RFC 6386 : VP8 Data Format and Decoding Guide

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 11 novembre 2011

Date de publication du RFC : Novembre 2011

https://www.bortzmeyer.org/6386.html

Le monde des formats vidéo est très hostile pour ceux qui apprécient la liberté et l'ouverture : formats privés, non documentés, contrôlés par une seule compagnie, plombé de brevets tous plus futiles les uns que les autres... Aujourd'hui, il est à peu près impossible de regarder une vidéo, quelle qu'elle soit, avec du logiciel entièrement libre, et sans enfreindre une centaine de brevets. Ce RFC documente un des très rares codecs vidéos ouverts, qui puisse être mis en œuvre dans du logiciel libre sans trop de crainte juridique. Il a été produit par Google, documenté pour l'IETF et se nomme VP8.

Lorsqu'on configure une machine utilisant du logiciel libre aujourd'hui, si on s'en tient strictement à ses principes, et qu'on refuse d'intégrer les sources de logiciel que Debian ou Ubuntu marquent comme non-libres, on n'arrivera pas à voir beaucoup de films... Même si le format du container vidéo est relativement ouvert, les techniques de compression ne le sont pas et on constate vite que son MPlayer ou son vlc n'arrivent pas à lire grand'chose. (C'est particulièrement vrai pour les films achetés légalement, pour lesquels l'usage de formats fermés sert de DRM. Les films partagés gratuitement sur le réseau réservent en général moins de mauvaises surprises.) Le problème de la personne qui produit le film est le manque de formats ouverts dans ce domaine. Le principal est Ogg (avec le codec vidéo Theora) mais les experts lui reprochent des performances en compression et décompression qui ne sont pas extraordinaires. Google prétend avoir une solution, son format VP8 (avec le format de container WebM), supposé bien meilleur (il a été développé à l'origine par On2). Mais utiliser un format spécifique à Google n'est pas mieux qu'utiliser un format spécifique à Adobe. Alors, Google a produit un RFC, soumis à l'IETF (en termes IETF, c'est une « soumission indépendante », pas le résutat d'un groupe de travail IETF) et qui devient donc un format ouvert, libre de la compagnie qui lui a donné naissance <http://www.ecrans.fr/Google-libere-le-codec-VP8,9942.html>. Ce RFC inclut également une mise en œuvre de référence du décodeur, distribuée sous une licence libre (la BSD, cf. section 20.26).

La question des brevets est traditionnellement une de celles qui plombent le plus les projets de codecs ouverts. La section 20.27 décrit en détail la situation pour VP8. Google promet une licence gratuite d'exploitation de ses brevets pour toute implémentation de VP8. Mais il n'est pas clair, pour un non-juriste, si

cela couvre la possibilité de versions ultérieures de VP8, qui ne seraient pas approuvées par Google. En attendant, pas mal de requins pourraient attaquer VP8 <http://www.numerama.com/magazine/18060-webm-vp8-l-action-en-justice-contre-google-se-prepare.html>. (Un bon article de synthèse est « "*Video codecs : The ugly business behind pretty pictures*" <http://www.infoworld.com/d/open-source-software/video-codecs-the-ugly-business-behind-pretty-pictures-2145

Ce RFC 6386[1] est essentiellement composé de code. VP8 est en effet défini par son implémentation de référence (la libvpx <http://www.webmproject.org/code/>), le texte n'étant là que pour expliquer. Le RFC note bien (section 1) que, s'il y a une contradiction entre le code et le texte, c'est le code qui a raison (choix inhabituel à l'IETF). Avec ses 310 pages, c'est un des plus longs RFC existants. Je ne vais donc pas le traiter dans sa totalité, d'autant plus que les codecs vidéos ne sont pas ma spécialité.

Donc, en simplifiant à l'extrême, VP8 décompose chaque image d'un flux vidéo en une série de sous-blocs, chacun des sous-blocs étant composé de pixels. La compression d'un sous-bloc dépend des sous-blocs précédents, pour bénéficier de la redondance temporelle de la vidéo (sauf dans les films avec Bruce Willis, il y a relativement peu de changements d'une image à la suivante, et en général sur une partie de l'image seulement). Il y a aussi une redondance spatiale (des parties de l'image qui sont répétées, par exemple un grand ciel noir pour les scènes de nuit). La technique de compression se nomme transformée en cosinus discrète, mais VP8 utilise également à certains endroits une autre technique, la transformée de Walsh-Hadamard. VP8 ne permet pas de reconstituer exactement les pixels originaux, il compte sur la tolérance de l'œil humain (qui peut accepter certains changements sans tiquer).

Contrairement à son concurrent MPEG, VP8 encode et décode uniquement avec des nombres entiers et ne perd donc pas de précision dans les calculs en virgule flottante. Ces pertes peuvent parfois provoquer des effets surprenants, visibles à l'écran (rappelez-vous que, dans l'arithmétique en virgule flottante, il y a moins d'assertions vérifiables. Par exemple, 1 - x + x n'est pas forcément égal à 1).

La section 2 du RFC expose les grandes lignes du format : utilisation de l'espace YUV en 4 :2 :0 8bits, c'est-à-dire que chaque pixel dans les deux plans chromatiques (U et V) fait huit bits et correspond à un bloc de quatre pixels dans le plan de lumière Y. Les pixels sont ensuite transmis en lignes, de haut en bas, chaque ligne étant transmise de gauche à droite. Les informations permettant le décodage ne sont pas au niveau du pixel mais à celui de blocs de pixels. Dans l'implémentation de référence, la description du format se trouve dans le fichier d'en-tête `vpx_image.h`.

Une fois comprimées, il y a deux types de trames dans un flux VP8 (section 3) : les intratrames (ou trames clés, ce que MPEG appelle les trames-I) sont complètes et n'ont pas besoin d'autres trames pour être décodées. La première trame envoyée est évidemment une intratrame. Et le deuxième type sont les intertrame (trames-P chez MPEG), qui codent une différence par rapport à la précédente intratrame, et aux autres intertrames arrivées entre temps. La perte d'une intertrame empêche donc de décoder les suivantes, jusqu'à l'arrivée de la prochaine intratrame, qui permettra au décodeur de repartir d'un état connu.

À noter que, contrairement à MPEG, il n'y a pas de trame-B, trame permettant d'indiquer le futur. En revanche, VP8 a des « trames en or » et des « trames de référence de secours » qui sont des intertrames contenant suffisamment d'informations pour permettre de reconstituer les quelques trames suivantes. Elles permettent à VP8 d'être plus tolérant aux pertes de trame (normalement, lorsqu'un perd une intratrame, il n'y a rien d'autre à faire qu'à attendre la suivante).

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc6386.txt`

Le format des trames compressées est décrit en section 4. Un en-tête (plus grand dans le cas des intratrames), et les données. Bon, en fait, c'est un peu plus compliqué que cela car, au sein des données, les macroblocs ont aussi leur en-tête. Pour décoder (section 5), le programme doit garder quatre tampons YUV : la trame actuellement en cours de reconstruction, la trame précédente (rappelez-vous que les intertrames sont uniquement des différences par rapport à la trame précédente, qu'il faut donc garder en mémoire), et les plus récentes trames en or et de référence de secours. Le principe est simple (on réapplique les différences à la trame de référence, obtenant ainsi la trame originale), mais il y plein de pièges, dus par exemple à l'abondance du contexte à maintenir (puisque le décodage dépend du contexte en cours). Le code se trouve dans `dixie.c` dans le RFC (l'implémentation de référence a été réorganisée et ce fichier n'y figure plus). Une façon de simplifier l'écriture d'un décodeur est de commencer par ne programmer que le décodage des intratrames, avant d'ajouter celui des trames intermdiaires. C'est d'ailleurs le plan que choisit le RFC.

Le langage de l'implémentation de référence est le C. Comme le note la section 6, c'est en raison de sa disponibilité universelle et de sa capacité à décrire exactement la précision des calculs qu'il a été choisi. Notez que le code C dans le RFC n'est **pas** exactement celui de l'implémentation de référence (qui est, elle, disponible en <http://www.webmproject.org/code/>). Plusieurs raisons à cela, notamment le fait que l'implémentation de référence a été optimisée pour la performance, le code dans le RFC pour la lisibilité, Mais les deux codes produisent le même résultat. Rappelons enfin que VP8 n'utilise pas de flottants, seulement des entiers.

Le code commence en section 7, avec le décodeur booléen. L'idée est que l'encodage est fait en produisant une série de booléens, avec pour chacun une probabilité d'être à zéro ou à un. Si la probabilité était de 0,5, aucune compression ne serait possible (il faudrait indiquer explicitement tous les booléens) mais ici, les probabilités sont souvent très éloignées de 0,5 (en raison des cohérences spatiales et temporelles du flux vidéo) et VP8 consomme donc moins d'un bit par booléen. Travailler sur bits étant typiquement peu efficace sur la plupart des processeurs, le code en section 7.3 les regroupe en octets.

Le reste du RFC est essentiellement composé de code C avec commentaires. Je saute directement à la section 19 / annexe A qui donne la liste complète des données encodées, Tous les objets manipulés par VP8 sont là.

Notez que le projet parent, WebM, utilise actuellement Vorbis pour l'audio (VP8 ne faisant que la vidéo). L'IETF a un projet de développement d'un codec audio, les détails figurent dans le RFC 6366.

Pour plus d'information, on peut consulter le site officiel du projet WebM <http://www.webmproject.org> qui contient entre autres une liste des outils disponibles <http://www.webmproject.org/tools/>. Pour le déploiement de WebM, on peut voir la page du Wikipédia anglophone. Une analyse (critique) de VP8 a été faite par Jason Garnett-Glaser <http://www.macgeneration.com/unes/voir/127991/webm-un-nouveau-pretendant-pour-le-html5>.

Parmi les autres mises en œuvre de VP8, un cas curieux, un décodeur en JavaScript <http://bemasc.net/wordpress/2011/11/30/route9-js/>...