Compression de contenus 360° et transmission adaptée à la navigation de l'utilisateur

Navid MAHMOUDIAN BIDGOLI, Thomas MAUGEY, Aline ROUMY

Inria, Univ Rennes, CNRS, IRISA
Campus de Beaulieu, 35042 Rennes
navid.mahmoudian-bidgoli@inria.fr, thomas.maugey@inria.fr,aline.roumy@inria.fr

Résumé – Dans cet article, nous proposons un nouveau schéma de compression pour les images omnidirectionnelles. En se basant sur des codes incrémentaux, ce schéma permet un encodage à la fois efficace et flexible, dans le sens où seule l'information affichée du côté de l'utilisateur peut être extraite et transmise à celui-ci. Les résultats expérimentaux montrent que notre schéma obtient de meilleurs débits de compression que les méthodes de référence. De plus, la transmission croit linéairement avec la taille de la requête et évite les effets de palier, ce qui en fait un codeur parfaitement adapté à la transmission interactive.

Abstract – In this paper, we propose a new compression scheme for omnidirectional images. Based on incremental coders, this scheme leads to an efficient and flexible coding, in the sense that only the information displayed at the user's side can be extracted and transmitted to him. The experimental results demonstrate that our coder obtains better compression ratio than the reference methods. Furthermore, the transmission cost grows linearly with the size of the request, which avoids a stair effect. This shows the perfect suitability of our coder to interactive transmission.

1 Introduction

Depuis l'arrivée de caméras 360° (appelées aussi omnidirectionnelles) à destination du grand public, la diffusion de contenus immersifs a explosé, en particulier sur les plateformes de vidéos. Dotées de plusieurs lentilles "œil-de-poisson", de miroirs incurvés, ou simplement de plusieurs caméras perspectives, ces caméras permettent de décrire l'activité lumineuse arrivant en un point central depuis *toutes les directions*. Autrement dit, l'image capturée et reconstruite après un certain nombre de post-traitements est une image sphérique. Afin de pouvoir être comprimée avec des outils standards et performants (*e.g.,* HEVC [3]), cette image sphérique est projetée sur un format image (ou vidéo) standard 2D. Deux projections sont à ce jour communément utilisées : la projection equirectangulaire (tel un planisphère) et la projection cubique (sur les 6 faces d'un cube) [1].

Une spécificité de ces contenus est que *par nature* une image ou vidéo 360° nécessite une visualisation *interactive*. En effet, contrairement à des images 2D classiques, une image 360° n'est observée, à un instant t, qu'en partie par un utilisateur. Ainsi, il est naturel d'essayer de n'envoyer à l'utilisateur que l'information visualisée par celui-ci afin de limiter le surcoût en bande-passante. Cette simple intuition est néanmoins extrêmement difficile à mettre en œuvre dans le cadre des codeurs actuels. En effet, ceux-ci procèdent à une subdivision de l'image par blocs et chacun est utilisé pour prédire ses voisins entraînant une grande dépendance entre eux. Ainsi, l'extraction d'un sous ensemble de ces blocs est quasiment impossible. Le seul moyen qui a été proposé dans la littérature est une partition de l'image d'origine en sous-images (*tile*) codées indépendamment [6]. Cette méthode est plus efficace que le codage de l'image intégrale car elle permet d'envoyer moins d'informations non visualisées par l'utilisateur. Néanmoins, cette approche basée partition présente deux défauts. Les sous-images ne correspondent pas forcément aux parties visualisées par l'utilisateur. Aussi, plusieurs sous-images doivent être transmises et la corrélation entre ces sous-images n'est pas exploitée. De plus, les sous-images transmises couvrent une zone plus grande que la zone visualisée et donc des informations inutiles sont encore transmises.

Dans cet article, nous proposons de revoir complètement la stratégie de codage, afin d'éviter la partition de l'image. Dans le codeur proposé, toutes les corrélations entre blocs sont prises en compte, et seuls les blocs nécessaires à la visualisation sont transmis à l'utilisateur. Cela est possible grâce au remplacement du codage entropique par un codage incrémental plus flexible. En effet, le codage entropique classique dépend du bloc voisin qui a été utilisé pour la prédiction. Le codeur incrémental, quant à lui, peut s'adapter à n'importe quel bloc voisin. Le codeur proposé est ainsi construit autour de ce codeur incrémental. Nous proposons alors de nouvelles méthodes de prédiction, un nouveau scan des blocs, et un nouveau positionnement des blocs codés indépendamment. Lors des tests, nous mettons en lumière les différents avantages du codeur proposé comparé à une méthode de partition classique.

Ce travail a été financé par le laboratoire d'excellence Cominlabs, supervisé par l'ANR dans le cadre du programme "Investing for the Future" avec la référénce ANR-10-LABX-07-01.

2 Compression interactive

Dans cette section, nous présentons le problème de compression interactive de contenus 360° (Figure 1). Nous introduisons en particulier les différents coûts qui caractérisent les performances d'un schéma de compression. Ceux-ci diffèrent de la compression classique du fait de la subdivision du débit en deux coûts distincts (le stockage et la transmission) ainsi que de la dépendance aux requêtes d'utilisateurs.

Soit Γ une image sphérique. Afin d'obtenir un format compatible avec les outils de compression classique, cette image est généralement projetée à l'aide d'une fonction :

$$m : \Gamma \mapsto m(\Gamma). \tag{1}$$

Un encodage de cette image 2D est ensuite effectué. Cet encodage est dit "hors-ligne" car il est effectué une fois pour toutes, avant toute requête utilisateur. La fonction d'encodage est définie comme suit :

$$f : m(\Gamma) \mapsto (b_1, \ldots, b_n), \tag{2}$$

où les b_i sont différents trains de bits qui sont indépendamment extractables. Le coût de stockage associé à cette fonction d'encodage est défini comme :

$$S = |(f \circ m)(\Gamma)| = \sum_{i=1}^{n} |b_i|, \tag{3}$$

où $|.|$ correspond à la taille (en bits) du train de bits. La quantité S correspond à la taille de la vidéo codée et stockée sur le serveur. Lors d'une requête d'utilisateur, seule une partie des trains de bits extractables est transmise. Lors de la visualisation d'une image omnidirectionnelle, un utilisateur a la possibilité de changer son orientation θ, correspondant à une sous-partie de la sphère $\gamma(\theta) \in \Gamma$. L'utilisateur communique au serveur la position de son angle de vue, et celui-ci effectue une opération d'extraction g_θ définie comme suit :

$$g_\theta = (b_1, \ldots, b_n) \mapsto \{b_i\}_{i \in \mathcal{I}_\theta}, \tag{4}$$

où $\mathcal{I}_\theta$ correspond à l'ensemble des indices des train de bits nécessaires à l'affichage de la portion de sphère $\gamma(\theta)$. Le coût de transmission est alors défini comme suit :

$$R_\theta = |g_\theta(b_1, \ldots, b_n)| = \sum_{i \in \mathcal{I}_\theta} |b_i|. \tag{5}$$

Les trains de bits sont alors décodés et la portion d'image 2D retrouvée est ensuite reprojetée sur la sphère, $\hat{\gamma}(\theta)$. Alors que certaines méthodes proposent d'évaluer la distorsion sur la sphère (ou même directement sur la projection 2D), nous optons, ici, pour une approche plus proche de la réalité : calculer la distorsion sur l'image affichée du côté de l'utilisateur. En effet, celui-ci n'observe pas directement l'image sphérique $\gamma(\theta)$ mais une projection de $\gamma(\theta)$ sur une fenêtre d'affichage $v(\theta)$. Cette fenêtre correspond formellement à un plan tangent à la sphère en θ, et en pratique à un dispositif d'affichage type visiocasque (HMD) ou simplement une fenêtre 2D. L'opération de projection de la sphère sur la fenêtre d'affichage est appelée ϕ_θ. La distortion est évaluée ainsi :

$$D_\theta = ||\phi_\theta(\gamma(\theta)) - \phi_\theta(\hat{\gamma}(\theta))||_2^2. \tag{6}$$

Il est important de noter que la distorsion et le coût de transmission dépendent de la requête. Il est ainsi nécessaire de calculer l'espérance de ces valeurs sur l'ensemble des requêtes des utilisateurs. Un codeur est ainsi caractérisé par les trois quantités

$$(S, \mathbb{E}_\theta(R_\theta), \mathbb{E}_\theta(D_\theta)).$$

Chacune d'entre elles doit être aussi petite que possible ou bien satisfaire des contraintes.

Comme indiqué dans l'introduction, les schémas existants garde une fonction d'encodage f conventionnelle (*e.g.,* HEVC) et permettent l'interactivité en partitionnant l'image projetée et en encodant séparément chaque sous-image. Ce partitionnement, bien que fortement utilisé, présente deux grands défauts. Tout d'abord, il empêche la fonction d'encodage d'exploiter les corrélations entre les sous-images. De plus, celui-ci étant constitué avant toute requête, ne correspond pas forcément avec la sous sphère demandée $\gamma(\theta)$. Ainsi plusieurs sous-images sont souvent nécessaires pour servir la requête, sous-images dont la corrélation n'est pas exploitée. De plus, une partie des pixels transmis et décodés sont inutiles car non-visualisés (*cf* Figure 2). Dans la section suivante, nous expliquons comment le codeur proposé rompt avec l'idée de partitionnement.

3 Codeur proposé

3.1 Principe général

La force des codeurs conventionnels réside dans la subdivision des images en blocs. Au fur et à mesure du codage, les blocs déjà encodés sont utilisés pour prédire les blocs voisins. Un résidu défini comme la différence entre le bloc d'origine et sa prédiction par ses voisins est alors calculé puis encodé grâce à un codage entropique (*e.g.,* arithmétique). Cette technique est extrêmement efficace lorsque le balayage des blocs est fixe, autrement dit lorsque, pour traiter un bloc donné, le même bloc voisin est utilisé par le codeur et le décodeur. Dans le cas d'une navigation interactive, au contraire, les blocs à transmettre et leur balayage dépendent de la requête. Par exemple, dans la Figure 2, seuls les blocs compris dans la zone verte devraient être transmis. Pour cela, il est nécessaire que le codage d'un bloc ne dépende pas de l'ordre de codage des blocs et donc de la prédiction qui est générée. Dans notre solution, nous considérons l'ensemble des ordres de décodage et donc l'ensemble des prédictions possibles au moment de l'encodage afin de d'être le plus adaptif possible du côté de l'utilisateur.

Dans des travaux théoriques précédents [2], il est montré que de tels codes, dits *incrémentaux*, permettent justement d'utiliser un train de bits astucieusement généré pour décoder des prédictions différentes. Encore mieux, si la qualité des prédictions est différente, il est même possible avec de tels codes, d'extraire seulement la quantité de bits nécessaire du train de bit incrémental stocké. Autrement dit, au moment du codage d'un bloc, l'encodeur peut lister toutes les prédictions possibles et préparer un train de bits incrémental tel que si une de ces

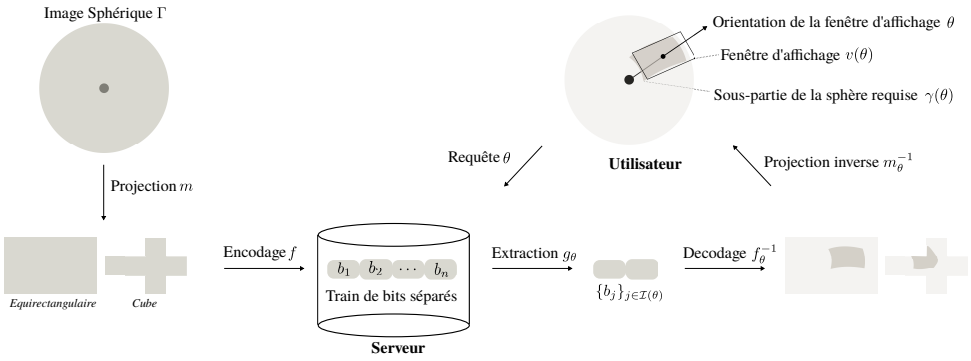

FIGURE 1 – Compression interactive de contenus 360°. L'ensemble de ce qui est encodé est stocké au niveau du serveur. En revanche, seule la partie extraite est transmise à l'utilisateur (en fonction de sa requête).

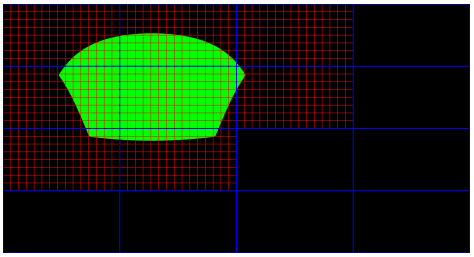

FIGURE 2 – Exemple de sous-optimalité des méthodes de codage par partionement. La zone en vert correspond à la requête $m(\gamma(\theta))$. Les frontières bleues délimitent le partitionnement. Les zones rouges indiquent les blocs transmis pour permettre l'affichage de la zone demandée.

prédictions est disponible chez l'utilisateur, ce bloc puisse être décodé en extrayant uniquement la partie nécessaire.

Changer la manière dont est codée un bloc nécessite, ensuite de revoir l'ensemble de la stratégie de codage. Bien que l'image soit encore subdivisée en blocs, il est nécessaire de lister l'ensemble des prédictions possibles pour chacun d'entre eux. Ensuite, comme dans les schémas conventionnels, un ou plusieurs blocs doivent être choisis comme étant codés en premier, les *checkpoints*, et donc indépendamment des autres. Nous étudions leur nombre et leurs positions dans l'image. Enfin, la troisième modification majeure est de revoir l'ordre de balayage des blocs, *i.e.,* comment parcourir l'ensemble des blocs à transmettre à partir du checkpoint. Dans la suite, nous détaillons chacune de ces modifications.

Stratégie de décodage : étant donnée une requête d'utilisateur (zone en vert dans la Figure 2), le décodage du côté de l'utilisateur s'effectue comme suit. Il est supposé qu'au moins un *checkpoint*, *i.e.,* un bloc encodé indépendamment, est présent dans la zone de requête (Section 3.3). Celui-ci est décodé dans un premier temps. Puis, à partir de ce bloc, une prédiction pour un bloc voisin est calculée. Cette prédiction est corrigée grâce au train de bits (Section 3.2) généré par le codeur incrémental, ce qui reconstitue le bloc voisin. Le décodage des blocs suivants est effectué de la même manière en propageant la zone des blocs décodés comme indiqué dans la Section 3.3.

3.2 Prédictions disponibles

Pour la méthode de prédiction, nous utilisons la méthode de prédiction intra utilisée dans HEVC et qui s'avère être très performante. Celle-ci fonctionne en propageant des pixels de référence selon une direction désignée par le "mode" (il y en a 36 disponibles). Ces pixels de référence sont justement les pixels disponibles dans les blocs voisins.

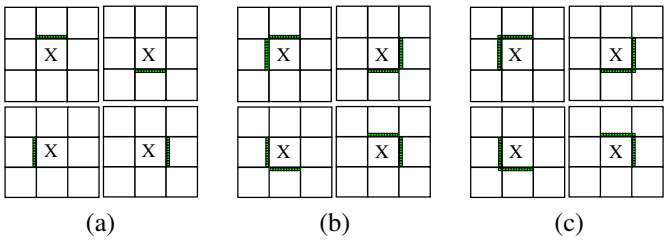

FIGURE 3 – Pixels de référence (en vert) utilisés pour générer la prédiction du bloc X pour chaque type de prédiction (a) Type 1, (b) Type 2 (c) Type 3.

Nous montrons dans la Figure 3 l'ensemble des cas de figure disponibles. Il y en a de trois types selon qu'un, deux ou trois blocs voisins sont disponibles. Pour chacun des types, 4 orientations sont possibles. Au total 12 prédictions sont en général envisageables pour chaque bloc.

3.3 Choix des checkpoints et ordre de balayage

Les checkpoints doivent être choisis parmi les blocs de manière à ce qu'ils soient le moins nombreux possible (fort coût d'encodage) et qu'au moins un checkpoint soit toujours disponible dans une requête d'utilisateur. Pour obtenir la solution montrée dans la Figure 4, nous avons supposé une taille standard de fenêtre d'affichage. Puis nous avons balayé la sphère et introduit un checkpoint dès qu'un précédent n'était plus visible dans la fenêtre. La position des checkpoints obtenus correspond en fait à un échantillonnage uniforme de la sphère. Le choix de la méthode de balayage au décodeur, est fait de manière à maximiser le nombre de pixels de référence lors de la prédiction (avoir le plus grand nombre de prédictions de type 3 au décodage). L'algorithme développé choisit à chaque étape

le prochain bloc à décoder ayant le plus de blocs voisins déjà décodés et disponibles.

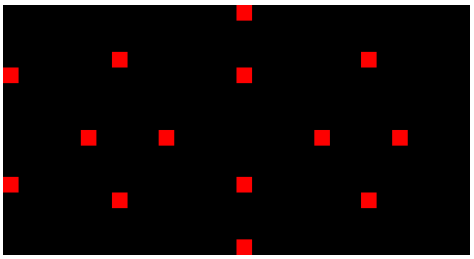

FIGURE 4 – Position des checkpoints utilisés dans notre codeur.

4 Résultats expérimentaux et conclusion

Pour valider l'intérêt de notre méthode, nous confrontons celle-ci à plusieurs approches : celles qui effectuent des partitions, et codent chaque sous-image séparément (T._ × _), celles qui effectuent un *Stockage exhaustif (ES)* de chaque résidu correspondant à chaque prédiction. Pour notre méthode, *Rate Adaptive (RA)*, nous présentons les résultats théoriques, puis deux implémentations de codes incrémentaux (LDPCA [4], et protographe [5]). Nous présentons d'abord en Figure 5, l'évolution du débit, en temps réel lors d'une requête d'un utilisateur. Nous pouvons clairement observer qu'une méthode basée partitions a une transmission de l'information par palier, contrairement à notre approche qui envoie, au fur et à mesure de la navigation de l'utilisateur, l'information nécessaire et a donc une forme linéaire plus convenable pour de la transmission adaptée à la requête.

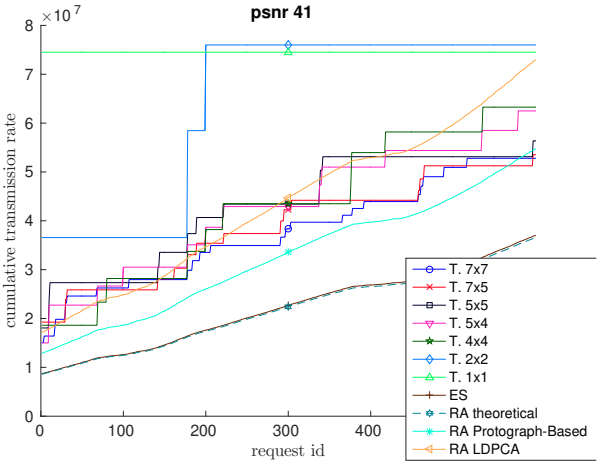

FIGURE 5 – Somme cumulative du débit de transmission pour une requête de 600 points de vues, pour différentes méthodes.

Dans la Figure 6, nous présentons les résultats de stockage et débit. L'image 360° a d'abord été encodée et stockée sur un serveur. Les performances de compression sont affichées dans la courbe (S,PSNR). Ensuite, des navigations de 10 angles de vue de 20 utilisateurs sont simulées et les débits nécessaires à celles-ci sont évalués. Ceux-ci sont rapportés dans les courbes ($E_\theta(R_\theta)$), PSNR). Nous pouvons voir dans ces deux courbes que les méthodes proposées présentent des débits qui sont bien performants avec seulement un petit surcoût en stockage.

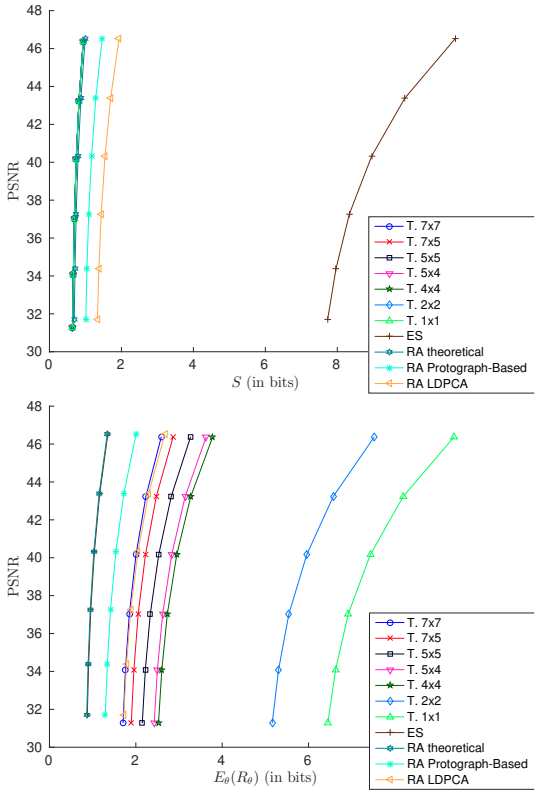

FIGURE 6 – Comparaison des performances de stockage et transmission pour différentes méthodes.

Références

[1] E. Kuzyakov and D. Pio. Next-generation video encoding techniques for 360 video and vr, 2016.

[2] A. Roumy and T. Maugey. Universal lossless coding with random user access : The cost of interactivity. In *2015 IEEE International Conference on Image Processing (ICIP)*, pages 1870–1874, Sep. 2015.

[3] G. J. Sullivan, J. Ohm, W. Han, and T. Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12) :1649–1668, Dec 2012.

[4] David Varodayan, Anne Aaron, and Bernd Girod. Rate-adaptive codes for distributed source coding. *Signal Processing*, 86(11) :3123 – 3130, 2006. Special Section : Distributed Source Coding.

[5] Fangping Ye, Elsa Dupraz, Zeina Mheich, and Karine Amis. Optimized rate-adaptive protograph-based LDPC codes for source coding with side information. *ArXiv*, abs/1808.06509, 2018.

[6] Alireza Zare, Alireza Aminlou, Miska M. Hannuksela, and Moncef Gabbouj. Hevc-compliant tile-based streaming of panoramic video for virtual reality applications. In *Proceedings of the 24th ACM International Conference on Multimedia*, pages 601–605, 2016.