eye movements and visual encoding during scene perception ». In : *Psychological science* 20.1 (2009), p. 6–10 (cf. p. 83).

[35] Rusty Russell. « virtio : towards a de-facto standard for virtual I/O devices ». In : *ACM SIGOPS Operating Systems Review* 42.5 (2008), p. 95–103 (cf. p. 79).

[36] Robert W Scheifler et Jim Gettys. « The X window system ». In : *ACM Transactions on Graphics (TOG)* 5.2 (1986), p. 79–109 (cf. p. 64).

[37] Claude Elwood Shannon. « A mathematical theory of communication ». In : *ACM SIGMOBILE Mobile Computing and Communications Review* 5.1 (2001), p. 3–55 (cf. p. 93).

[38] Julien Signes, Yuval Fisher et Alexandros Eleftheriadis. « MPEG-4's binary format for scene description ». In : *Signal Processing : Image Communication* 15.4 (2000), p. 321–345 (cf. p. 32).

[39] Iraklis M Spiliotis et Basil G Mertzios. « Real-time computation of two-dimensional moments on binary images using image block representation ». In : *Image Processing, IEEE Transactions on* 7.11 (1998), p. 1609–1615 (cf. p. 116, 179).

[40] Stephen M. Stigler. « Francis Galton's account of the invention of correlation. » In : *Statistical Science* 4.2 (1989), p. 73–79. issn : 0883-4237. doi : 10.1214/ss/1177012580 (cf. p. 93).

[41] Subarna Tripathi et al. « Region-of Interest based Parametric Video Compression Scheme ». In : *Communicated to the IEEE Transactions on Circuits and Systems for Video Technology* (2012) (cf. p. 58).

[42] Bert Vankeirsbilck et al. « Platform for real-time subjective assessment of interactive multimedia applications ». In : *Multimedia Tools and Applications* (2013), p. 1–27 (cf. p. 82).

[43] S Webb et al. « Intensity-modulated radiation therapy using a variable-aperture collimator ». In : *Physics in medicine and biology* 48.9 (2003), p. 1223 (cf. p. 177).

[44] Steve Webb. « Fewer segments for IMRT generated by modulation splitting ». In : *Physics in medicine and biology* 47.17 (2002), N217 (cf. p. 177).

[45] Mathias Wien et al. « Real-time system for adaptive video streaming based on SVC ». In : *Circuits and Systems for Video Technology, IEEE Transactions on* 17.9 (2007), p. 1227–1237 (cf. p. 81).

[46] Brendt Wohlberg et Gerhard De Jager. « A review of the fractal image coding literature ». In : *Image Processing, IEEE Transactions on* 8.12 (1999), p. 1716–1729 (cf. p. 42).

[47] Zhidong Yan, Sunil Kumar et C-CJ Kuo. « Mesh segmentation schemes for error resilient coding of 3-D graphic models ». In : *Circuits and Systems for Video Technology, IEEE Transactions on* 15.1 (2005), p. 138–144 (cf. p. 66).

[48] MF Zakaria et al. « Fast algorithm for the computation of moment invariants ». In : *Pattern Recognition* 20.6 (1987), p. 639–643 (cf. p. 179).

[49] Song-Hai Zhang et al. « Vectorizing cartoon animations ». In : *Visualization and Computer Graphics, IEEE Transactions on* 15.4 (2009), p. 618–629 (cf. p. 30, 34, 112, 119, 120).

Livres

[50] Royal Society (Great Britain). *Proceedings of the Royal Society of London.* vol. 58. Taylor & Francis, 1895. url : `http://books.google.fr/books?id=60aL0zlT-90C` (cf. p. 93).

[51] James Richard Diebel et Sebastian Adviser-Thrun. *Bayesian Image Vectorization : the probabilistic inversion of vector image rasterization.* Stanford University, 2008 (cf. p. 30, 31).

[52] C. Levcopoulos. *Minimum Length and "thickest-first" Rectangular Partitions of Polygons.* LiTH-IDA-R. 1986. url : `http://books.google.fr/books?id=s7OnygAACAAJ` (cf. p. 175).

[53] Yun Q Shi et Huifang Sun. *Image and video compression for multimedia engineering : fundamentals, algorithms, and standards.* CRC press, 1999 (cf. p. 35, 52).

Brevets

[54] Miriam Briskin, Yoram Elihai et Yosef Yomdin. « Apparatus and method for picture representation by data compression ». Brev. US Patent 5,510,838. 1996 (cf. p. 32).

[55] S. Chaudhury et al. « System and method for object based parametric video coding ». Brev. US Patent App. 12/554,579. 2011. url : `http://www.google.com/patents/US20110058609` (cf. p. 57).

[56] Yoram Elichai et Yosef Yomdin. « Method and apparatus for image analysis and processing by identification of characteristic lines and corresponding parameters ». Brev. US Patent 6,760,483. 2004 (cf. p. 32).

[57] E. Gunnar, E. John et S. Maarten. « Method and installation for detecting and following an eye and the gaze direction thereof ». Brev. WO Patent App. PCT/SE2003/001,813. 2004. URL : http://www.google.com/patents/WO2004045399A1?cl=en (cf. p. 159).

[58] L. Liang et al. « Creating optimized gradient mesh of a vector-based image from a raster-based image ». Brev. WO Patent App. PCT/US2008/062,970. 2008. URL : http://www.google.com/patents/WO2008137967A1?cl=en (cf. p. 33).

[59] J. Patterson et P. Wills. « Image processing and vectorisation ». Brev. WO Patent App. PCT/GB2007/002,470. 2008. URL : http://www.google.com/patents/WO2008003944A2?cl=en (cf. p. 55).

[60] Yosef Yomdin et Yoram Elichai. « Encoding of geometric modeled images ». Brev. US Patent App. 10/496,536. 2002 (cf. p. 32).

[61] Yosef Yomdin et Yoram Elichai. « Geometric and brightness modeling of images ». Brev. US Patent 20,040,174,361. 2004 (cf. p. 32).

[62] Yosef Yomdin et Yoram Elichai. « Method and apparatus for image representation by geometric and brightness modeling ». Brev. US Patent 6,801,210. 2004 (cf. p. 32).

Autres sources

[63] Ali Aghagolzadeh et al. « A Novel Video Compression Technique for Very Low Bit-Rate Coding by Combining H. 264/AVC Standard and 2-D Wavelet Transform ». In : *Signal Processing, 2008. ICSP 2008. 9th International Conference on*. IEEE. 2008, p. 1251–1254 (cf. p. 56).

[64] Hamed Ahmadi et al. « Efficient bitrate reduction using a Game Attention Model in cloud gaming ». In : *Haptic Audio Visual Environments and Games (HAVE), 2013 IEEE International Symposium on*. IEEE. 2013, p. 103–108 (cf. p. 83, 84).

[65] Matt Aranha et al. « A physically-based client-server rendering solution for mobile devices ». In : *Proceedings of the 6th international conference on Mobile and ubiquitous multimedia*. ACM. 2007, p. 149–154 (cf. p. 63, 66).

[66] Curtis A. Armstrong. « Abstract Vectorization of Raster Images Using B-Spline Surfaces ». Master Thesis. 2006 (cf. p. 30, 31).

[67] Ivica Arsov. « A framework for distributed 3D graphics applications based on compression and streaming ». Thèse de Doctorat. Institut National des Télécommunications, 2011 (cf. p. 62).

[68] MF Barnsley et SG Demko. « Iterated function schemes and the global construction of fractals[J] ». In : *Proceedings of the Royal Society of London*. T. A399. 1985, p. 243–275 (cf. p. 41).

[69] MF Barnsley, JH Elton et DP Hardin. « Recurrent iterated function systems[J] ». In : *Constructive Approximation*. T. 5(1). 1989, p. 3–31 (cf. p. 41).

[70] M Oguz Bici, Andrey Norkin et Gozde Bozdagi Akar. « Packet loss resilient transmission of 3D models ». In : *Image Processing, 2007. ICIP 2007. IEEE International Conference on*. T. 5. IEEE. 2007, p. V–121 (cf. p. 66).

[71] Azzedine Boukerche, Raed Jarrar et Richard Werner Pazzi. « An efficient protocol for remote virtual environment exploration on wireless mobile devices ». In : *Proceedings of the 4th ACM workshop on Wireless multimedia networking and performance modeling*. ACM. 2008, p. 45–52 (cf. p. 63).

[72] Ian Buck, Greg Humphreys et Pat Hanrahan. « Tracking graphics state for networked rendering ». In : *Proceedings of the ACM SIGGRAPH /EUROGRAPHICS workshop on Graphics hardware*. ACM. 2000, p. 87–95 (cf. p. 63).

[73] Kuan-Ta Chen et al. « Measuring the latency of cloud gaming systems ». In : *Proceedings of the 19th ACM international conference on Multimedia*. ACM. 2011, p. 1269–1272 (cf. p. 83).

[74] Liang Cheng et al. « Realtime 3d graphics streaming using mpeg-4 ». In : *Proceedings of the IEEE/ACM Workshop on Broadband Wireless Services and Applications (BroadWise'04)*. 2004, p. 1–16 (cf. p. 64).

[75] *Cisco Visual Networking Index : Forecast and Methodology, 2012-2017; [Visual Networking Index (VNI)]*. 2013 (cf. p. 3, 46).

[76] *Coding of audio-visual objects – Part 10 : Advanced Video Coding*. Rapp. tech. ISO/IEC 14496-10. 2010 (cf. p. 54).

[77] *Coding of audio-visual objects – Part 11 : Scene description and application engine*. Rapp. tech. ISO/IEC 14496-11. 2005 (cf. p. 32).

[78] *Coding of audio-visual objects – Part 2 : Visual*. Rapp. tech. ISO/IEC 14496-2. 2009 (cf. p. 53).

[79] *Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 1 : Systems*. Rapp. tech. ISO/IEC 11172-1 : 1993 (cf. p. 52).

[80] Volker Coors. « Resource-adaptive interactive 3d maps ». In : *Proceedings of the 2nd international symposium on Smart graphics*. ACM. 2002, p. 140–144 (cf. p. 67, 68).

[81] Davy De Winter et al. « A hybrid thin-client protocol for multimedia streaming and interactive gaming applications ». In : *Proceedings of the 2006 international workshop on Network and operating systems support for digital audio and video*. ACM. 2006, p. 15 (cf. p. 67).

[82] Advanced Micro Devices. *AMD I/O Virtualization Technology (IOMMU) Specification*. 2006. URL : `http://support.amd.com/TechDocs/34434-IOMMU-Rev_1.26_2-11-09.pdf` (cf. p. 79).

[83] Joachim Diepstraten, Martin Gorke et Thomas Ertl. « Remote line rendering for mobile devices ». In : *Computer Graphics International, 2004. Proceedings*. IEEE. 2004, p. 454–461 (cf. p. 65, 68).

[84] *Digital compression and coding of continuous-tone still images - Requirements and guidelines*. Rapp. tech. ISO/IEC 10918-1. 1994 (cf. p. 22).

[85] *Dynamic adaptive streaming over HTTP (DASH) – Part 1 : Media presentation description and segment formats*. Rapp. tech. ISO/IEC 23009-1. Avr. 2012 (cf. p. 81).

[86] Claudia Ehmke et Stephanie Wilson. « Identifying web usability problems from eye-tracking data ». In : *Proceedings of the 21st British HCI Group Annual Conference on People and Computers : HCI... but not as we know it-Volume 1*. British Computer Society. 2007, p. 119–128 (cf. p. 82).

[87] Klaus Engel, Ove Sommer et Thomas Ertl. « A framework for interactive hardware accelerated remote 3d-visualization ». In : *Data Visualization 2000*. Springer, 2000, p. 167–177 (cf. p. 63).

[88] David Eppstein. « Graph-theoretic solutions to computational geometry problems ». In : *Graph-Theoretic Concepts in Computer Science*. Springer. 2010, p. 1–16 (cf. p. 177).

[89] M Ezhilarasan et P Thambidurai. « A Hybrid Transform Coding for Video Codec ». In : *Information Technology, 2006. ICIT'06. 9th International Conference on*. IEEE. 2006, p. 117–120 (cf. p. 56).

[90] Dashan Gao et Yizhou Wang. « Decomposing document images by heuristic search ». In : *Energy Minimization Methods in Computer Vision and Pattern Recognition*. Springer. 2007, p. 97–111 (cf. p. 177).

[91] *Generic coding of moving pictures and associated audio information – Part 1 : Systems*. Rapp. tech. ISO/IEC 13818-1. 2013 (cf. p. 53).

[92] Spyridon V Gogouvitis et al. « A Service Oriented Architecture for achieving QoSaware Workflow Management in Virtualized Environments ». In : *Network and Service Management (CNSM), 2010 International Conference on*. IEEE. 2010, p. 398–401 (cf. p. 79).

[93] *H.264 : Advanced video coding for generic audiovisual services*. Rapp. tech. ITU-T Recommendation H.264. 2014 (cf. p. 54).

[94] Mike Heath et al. « Comparison of edge detectors : a methodology and initial study ». In : *Computer Vision and Pattern Recognition, 1996. Proceedings CVPR'96, 1996 IEEE Computer Society Conference on*. IEEE. 1996, p. 143–148 (cf. p. 112).

[95] Berthold K Horn et Brian G Schunck. « Determining optical flow ». In : *1981 Technical Symposium East*. International Society for Optics et Photonics. 1981, p. 319–331 (cf. p. 57).

[96] Chun-Ying Huang et al. « GamingAnywhere : An open cloud gaming system ». In : *Proceedings of the 4th ACM Multimedia Systems Conference*. ACM. 2013, p. 36–47 (cf. p. 62, 68, 74, 85).

[97] Wei Huang et al. « A case for high performance computing with virtual machines ». In : *Proceedings of the 20th annual international conference on Supercomputing*. ACM. 2006, p. 125–134 (cf. p. 79).

[98] Greg Humphreys et al. « Chromium : a stream-processing framework for interactive rendering on clusters ». In : *ACM Transactions on Graphics (TOG)*. T. 21. 3. ACM. 2002, p. 693–702 (cf. p. 64).

[99] Greg Humphreys et al. « WireGL : a scalable graphics system for clusters ». In : *SIGGRAPH*. T. 1. 2001, p. 129–140 (cf. p. 62).

[100] *Inside Gaikai : how to make cloud gaming as easy as watching YouTube.* 2013. URL : http://www.theverge.com/2013/7/16/4442372/inside-gaikai-how-to-make-cloud-gaming-as-easy-as-watching-youtube (visité le 27/01/2014) (cf. p. 73).

[101] Michael JARSCHEL et al. « An evaluation of QoE in cloud gaming based on subjective tests ». In : *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011 Fifth International Conference on.* IEEE. 2011, p. 330–335 (cf. p. 83).

[102] Tom JEHAES, Peter QUAX et Wim LAMOTTE. « Adapting a large scale networked virtual environment for display on a PDA ». In : *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology.* ACM. 2005, p. 217–220 (cf. p. 68).

[103] *JPEG 2000 image coding system : Core coding system.* Rapp. tech. ISO /IEC 15444-1. 2004 (cf. p. 22).

[104] Tilke JUDD et al. « Learning to predict where humans look ». In : *Computer Vision, 2009 IEEE 12th international conference on.* IEEE. 2009, p. 2106–2113 (cf. p. 84).

[105] Aditya KHANDELIA et al. « Parametric video compression scheme using AR based texture synthesis ». In : *Computer Vision, Graphics & Image Processing, 2008. ICVGIP'08. Sixth Indian Conference on.* IEEE. 2008, p. 219–225 (cf. p. 57).

[106] David KOLLER et al. « Protected interactive 3D graphics via remote rendering ». In : *ACM Transactions on Graphics (TOG).* T. 23. 3. ACM. 2004, p. 695–703 (cf. p. 68).

[107] Martin KURZE et Roman ENGLERT. « Network centric photorealistic mixed reality on mobile devices ». In : *Proceedings of the 3rd international conference on Mobile technology, applications & systems.* ACM. 2006, p. 26 (cf. p. 63).

[108] Fabrizio LAMBERTI et al. « An accelerated remote graphics architecture for PDAS ». In : *Proceedings of the eighth international conference on 3D Web technology.* ACM. 2003, 55–ff (cf. p. 64).

[109] Antoine LAVIGNOTTE. « Prise en compte de la qualité de l'expérience utilisateur au sein des protocoles de streaming HTTP adaptatifs ». Thèse de Doctorat. Université Jean Monnet, 2014 (cf. p. 82).

[110] Antoine LAVIGNOTTE et al. « Quality of Experience, a very personal experience ! » In : *Database and Expert Systems Applications (DEXA), 2013 24th International Workshop on.* IEEE. 2013, p. 231–235 (cf. p. 85).

[111] Jean LE FEUVRE, Cyril CONCOLATO et Jean-Claude MOISSINAC. « GPAC : open source multimedia framework ». In : *Proceedings of the 15th international conference on Multimedia.* ACM. 2007, p. 1009–1012 (cf. p. 74, 77).

[112] Gregory LECOT et Bruno LEVY. « ARDECO : Automatic region detection and conversion ». In : *Proceedings of the 17th Eurographics conference on Rendering Techniques.* Eurographics Association. 2006, p. 349–360 (cf. p. 30, 35, 36).

[113] Christos Levcopoulos. « Fast heuristics for minimum length rectangular partitions of polygons ». In : *Proceedings of the second annual symposium on Computational geometry*. ACM. 1986, p. 100–108 (cf. p. 175).

[114] Christos Levcopoulos. « Improved bounds for covering general polygons with rectangles ». In : *Foundations of Software Technology and Theoretical Computer Science*. Springer. 1987, p. 95–102 (cf. p. 177).

[115] Andrzej Lingas et al. « Minimum edge length partitioning of rectilinear polygons ». In : *Proc. 20th Allerton Conf. Commun. Control Comput.* 1982, p. 53–63 (cf. p. 175).

[116] DT MacDonald et J Lang. « Bitmap to vector conversion for multilevel analysis and visualization ». In : *Proceedings of the International Conference on Scalable Vector Graphics*. 2008 (cf. p. 36, 37).

[117] *Methodology for the subjective assessment of the quality of television pictures*. Rapp. tech. ITU-R BT.500-13. 2012 (cf. p. 47, 84).

[118] Vlad Nae, Radu Prodan et Thomas Fahringer. « Cost-efficient hosting and load balancing of massively multiplayer online games ». In : *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*. IEEE. 2010, p. 9–16 (cf. p. 79).

[119] Ripal Nathuji, Aman Kansal et Alireza Ghaffarkhah. « Q-clouds : managing performance interference effects for QoS-aware clouds ». In : *Proceedings of the 5th European conference on Computer systems*. ACM. 2010, p. 237–250 (cf. p. 78, 79).

[120] Itay Nave et al. « Games Large graphics streaming architecture ». In : *Consumer Electronics, 2008. ISCE 2008. IEEE International Symposium on*. IEEE. 2008, p. 1–4 (cf. p. 63, 64, 67).

[121] Antti Nurminen. « m-LOMA-a mobile 3D city map ». In : *Proceedings of the eleventh international conference on 3D web technology*. ACM. 2006, p. 7–18 (cf. p. 67, 68).

[122] Antti Nurminen. « Mobile, hardware-accelerated urban 3D maps in 3G networks ». In : *Proceedings of the twelfth international conference on 3D web technology*. ACM. 2007, p. 7–16 (cf. p. 67, 68).

[123] Tatsuo Ohtsuki. « Minimum dissection of rectilinear regions ». In : *Proc. 1982 IEEE Symp. on Circuits and Systems, Rome*. 1982, p. 1210–1213 (cf. p. 177).

[124] *Parameter values for the HDTV standards for production and international programme exchange*. Rapp. tech. UIT-R BT.709-5. 2002 (cf. p. 17).

[125] J Parkkinen, T Jaaskelainen et M Kuittinen. « Spectral representation of color images ». In : *Pattern Recognition, 1988., 9th International Conference on*. IEEE. 1988, p. 933–935 (cf. p. 42).

[126] JW Patterson, CD Taylor et Philip J Willis. « Reconstructing vectorised photographic images ». In : *Visual Media Production, 2009. CVMP '09. Conference for*. IEEE. 2009, p. 15–24 (cf. p. 55).

[127] Ignacio Perea et Damián López. « Syntactic modeling and recognition of document images ». In : *Structural, Syntactic, and Statistical Pattern Recognition*. Springer, 2004, p. 416–424 (cf. p. 42).

[128] Jean-Charles Quillet et al. « Using expressive rendering for remote visualization of large city models ». In : *Proceedings of the eleventh international conference on 3D web technology*. ACM. 2006, p. 27–35 (cf. p. 65).

[129] Hesam Rahimi, AA Nazari Shirehjini et Shervin Shirmohammadi. « Activity-centric streaming of virtual environments and games to mobile devices ». In : *Haptic Audio Visual Environments and Games (HAVE), 2011 IEEE International Workshop on*. IEEE. 2011, p. 45–50 (cf. p. 84).

[130] Uma S Ranjan et KR Ramakrishnan. « A stochastic scale space for multiscale image representation ». In : *Scale-Space Theories in Computer Vision*. Springer, 1999, p. 441–446 (cf. p. 42).

[131] Nathan Regola et J-C Ducom. « Recommendations for virtualization technologies in high performance computing ». In : *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*. IEEE. 2010, p. 409–416 (cf. p. 79).

[132] Pierre-Olivier Rocher. « Transmodalité des flux vidéos dans les réseaux du futur ». In : *Journée d'étude : le contexte dans tout ses états*. INSA de Lyon, 14 mars 2013 (cf. p. 152, 205).

[133] Pierre-Olivier Rocher et al. « Video Stream Transmodality ». In : *ICEIS 2014-Proceedings of the 16th International Conference on Enterprise Information Systems*. 2014, p. 28–37 (cf. p. 144, 152, 205).

[134] Pierre-Olivier Rocher et al. « Video Stream Transmodality ». In : *Springer LNBIP Series Book*. Accepted, 2014 (cf. p. 152, 205).

[135] Thomas W Sederberg et al. « T-splines and T-NURCCs ». In : *ACM transactions on graphics (TOG)*. T. 22. 3. ACM. 2003, p. 477–484 (cf. p. 29).

[136] Simon Stegmaier, Marcelo Magallón et Thomas Ertl. « A generic solution for hardware-accelerated remote visualization ». In : *Proceedings of the symposium on Data Visualisation 2002*. Eurographics Association. 2002, 87–ff (cf. p. 64).

[137] *Studio encoding parameters of digital television for standard 4 :3 and wide-screen 16 :9 aspect ratios*. Rapp. tech. UIT-R BT.601-7. 2011 (cf. p. 17).

[138] Julien Subercaze et al. « WSRM - A linear-time algorithm for the rectangle decomposition of binary images ». In : Submitted, 2014 (cf. p. 116, 152, 181, 205).

[139] Tomáš Suk, Cyril Höschl IV et Jan Flusser. « Rectangular decomposition of binary images ». In : *Advanced Concepts for Intelligent Vision Systems*. Springer. 2012, p. 213–224 (cf. p. 178–180, 182).

[140] Jian Sun et al. « Image vectorization using optimized gradient meshes ». In : *ACM Transactions on Graphics (TOG)*. T. 26. 3. ACM. 2007, p. 11 (cf. p. 30, 32–34).

[141] Sriram Swaminarayan et Lakshman Prasad. « RaveGrid : Raster to Vector Graphics for Image Data ». In : *SVG Open*. 2007 (cf. p. 36).

[142] Nicolas Tizon, Christina Moreno et Marius Preda. « ROI based video streaming for 3D remote rendering ». In : *Multimedia Signal Processing (MMSP), 2011 IEEE 13th International Workshop on*. IEEE. 2011, p. 1–6 (cf. p. 77, 78).

[143] Nicolas Tizon et al. « MPEG-4-based adaptive remote rendering for video games ». In : *Proceedings of the 16th International Conference on 3D Web Technology*. ACM. 2011, p. 45–50 (cf. p. 75–77).

[144] Ana Paula Tomás et António Leslie Bajuelos. « Quadratic-time linear-space algorithms for generating orthogonal polygons with a given number of vertices ». In : *Computational Science and Its Applications–ICCSA 2004*. Springer, 2004, p. 117–126 (cf. p. 182).

[145] Jilin Tu et al. « Coding face at very low bit rate via visual face tracking ». In : *Proceeding of Picture Coding Symposium*. Citeseer. 2003, p. 301–304 (cf. p. 40, 57).

[146] Christian Wachinger et Nassir Navab. « Structural image representation for image registration ». In : *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*. IEEE. 2010, p. 23–30 (cf. p. 42).

[147] RK Wallis, WK Pratt et M Plotkin. « Video conferencing at 9600 bps ». In : *Proceedings of Picture Coding Symposium*. 1981, p. 104–105 (cf. p. 57).

[148] Guobao Wang et Jinyi Qi. « Direct reconstruction of dynamic PET parametric images using sparse spectral representation ». In : *Biomedical Imaging : From Nano to Macro, 2009. ISBI'09. IEEE International Symposium on*. IEEE. 2009, p. 867–870 (cf. p. 42).

[149] Tang Guo Wei, Wu Shuang et Zhang Yan. « An improved fast fractal image coding algorithm ». In : *Computer Science and Network Technology (ICCSNT), 2012 2nd International Conference on*. IEEE. 2012, p. 730–732 (cf. p. 42).

[150] Yosi Yomdin et al. « Synthesized textures in MPEG-4 ». In : *Image Processing. 2002. Proceedings. 2002 International Conference on*. T. 3. IEEE. 2002, p. III–21 (cf. p. 30, 32, 33).

[151] Chunbo Zhu et al. « Video coding with spatio-temporal texture synthesis ». In : *Multimedia and Expo, 2007 IEEE International Conference on*. IEEE. 2007, p. 112–115 (cf. p. 58).

Résumé de thèse

Ces dernières années, l'utilisation de la vidéo comme support de diffusion de l'information est devenue prépondérante. Selon certains analystes, d'ici 2017, environ 90% de la bande passante mondiale sera consommée par des services utilisant des flux vidéos. Basées sur ce genre de services, les solutions de *cloud gaming* se démocratisent. Ces solutions ont été imaginées dans un contexte de développement fort du paradigme de *cloud computing*, et elles ont été dopées par la prolifération des terminaux mobiles ainsi que par la qualité des réseaux qui ne cesse de croître. Les technologies mises en œuvre dans ce type de solutions se réfèrent au rendu à distance. Pour permettre au plus grand nombre l'accès à ce type d'applications, mais aussi pour maximiser le nombre de clients par serveur, il est primordial de maîtriser au mieux la bande passante nécessaire au bon fonctionnement du service.

Toutes les solutions de *cloud gaming* existantes utilisent une compression vidéo pour la transmission des images générées sur un serveur et à destination d'un client : le pixel règne en maître. Néanmoins, il existe bien d'autres façons de représenter une image numérique, notamment de manière paramétrique. Un certain nombre de travaux – à la fois sur l'image et la vidéo – montrent que cette approche est viable.

Dans cette thèse, nous proposons un espace de représentation hybride afin de réduire le débit binaire. Notre approche repose à la fois sur une approche pixel, mais aussi sur une approche paramétrique pour la compression d'un même flux vidéo. L'utilisation de deux systèmes de compression nécessite la définition de zones, auxquelles s'appliqueront les différents encodeurs. Pour le cas d'utilisation choisi, l'utilisateur étant un joueur impliqué de manière active dans la chaîne de rendu, il est intéressant d'utiliser un partitionnement de l'image dépendant des zones ou se porte son attention. Pour déterminer les zones importantes à ses yeux, un *eye-tracker* a été utilisé sur plusieurs jeux et par plusieurs testeurs. Cette étude permet de mettre en avant différentes corrélations, tant au niveau des caractéristiques des images que du type de jeu utilisé. Cette étude permet de connaître les zones que le joueur regarde ou ne regarde par directement (obtention des « cartes d'attention sélective »), et ainsi de gérer les encodeurs en conséquences. Nous établissons ensuite l'architecture et l'implémentation d'un tel encodeur multimodal (que nous appelons « transmodeur ») afin d'établir la preuve de réalisation d'un tel encodeur. Profitant alors de la maîtrise complète de l'implémentation, nous nous livrons ensuite à l'analyse de l'influence des paramètres de notre transmodeur quant à son efficacité au moyen d'une étude objective.

Le transmodeur a été intégré dans la chaîne de rendu utilisée par le projet XLcloud. Un certain nombre d'améliorations (au niveau des performances notamment) seront nécessaires pour une utilisation en production, mais il est dès à présent possible de l'utiliser de manière fluide en se limitant à des résolutions spatiales légèrement inférieures au 720p et à 30 images par seconde.

[ht]

FONDATION TELECOM

télécom saint-étienne
école d'ingénieurs
nouvelles technologies

TELECOM SudParis