

Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: April 21, 2016

T. Daede  
J. Moffitt  
Mozilla  
October 19, 2015

Video Codec Testing and Quality Measurement  
draft-daede-netvc-testing-02

Abstract

This document describes guidelines and procedures for evaluating an internet video codec specified at the IETF. This covers subjective and objective tests, test conditions, and materials used for the test.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 21, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Subjective Metrics . . . . .	2
2.1. Still Image Pair Comparison . . . . .	2
3. Objective Metrics . . . . .	3
3.1. PSNR . . . . .	3
3.2. PSNR-HVS-M . . . . .	3
3.3. SSIM . . . . .	4
3.4. Fast Multi-Scale SSIM . . . . .	4
4. Comparing and Interpreting Results . . . . .	4
4.1. Graphing . . . . .	4
4.2. Bjontegaard . . . . .	4
4.3. Ranges . . . . .	5
5. Test Sequences . . . . .	5
5.1. Sources . . . . .	5
5.2. Test Sets . . . . .	6
5.3. Operating Points . . . . .	7
5.3.1. High Latency . . . . .	7
5.3.2. Unconstrained Low Latency . . . . .	8
5.3.3. Constrained Low Latency . . . . .	8
6. Automation . . . . .	8
7. Informative References . . . . .	9
Authors' Addresses . . . . .	10

## 1. Introduction

When developing an internet video codec, changes and additions to the codec need to be decided based on their performance tradeoffs. In addition, measurements are needed to determine when the codec has met its performance goals. This document specifies how the tests are to be carried about to ensure valid comparisons and good decisions.

## 2. Subjective Metrics

Subjective testing is the preferable method of testing video codecs.

Because the IETF does not have testing resources of its own, it has to rely on the resources of its participants. For this reason, even if the group agrees that a particular test is important, if no one volunteers to do it, or if volunteers do not complete it in a timely fashion, then that test should be discarded. This ensures that only important tests be done in particular, the tests that are important to participants.

### 2.1. Still Image Pair Comparison

A simple way to determine superiority of one compressed image over another is to visually compare two compressed images, and have the viewer judge which one has a higher quality. This is mainly used for rapid comparisons during development. For this test, the two compressed images should have similar compressed file sizes, with one image being no more than 5% larger than the other. In addition, at least 5 different images should be compared.

### 3. Objective Metrics

Objective metrics are used in place of subjective metrics for easy and repeatable experiments. Most objective metrics have been designed to correlate with subjective scores.

The following descriptions give an overview of the operation of each of the metrics. Because implementation details can sometimes vary, the exact implementation is specified in C in the Daala tools repository [DAALA-GIT].

All of the metrics described in this document are to be applied to the luma plane only. In addition, they are single frame metrics. When applied to the video, the scores of each frame are averaged to create the final score.

Codecs are allowed to internally use downsampling, but must include a normative upsampler, so that the metrics run at the same resolution as the source video. In addition, some metrics, such as PSNR and FASTSSIM, have poor behavior on downsampled images, so it must be noted in test results if downsampling is in effect.

#### 3.1. PSNR

PSNR is a traditional signal quality metric, measured in decibels. It is directly derived from mean square error (MSE), or its square root (RMSE). The formula used is:

$$20 * \log_{10} ( \text{MAX} / \text{RMSE} )$$

or, equivalently:

$$10 * \log_{10} ( \text{MAX}^2 / \text{MSE} )$$

which is the method used in the `dump_psnr.c` reference implementation.

#### 3.2. PSNR-HVS-M

The PSNR-HVS metric performs a DCT transform of 8x8 blocks of the image, weights the coefficients, and then calculates the PSNR of

those coefficients. Several different sets of weights have been considered. [PSNRHVS] The weights used by the dump\_pnsrhvs.c tool in the Daala repository have been found to be the best match to real MOS scores.

### 3.3. SSIM

SSIM (Structural Similarity Image Metric) is a still image quality metric introduced in 2004 [SSIM]. It computes a score for each individual pixel, using a window of neighboring pixels. These scores can then be averaged to produce a global score for the entire image. The original paper produces scores ranging between 0 and 1.

For the metric to appear more linear on BD-rate curves, the score is converted into a nonlinear decibel scale:

$$-10 * \log_{10} (1 - \text{SSIM})$$

### 3.4. Fast Multi-Scale SSIM

Multi-Scale SSIM is SSIM extended to multiple window sizes [MSSSIM]. This is implemented in the Fast implementation by downscaling the image a number of times, and computing SSIM over the same number of pixels, then averaging the SSIM scores together [FASTSSIM]. The final score is converted to decibels in the same manner as SSIM.

## 4. Comparing and Interpreting Results

### 4.1. Graphing

When displayed on a graph, bitrate is shown on the X axis, and the quality metric is on the Y axis. For clarity, the X axis bitrate is always graphed in the log domain. The Y axis metric should also be chosen so that the graph is approximately linear. For metrics such as PSNR and PSNR-HVS, the metric result is already in the log domain and is left as-is. SSIM and FASTSSIM, on the other hand, return a result between 0 and 1. To create more linear graphs, this result is converted to a value in decibels:

$$-1 * \log_{10} (1 - \text{SSIM})$$

### 4.2. Bjontegaard

The Bjontegaard rate difference, also known as BD-rate, allows the comparison of two different codecs based on a metric. This is commonly done by fitting a curve to each set of data points on the plot of bitrate versus metric score, and then computing the difference in area between each of the curves. A cubic polynomial

fit is common, but will be overconstrained with more than four samples. For higher accuracy, at least 10 samples and a linear piecewise fit should be used. In addition, if using a truncated BD-rate curve, there should be at least 4 samples within the point of interest.

#### 4.3. Ranges

The curve is split into three regions, for low, medium, and high bitrate. The ranges are defined as follows:

- o Low bitrate: 0.005 - 0.02 bpp
- o Medium bitrate: 0.02 - 0.06 bpp
- o High bitrate: 0.06 - 0.2 bpp

Bitrate can be calculated from bits per pixel (bpp) as follows:

$$\text{bitrate} = \text{bpp} * \text{width} * \text{height} * \text{framerate}$$

### 5. Test Sequences

#### 5.1. Sources

Lossless test clips are preferred for most tests, because the structure of compression artifacts in already-compressed clips may introduce extra noise in the test results. However, a large amount of content on the internet needs to be recompressed at least once, so some sources of this nature are useful. The encoder should run at the same bit depth as the original source. In addition, metrics need to support operation at high bit depth. If one or more codecs in a comparison do not support high bit depth, sources need to be converted once before entering the encoder.

The JCT-VC standards organization includes a set of standard test clips for video codec testing, and parameters to run the clips with [L1100]. These clips are not publicly available, but are very useful for comparing to published results.

Xiph publishes a variety of test clips collected from various sources.

The Blender Open Movie projects provide a large test base of lossless cinematic test material. The lossless sources are available, hosted on Xiph.

## 5.2. Test Sets

Sources are divided into several categories to test different scenarios the codec will be required to operate in. For easier comparison, all videos in each set should have the same color subsampling, same resolution, and same number of frames. In addition, all test videos must be publicly available for testing use, to allow any results

- o Still images are useful when comparing intra coding performance. Xiph.org has four sets of lossless, one megapixel images that have been converted into YUV 4:2:0 format. There are four sets that can be used:
  - \* subset1 (50 images)
  - \* subset2 (50 images)
  - \* subset3 (1000 images)
  - \* subset4 (1000 images)
- o video-hd-2, a set that consists of the following 1920x1080 clips from [DERFVIDEO], cropped to 50 frames (and converted to 4:2:0 if necessary)
  - \* aspen
  - \* blue\_sky
  - \* crowd\_run
  - \* ducks\_take\_off
  - \* factory
  - \* life
  - \* old\_town\_cross
  - \* park\_joy
  - \* pedestrian\_area
  - \* red\_kayak
  - \* riverbed

- \* rush\_hour
- \* station2
- o A video conferencing test set, with 1280x720 content at 60 frames per second. Unlike other sets, the videos in this set are 10 seconds long.
- \* TBD
- o Game streaming content: 1920x1080, 60 frames per second, 4:2:0 chroma subsampling. 1080p is chosen as it is currently the most common gaming monitor resolution [STEAM]. All clips should be two seconds long.
- \* TBD
- o Screensharing content is low framerate, high resolution content typical of a computer desktop.
- \* screenshots - desktop screenshots of various resolutions, with 4:2:0 subsampling
- \* Video sets TBD

### 5.3. Operating Points

Two operating modes are defined. High latency is intended for on demand streaming, one-to-many live streaming, and stored video. Low latency is intended for videoconferencing and remote access.

#### 5.3.1. High Latency

The encoder should be run at the best quality mode available, using the mode that will provide the best quality per bitrate (VBR or constant quality mode). Lookahead and/or two-pass are allowed, if supported. One parameter is provided to adjust bitrate, but the units are arbitrary. Example configurations follow:

- o x264: -crf=x
- o x265: -crf=x
- o daala: -v=x
- o libvpx: -codec=vp9 -end-usage=q -cq-level=x -lag-in-frames=25 -auto-alt-ref=1

### 5.3.2. Unconstrained Low Latency

The encoder should be run at the best quality mode available, using the mode that will provide the best quality per bitrate (VBR or constant quality mode), but no frame delay, buffering, or lookahead is allowed. One parameter is provided to adjust bitrate, but the units are arbitrary. Example configurations follow:

- o x264: -crf-x -tune zerolatency
- o x265: -crf=x -tune zerolatency
- o daala: -v=x
- o libvpx: -codec=vp9 -end-usage=q -cq-level=x -lag-in-frames=0  
-auto-alt-ref=0

### 5.3.3. Constrained Low Latency

The encoder is given one parameter, which is absolute bitrate. No frame delay, buffering, or lookahead is allowed. The maximum achieved bitrate deviation from the supplied parameter is determined by a buffer model:

- o The buffer starts out empty.
- o After each frame is encoded, the buffer is filled by the number of bits spent for the frame.
- o The buffer is then emptied by (bitrate \* frame duration) bits.
- o The buffer fill level is checked. If it is over the limit, the test is considered a failure.

The buffer size limit is defined by the bitrate target \* 0.3 seconds.

## 6. Automation

Frequent objective comparisons are extremely beneficial while developing a new codec. Several tools exist in order to automate the process of objective comparisons. The Compare-Codecs tool allows BD-rate curves to be generated for a wide variety of codecs [COMPARECODECS]. The Daala source repository contains a set of scripts that can be used to automate the various metrics used. In addition, these scripts can be run automatically utilizing distributed computer for fast results [AWCY].



## 7. Informative References

- [AWCY] Xiph.Org, "Are We Compressed Yet?", 2015, <<https://arewecompressedyet.com/>>.
- [COMPARECODECS] Alvestrand, H., "Compare Codecs", 2015, <<http://compare-codecs.appspot.com/>>.
- [DAALA-GIT] Xiph.Org, "Daala Git Repository", 2015, <<http://git.xiph.org/?p=daala.git;a=summary>>.
- [DERFVIDEO] Terriberry, T., "Xiph.org Video Test Media", n.d., <<https://media.xiph.org/video/derf/>>.
- [FASTSSIM] Chen, M. and A. Bovik, "Fast structural similarity index algorithm", 2010, <[http://live.ece.utexas.edu/publications/2011/chen\\_rtip\\_2011.pdf](http://live.ece.utexas.edu/publications/2011/chen_rtip_2011.pdf)>.
- [L1100] Bossen, F., "Common test conditions and software reference configurations", JCTVC L1100, 2013, <<http://phenix.int-evry.fr/jct/>>.
- [MSSSIM] Wang, Z., Simoncelli, E., and A. Bovik, "Multi-Scale Structural Similarity for Image Quality Assessment", n.d., <<http://www.cns.nyu.edu/~zwang/files/papers/msssim.pdf>>.
- [PSNRHVS] Egiazarian, K., Astola, J., Ponomarenko, N., Lukin, V., Battisti, F., and M. Carli, "A New Full-Reference Quality Metrics Based on HVS", 2002.
- [SSIM] Wang, Z., Bovik, A., Sheikh, H., and E. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity", 2004, <<http://www.cns.nyu.edu/pub/eero/wang03-reprint.pdf>>.
- [STEAM] Valve Corporation, "Steam Hardware & Software Survey: June 2015", June 2015, <<http://store.steampowered.com/hwsurvey>>.

Authors' Addresses

Thomas Daede  
Mozilla

Email: tdaede@mozilla.com

Jack Moffitt  
Mozilla

Email: jack@metajack.im

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: January 7, 2016

T. Davies  
Cisco  
October 19, 2015

Interpolated reference frames for video coding  
draft-davies-netvc-irfvc-00

Abstract

This document describes the use of interpolated reference frames in video coding in general, and in the Thor video codec in particular.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 7, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Definitions . . . . .	3
2.1. Requirements Language . . . . .	3
2.2. Terminology . . . . .	3
3. The interpolation process . . . . .	3
3.1. Interpolation framework . . . . .	3
3.2. Motion estimation process . . . . .	4
3.3. Complexity considerations . . . . .	5
4. Coding using interpolated reference frames . . . . .	6
5. Compression performance . . . . .	6
6. IANA Considerations . . . . .	8
7. Security Considerations . . . . .	8
8. Acknowledgements . . . . .	8
9. Normative References . . . . .	8
Author's Address . . . . .	8

## 1. Introduction

This document describes a method of generating synthetic reference frames for video coding using a simplified frame interpolation method. The aim is to create a reference frame that is temporally co-located with the current frame being predicted, leveraging the motion information already present in the previously-coded frames, and removing the need for techniques such as motion vector scaling in motion vector prediction.

Since the decoder will have to generate the same interpolated reference frame as the encoder, complexity considerations are a paramount concern. The interpolation process is therefore a highly simplified block-matching algorithm and uses only pixel-accurate motion vectors, for example. Worst-case complexity can be managed by controlling the number of matches per block, per region and per frame as well as the total vertical excursion to manage memory bandwidth.

The method gives most gain in Thor at high quantisation (QP) levels i.e. low bitrates. Overall, Bjontegaard delta-rate (BDR) reductions across QP ranges 22-37 are on average 5.2% for a range of HD test sequences. For higher QP (32-44) the reductions gains are larger: 8.8% on average.

Interpolated reference frames are enabled by default in the high complexity random access (RA) and High Delay B (HDB) configurations in the Thor repository [github.com/cisco/thor](https://github.com/cisco/thor).

Section 3 describes the interpolation process, which is based on a simplified hierarchical motion estimation (HME). Section 4 describes the modifications to the Thor syntax coding processes. Section 5 provides details of compression performance.

## 2. Definitions

### 2.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

### 2.2. Terminology

This document frequently uses the following terms.

MV: Motion vector - a horizontal and vertical vector displacement (x,y)

ME: Motion Estimation

HME: Hierarchical ME

SAD: Sum of Absolute Differences. A metric defined for a pair of equal dimension blocks of numerical values consisting of the sum of the absolute differences of the corresponding values in each location in the blocks

QP: quantisation parameter

BDR: Bjontegaard Delta-Rate

## 3. The interpolation process

### 3.1. Interpolation framework

Consider two frames R0 and R1 and a frame F equidistant in time between them which is to be interpolated (Figure 1). Image data must be created for each block in F by combining information from R0 and R1 using a linear model for the block motion.

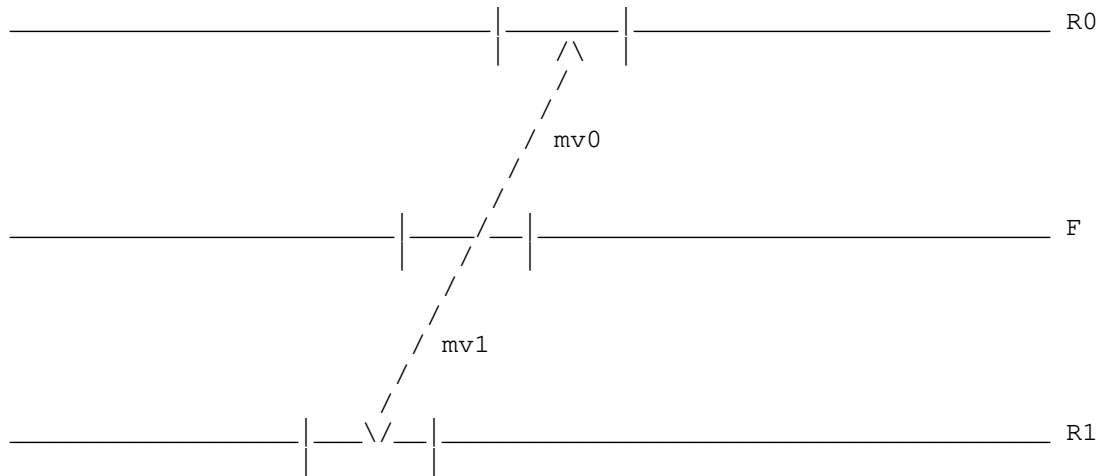


Figure 1: forward and backward motion pairs for a block

For each block in the frame F there is an associated motion vector  $mv0$  pointing at a displaced block in R0 and a corresponding motion vector  $mv1$  which is equal to  $-mv0$  pointing at R1.

Where F is not equidistant from the reference frames the linear model can simply be scaled appropriately.

If both blocks fall within the reference frames, then the interpolated block is just the average of the two reference blocks. At the edges of the frames one of the reference blocks may fall off the edge - here the other reference only is used instead.

### 3.2. Motion estimation process

Since F does not exist the motion estimation process consists of matching blocks  $B+mv0$  in R0 with blocks  $B+mv1$  in R1. A basic block size of  $8 \times 8$  is used but the bulk of the motion estimation is done for  $16 \times 16$  blocks. For UHD resolutions, perhaps a larger basic block size would be better. The overall approach is to use hierarchical motion estimation (HME), as this is amenable to limiting both average and worst-case complexity.

In the HME scheme each reference frame is down-scaled vertically and horizontally by a factor 2, using a  $(1/2, 1/2)$  filter. This is done repeatedly to get a series  $R0(n)$  and  $R1(n)$  of reference frames. Then motion estimation is done very simply on each resolution layer  $n$ , but

using candidates from next layer (n+1) as well as spatial neighbours. The block sizes are the same at each layer, so each block at layer n+1 corresponds to 4 blocks at layer n.

For each layer, the ME stages are as follows:

1. For each 16x16 block in raster order:
  - a. Check if ME can be bypassed.
  - b. If not bypassed, determine candidates from lower layer blocks and from neighbour blocks in raster order
  - c. Perform an adaptive cross search around each candidate vector and determine the best vector
2. For each 8x8 block in raster order, find the best merge candidate, i.e. choose which MV to use: the original 16x16 block vector, or one of 4 neighbouring block 16x16 vectors (above, below, left or right)

The majority of blocks bypass ME at step 1a. Here a skip candidate is generated as:

$$\text{skipmv} = \operatorname{argmin}\{\text{mvx in } \{\text{mv0}, \text{mv1}, \text{mv2}\}: \sum_{i=0}^2 |\text{mvx} - \text{mvi}|\}$$

where mv0, mv1, and mv2 are the motion vectors for blocks above, left and above-right the current block. If the cost for this vector is below a fixed value for each 8x8 sub-block, no further ME is done.

In step 1c, the ranges of the cross search are restricted to just 2 steps (max 8 matches) for each candidate, if the search is not at the lowest resolution layer. This is because vector candidates from the lower layer or from neighbours will already be highly accurate by this point.

In step 1, the cost metric is a combination of luma SAD and a fixed multiple of the sum of absolute motion vector difference between the vector mvx and the four neighbours mv0, mv1, mv2, mv3 to the left, right, above and above right, i.e.

$$\sum_{i=0}^3 |\text{mvx} - \text{mvi}|$$

This helps make the motion estimation process less sensitive to noise and spurious matches.

In step 2 the cost metric is SAD alone.

### 3.3. Complexity considerations

The ME process is not that sensitive to the selection of candidates,

at least in terms of the impact on coding performance. If the interpolated frames are used directly this might not be so, but in effect the interpolated blocks are only going to be used for prediction if they are interpolated well: therefore effort refining bad matches is generally wasted, so should be avoided.

This means that the ME process can be quite truncated. The only candidates considered are up to three neighbour block candidates and one from the layer below. The majority of motion estimation is skipped, and so only requires a single match. For HW applications the total number of matches would still require a hard limit, as well as limits for the matches per block and possibly per region. Vertical motion vector limits could also be imposed to reduce memory bandwidth costs.

#### 4. Coding using interpolated reference frames

In the Thor implementation, when an interpolated reference frame is used it is inserted at the beginning of the reference pictures list and is given the same frame number as the current frame. Typically use of the interpolated reference frame causes a considerable increase in uni-pred prediction, often with no residual to code, and a reduction of bi-prediction modes. This changes the probability of the various supermode values used in Thor. Therefore in such frames it makes sense to modify the supermode coding to reflect this, and this contributes a small amount to coding gains. Full details are in [Fuld1].

#### 5. Compression performance

Luma PSNR BDR percentage gains for standard QP ranges (22,27,32,37) are given in Table 1. For high QP (32,36,40,44), the results are in Table 2.



-----	
1920x1080	
-----	
Kimono	-3.5
ParkScene	-3.1
Cactus	-4.9
BasketballDrive	-2.1
BQTerrace	-1.9
ChangeSeats	-5.8
HeaAndShoulder	-6.6
TelePresence	-6.6
WhiteBoard	-7.5
-----	
1280x720	
-----	
FourPeople	-7.0
Johnny	-6.2
KristenAndSara	-7.0
-----	
Average	-5.2

Table 1: BDR reductions for standard QPs

-----	
1920x1080	
-----	
Kimono	-6.6
ParkScene	-7.0
Cactus	-8.9
BasketballDrive	-5.5
BQTerrace	-4.7
ChangeSeats	-12.1
HeaAndShoulder	-10.1
TelePresence	-11.0
WhiteBoard	-12.4
-----	
1280x720	
-----	
FourPeople	-9.1
Johnny	-8.0
KristenAndSara	-9.9
-----	
Average	-8.8

Table 2: BDR reductions for high QPs

## 6. IANA Considerations

This document has no IANA considerations.

## 7. Security Considerations

This document has no security considerations.

## 8. Acknowledgements

The author would like to thank Arild Fuldseth for assistance with experimental investigations, and Mo Zanaty for reviewing this document.

## 9. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [Fuld1] Fuldseth, A., Bjontegaard, G., Zanaty, M. "The Thor video codec", draft-fuldseth-netvc-thor-01, October 2015.

## Authors' Addresses

Thomas Davies  
Cisco  
Feltham  
UK

Email: thdavies@cisco.com



NETVC Working Group  
Internet-Draft  
Intended status: Informational  
Expires: May 19, 2018

N. Egge  
L. Trudeau  
Mozilla  
D. Barr  
Xiph.Org Foundation  
November 15, 2017

Chroma From Luma Intra Prediction for NETVC  
draft-egge-netvc-cfl-01

Abstract

Chroma from luma (CfL) prediction is a new and promising chroma-only intra predictor that models chroma pixels as a linear function of the coincident reconstructed luma pixels. In this document, we propose the CfL predictor adopted in Alliance Video 1 (AV1) to the NETVC working group. The proposed CfL distinguishes itself from prior art not only by reducing decoder complexity, but also by producing more accurate predictions. On average, CfL reduces the BD-rate, when measured with CIEDE2000, by 5% for still images and 2% for video sequences.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 19, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. State of the Art in Chroma from Luma Prediction . . . . .	4
3. Model Fitting the "AC" Contribution . . . . .	5
4. Chroma "DC" Prediction for "DC" Contribution . . . . .	6
5. Parameter Signaling . . . . .	7
6. Experimental Results . . . . .	8
7. Conclusion . . . . .	10
8. Informative References . . . . .	10
Authors' Addresses . . . . .	12

## 1. Introduction

Still image and video compression is typically not performed using red, green, and blue (RGB) color primaries, but rather with a color space that separates luma from chroma. There are many reasons for this, notably that luma and chroma are less correlated than RGB, which favors compression; and also that the human visual system is less sensitive to chroma allowing one to reduce the resolution in the chromatic planes, a technique known as chroma subsampling [Wang01].

Another way to improve compression in still images and videos is to subtract a predictor from the pixels. When this predictor is derived from previously reconstructed information inside the current frame, it is referred to as an intra prediction tool. In contrast, an inter prediction tool uses information from previously reconstructed frames. For example, "DC" prediction is an intra prediction tool that predicts the pixels values in a block by averaging the values of neighboring pixels adjacent to the above and left borders of the block [Li14].

Chroma from luma (CfL) prediction is a new and promising chroma-only intra predictor that models chroma pixels as a linear function of the coincident reconstructed luma pixels [Kim10]. It was proposed for the HEVC video coding standard [Chen11b], but was ultimately rejected, as the decoder model fitting caused a considerable complexity increase.

More recently, CfL prediction was implemented in the Thor codec [Midtskogen16] as well as in the Daala codec [Egge15]. The

inherent conceptual differences in the Daala codec, when compared to HEVC, led to multiple innovative contributions by Egge and Valin [Egge15] to CfL prediction. Most notably a frequency domain implementation and the absence of decoder model fitting.

As both Thor and Daala are part of NETVC working group, a research initiative was established regarding CfL, the results of which are presented in this draft. The proposed CfL implementation not only builds on the innovations of [Egge15], but does so in a way that is compatible with the more conventional compression tools found in Alliance Video 1 (AV1). The following table details the key differences between LM Mode [Chen11b], Thor CfL [Midtskogen16], and Daala CfL [Egge15] (the previous version of this draft):

	LM Mode	Thor CfL	Daala CfL	AV1 CfL
Prediction Domain	Spatial	Spatial	Frequency	Spatial
Bitsream Signaling	No	No	Sign bit PVQ Gain	Signs + Index
Requires PVQ	No	No	Yes	No
Encoder Model Fitting	Yes	Yes	Via PVQ	Search
Decoder Model Fitting	Yes	Yes	No	No

This new implementation is considerably different from its predecessors. Its key contributions are:

- o Parameter signaling, which avoids model fitting on the decoder and, as explained in Section 2, results in more precise predictions, as the chroma reference pixels are used for fitting (which is impossible when fitting on the decoder). The actual signaling is described in Section 5.
- o Model fitting the "AC" contribution of the reconstructed luma pixels, as shown in Section 3, which simplifies the model and allows for a more precise fit.
- o Chroma "DC" prediction for "DC" contribution, which requires no signaling and, as described in Section 4, is more precise.

Finally, Section 6 presents detailed results of the compression gains of the proposed CfL prediction implementation in AV1.

## 2. State of the Art in Chroma from Luma Prediction

As described in [Kim10], CfL prediction models chroma pixels as a linear function of the coincident reconstructed luma pixels. More precisely, Let  $L$  be an  $M \times N$  matrix of pixels in the luma plane; we define  $C$  to be the chroma pixels spatially coincident to  $L$ . Since  $L$  is not available to the decoder, the reconstructed luma pixels,  $L^r$ , corresponding to  $L$  are used instead. The chroma pixel prediction,  $C^p$ , produced by CfL uses the following linear equation:

$$C^p = \alpha * L^r + \beta$$

Some implementations of CfL [Kim10], [Chen11b] and [Midtskogen16] determine the linear model parameters  $\alpha$  and  $\beta$  using linear least-squares regression

$$\alpha = \frac{\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} L^r(i, j) * C(i, j) - \left( \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} L^r(i, j) \right) \left( \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} C(i, j) \right)}{\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (L^r(i, j))^2 - \left( \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} L^r(i, j) \right)^2}$$

$$\beta = \frac{\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} C(i, j) - \alpha \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} L^r(i, j)}{M * N}$$

We classify [Kim10], [Chen11b], and [Midtskogen16] as implicit implementations of CfL, since  $\alpha$  and  $\beta$  are not signaled in the bitstream, but are implied from the bitstream. The main advantage of the implicit implementation is the absence of signaling.

However, implicit implementations have numerous disadvantages. As mentioned before, computing least squares considerably increases decoder complexity. Another important disadvantage is that the chroma pixels,  $C$ , are not available when computing least squares on the decoder. As such, prediction error increases since neighboring reconstructed chroma pixels must be used instead.

In [Egge15], the authors argue that the advantages of explicit signaling considerably outweigh the signaling cost. Based on these

findings, we propose a hybrid approach that signals alpha and implies beta.

### 3. Model Fitting the "AC" Contribution

In [Egge15], Egge and Valin demonstrate the merits of separating the "DC" and "AC" contributions of the frequency domain CfL prediction. In the pixel domain, the "AC" contribution of a block can be obtained by subtracting it by its average.

An important advantage of the "AC" contribution is that it is zero mean, which results in significant simplifications to the least squares model parameter equations. More precisely, let  $L^{AC}$  be the zero-meanded reconstructed luma pixels. Because

$$\frac{1}{M} \frac{1}{N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} L_{AC}(i, j) = 0$$

substituting  $L^r$  by  $L_{AC}$  yields the following simplified model parameters equations:

$$\alpha_{AC} = \frac{\frac{1}{M} \frac{1}{N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} L_{AC}(i, j) * C(i, j)}{\frac{1}{M} \frac{1}{N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (L^r(i, j))^2}$$

$$\beta_{AC} = \frac{\frac{1}{M} \frac{1}{N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} C(i, j)}{(M*N)}$$

We define the zero-mean chroma prediction,  $C_{AC}$ , like so

$$C_{AC} = \alpha_{AC} * L_{AC} + \beta_{AC}$$

When computing the zero-mean reconstructed pixels, the resulting values are stored using 1/8th precision fixed-point values. This ensures that even with 12-bit integer pixels, the average can be stored in a 16-bit signed integer.



By combining the luma subsampling step with the average subtraction step not only do the equations simplify, but the subsampling divisions and the corresponding rounding error are removed. The equation corresponding to the combination of both steps simplifies to:

$$L\_AC(i, j) = \frac{8}{sy \cdot sx} \left( \sum_{y=0}^{sy-1} \sum_{x=0}^{sx-1} L^r(sy \cdot i + y, sx \cdot j + x) - \sum_{i=0}^{sy-1} \sum_{j=0}^{sx-1} \frac{8}{sy \cdot sx} \left( \sum_{y=0}^{sy-1} \sum_{x=0}^{sx-1} L^r(sy \cdot i + y, sx \cdot j + x) \right) \right) \\ \text{-----} \\ (M \cdot N)$$

Note that this equation uses an integer division.

In the previous equation,  $sx$  and  $sy$  are the subsampling steps for the  $x$  and  $y$  axes, respectively. The proposed CfL only supports 4:2:0, 4:2:2, 4:4:0 and 4:4:4 chroma subsamplings [Wang01], for which:

$$sy \cdot sx \text{ in } \{1, 2, 4\}.$$

Also, because both  $M$  and  $N$  are powers of two,  $M \cdot N$  is also a power of two. It follows that the previous integer divisions can be replaced by bit shift operations.

#### 4. Chroma "DC" Prediction for "DC" Contribution

Switching the linear model to use zero mean reconstructed luma pixels also changes  $\beta\_AC$ , to the extent that it now only depends on  $C$ . More precisely,  $\beta\_AC$  is the average of the chroma pixels.

The chroma pixel average for a given block is not available in the decoder. However, there already exists an intra prediction tool that predicts this average. When applied to the chroma plane, the "DC" prediction predicts the pixel values in a block by averaging the values of neighboring pixels adjacent to the above and left borders of the block [Lil14].

Concretely, the output of the chroma "DC" predictor can be injected inside the proposed CfL implementation as an approximation for  $\beta\_AC$ .

The proposed CfL prediction is expressed as follows:

$$\text{CfL}(\alpha) = \alpha * L\_AC + DC\_PRED.$$

## 5. Parameter Signaling

Signaling the scaling parameters allows encoder-only fitting of the linear model. This reduces decoder complexity and results in a more precise prediction, as the best scaling parameter can be determined based on the reference chroma pixels which are only available to the encoder. The scaling parameters for both chromatic planes are jointly coded using the following scheme.

First, we signal the joint sign of both scaling parameters. A sign is either negative, zero, or positive. In the proposed scheme, signaling (zero, zero) is not permitted as it results in "DC" prediction. It follows that the joint sign requires an eight-value symbol.

As for each scaling parameter, a 16-value symbol is used to represent values ranging from 0 to 2 with a step of 1/8th. The entropy coding details are beyond the scope of this document; however, it is important to note that a 16-value symbol fully utilizes the capabilities of the multi-symbol entropy encoder [Valin16]. Finally, scaling parameters are signaled only if they are non-zero.

Signaling the scaling parameters fundamentally changes their selection. In this context, the least-squares regression used in [Kim10], [Chen11b], and [Midtskogen16] does not yield an RD-optimal solution as it ignores the trade-off between the rate and the distortion of the scaling parameters.

For the proposed CfL prediction, the scaling parameter is determined using the same rate-distortion optimization mechanics as other coding tools and parameters of AV1. Concretely, given a set of scaling parameters  $A$ , the selected scaling parameter is the one that minimizes the trade-off between the rate and the distortion

$$\alpha = \underset{a \in A}{\operatorname{argmin}} \left( D(\text{CfL}(a)) + \lambda R(a) \right).$$

In the previous equation, the distortion,  $D$ , is the sum of the squared error between the reconstructed chroma pixels and the reference chroma pixels. Whereas, the rate,  $R$ , is the number of bits required to encode the scaling parameter and the residual coefficients. Furthermore,  $\lambda$  is the weighing coefficient between rate and distortion used by AV1.

## 6. Experimental Results

To ensure a valid evaluation of coding efficiency gains, our testing methodology conforms to that of [Daedel17]. All simulation parameters and a detailed sequence-by-sequence breakdown for all the results presented in this paper are available online at [AWCY]. Furthermore, the bitstreams generated in these simulations can be retrieved and analyzed online at [Analyzer].

The following tables show the average percent rate difference measured using the Bjontegaard rate difference, also known as BD-rate [Bjontegaard01]. The BD-rate is measured using the following objective metrics: PSNR, PSNR-HVS [Egiazarian2006], SSIM [Wang04], CIEDE2000 [Yang12] and MSSIM [Wang03]. Of all the previous metrics, only the CIEDE2000 considers both luma and chroma planes. It is also important to note that the distance measured by this metric is perceptually uniform [Yang12].

As required in [Daedel17], for individual feature changes in libaom, we use quantizers: 20, 32, 43, and 55. We present results for three test sets: Objective-1-fast [Daedel17], Subset1 [Testset] and Twitch [Testset].

In the following table, we present the results for the Subset1 test set [AWCYSubset1]. This test set contains still images, which are ideal to evaluate the chroma intra prediction gains of CfL when compared to other intra prediction tools in AV1.

	PSNR	PSNR Cb	PSNR Cr	PSNR HVS	SSIM	MS SSIM	CIEDE 2000
Average	-0.53	-12.87	-10.75	-0.31	-0.34	-0.34	-4.87

For still images, when compared to all of the other intra prediction tools of AV1 combined, CfL prediction reduces the rate by an average of 5% for the same level of visual quality measured by CIEDE2000.

For video sequences, next table breaks down the results obtained over the objective-1-fast test set [AWCYObjective1].

	PSNR	PSNR Cb	PSNR Cr	PSNR HVS	SSIM	MS SSIM	CIEDE 2000
Average	-0.43	-5.85	-5.51	-0.42	-0.38	-0.40	-2.41
1080p	-0.32	-6.80	-5.31	-0.37	-0.28	-0.31	-2.52
1080psc	-1.82	-17.76	-12.00	-1.72	-1.71	-1.75	-8.22
360p	-0.15	-2.17	-6.45	-0.05	-0.10	-0.04	-0.80
720p	-0.12	-1.08	-1.23	-0.11	-0.07	-0.12	-0.52

Not only does CfL yield better intra frames, which produces a better reference for inter prediction tools, but it also improves chroma intra prediction in inter frames. We observed CfL predictions in inter frames when the predicted content was not available in the reference frames. As such, CfL prediction reduces the rate of video sequences by an average of 2% for the same level of visual quality when measured with CIEDE2000.

The average rate reductions for 1080psc are considerably higher than those of other types of content. This indicates that CfL prediction considerably outperforms other AV1 predictors for screen content coding. As shown in the following table, the results on the Twitch test set [AWCYTwitch], which contains only gaming-based screen content, corroborates this finding.

	PSNR	PSNR Cb	PSNR Cr	PSNR HVS	SSIM	MS SSIM	CIEDE 2000
Average	-1.01	-15.58	-9.96	-0.93	-0.90	-0.81	-5.74

Furthermore, individual sequences in the Twitch test set show considerable gains. We present the results for Minecraft\_10\_120f (Mine), GTAV\_0\_120F (GTAV), and Starcraft\_10\_120f (Star) in the following table. It would appear that CfL prediction is particularly efficient for sequences of the game Minecraft both sequences reduces the average rate by 20% for the same level of visual quality measured by CIEDE2000.

	PSNR	PSNR Cb	PSNR Cr	PSNR HVS	SSIM	MS SSIM	CIEDE 2000
Mine	-3.76	-31.44	-25.54	-3.13	-3.68	-3.28	-20.69
GTAV	-1.11	-15.39	-5.57	-1.11	-1.01	-1.04	-5.88
Star	-1.41	-6.18	-6.21	-1.43	-1.38	-1.43	-4.15

## 7. Conclusion

In this document, we presented the chroma from luma prediction tool adopted in AV1 that we proposed for NETVC. This new implementation is considerably different from its predecessors. Its key contributions are: parameter signaling, model fitting the "AC" contribution of the reconstructed luma pixels, and chroma "DC" prediction for "DC" contribution. Not only do these contributions reduce decoder complexity, but they also reduce prediction error; resulting in a 5% average reduction in BD-rate, when measured with CIEDE2000, for still images, and 2% for video sequences.

Possible improvements to Cfl for AV2 include non-linear prediction models and motion-compensated Cfl.

## 8. Informative References

### [Analyzer]

Bebenita, M., "AV1 Bitstream Analyzer",  
Mozilla <https://arewecompressedyet.com/analyzer/>, n.d..

### [AWCY]

"Are We Compressed Yet?", Xiph.Org  
Foundation <https://arewecompressedyet.com/>, n.d..

### [AWCYObjective1]

Trudeau, L., "Results of Chroma from Luma over the  
Objective-1-fast test set", Are We Compressed  
Yet? <https://doi.org/10.6084/m9.figshare.5577778.v1>,  
November 2017.

### [AWCYSubset1]

Trudeau, L., "Results of Chroma from Luma over the Subset1  
test set", Are We Compressed  
Yet? <https://doi.org/10.6084/m9.figshare.5577661.v2>,  
November 2017.

- [AWCYTwitch] Trudeau, L., "Results of Chroma from Luma over the twitch test set", Are We Compressed Yet? <https://doi.org/10.6084/m9.figshare.5577946.v1>, November 2017.
- [Bjontegaard01] Bjontegaard, G., "Calculation of average PSNR differences between RD-curves", Video Coding Experts Group (VCEG) of ITU-T VCEG-M33, 2001.
- [Chen11b] Chen, J., Seregin, V., Han, W., Kim, J., and B. Jeon, "CE6.a.4: Chroma intra prediction by reconstructed luma samples", Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, JCTVC-E266, March 2011.
- [Daede17] Daede, T., Norkin, A., and I. Brailovsky, "Video Codec Testing and Quality Measurement", IETF NETVC Internet-Draft draft-ietf-netvc-testing-05, March 2017.
- [Egge15] Egge, N. and J. Valin, "Predicting chroma from luma with frequency domain intra prediction", Proceedings of SPIE 9410, Visual Information Processing and Communication VI, March 2015.
- [Egiazarian2006] Egiazarian, K., Astola, J., Ponomarenko, N., Lukin, V., Battisti, F., and M. Carli, "Two new full-reference quality metrics based on HVS", Proceedings of the Second International Workshop on Video Processing and Quality Metrics for Consumer Electronics VPQM, January 2006.
- [Kim10] Kim, J., Park, S., Choi, Y., Jeon, Y., and B. Jeon, "New intra chroma prediction using inter-channel correlation", Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, JCTVC-B021, January 2010.
- [Li14] Ze-Nian, L., Drew, M., and J. Liu, "Fundamentals of Multimedia", ISBN 3319052896, org Springer Publishing Company, Incorporated, edition 2nd, 2014.
- [Midtskogen16] Midtskogen, S., "Improved chroma prediction", draft-midtskogen-netvc-chromapred-02 IETF NETVC Internet-Draft, October 2016.

- [Testset] Daede, T., "Test Sets", Hosted by the Xiph.org Foundation <https://people.xiph.org/~tdaede/sets/>, n.d..
- [Valin16] Valin, J., Terriberry, T., Egge, N., Daede, T., Cho, Y., Montgomery, C., and M. Bebenita, "Daala: Building A Next-Generation Video Codec From Unconventional Technology", Multimedia signal processing (MMSP) workshop arXiv:1608.01947, September 2016.
- [Wang01] Wang, Y., Zhang, Y., and J. Ostermann, "Video Processing and Communications", ISBN 23132985, Prentice Hall PTR, Upper Saddle River, NJ, USA, edition 1st, 2001.
- [Wang03] Wang, Z., Simoncelli, E., and A. Bovik, "Multiscale structural similarity for image quality assessment", The 37th Asilomar Conference on Signals, Systems Computers Volume 2, November 2003.
- [Wang04] Wang, Z., Bovik, A., Sheikh, H., and E. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity", issn 1057-7149, IEEE transactions on image processing Volume 13, number 4, April 2004.
- [Yang12] Yang, Y., Ming, J., and N. Yu, "Color Image Quality Assessment Based on CIEDE2000", Advances in multimedia Article ID 273723, 2012.

## Authors' Addresses

Nathan E. Egge  
Mozilla  
331 E Evelyn Ave  
Mountain View 94041  
USA  
  
Email: [negge@mozilla.com](mailto:negge@mozilla.com)

Luc N. Trudeau  
Mozilla  
331 E Evelyn Ave  
Mountain View 94041  
USA  
  
Email: [luc@trud.ca](mailto:luc@trud.ca)

David M. Barr  
Xiph.Org Foundation  
21 College Hill Road  
Somerville, MA 1124  
USA

Email: b@rr-dav.id.au



Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: January 7, 2016

N. Egge  
T. Terriberry  
Mozilla Corporation  
July 6, 2015

Time Domain Lapped Transforms for Video Coding  
draft-egge-netvc-tdlt-00

Abstract

This proposes the use of Time Domain Lapped Transforms (TDLT) as the transform step for video coding.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 7, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may not be modified, and derivative works of it may not be created, and it may not be published except as an Internet-Draft.

## Table of Contents

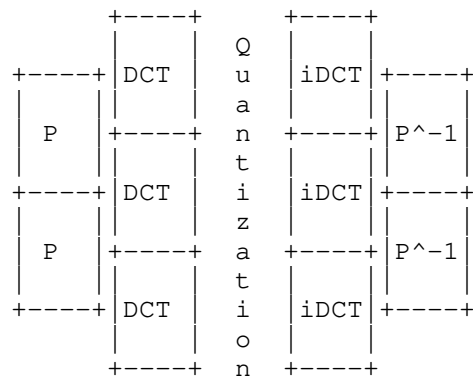
1. Introduction . . . . .	2
2. TDLT Defined . . . . .	2
3. Lapped-Transform Selection . . . . .	4
3.1. Coding Gain . . . . .	5
3.2. Transform Width . . . . .	6
4. Optimal Transform Coefficients . . . . .	6
4.1. Exhaustive Search . . . . .	6
4.2. Stochastic Search . . . . .	7
4.3. Ramp Constraint . . . . .	8
5. Intra Prediction . . . . .	10
6. Motion Compensation . . . . .	11
7. Multiple Block Sizes . . . . .	11
7.1. Variable Sized Lapping . . . . .	12
7.2. Fixed Sized Lapping . . . . .	15
8. IANA Considerations . . . . .	16
9. Security Considerations . . . . .	16
10. Acknowledgments . . . . .	16
11. Informative References . . . . .	16
Authors' Addresses . . . . .	17

## 1. Introduction

This draft outlines a proposal to adapt the Time-Domain Lapped Transforms (TDLT) for use in video coding. Lapped transforms were proposed for video coding at least as far back as 1989 [Malv89]. Like the loop filters more commonly found in recent video coding standards, TDLTs use a post-processing filter that runs between block edges to reduce or eliminate blocking artifacts. Unlike a loop filter, the TDLT filter is invertible, allowing the encoder to run the inverse filter on the input video. This decorrelates blocks before they are passed through a normal block transform and quantization step, improving coding gain (which helps in both smooth and highly textured areas), in addition to reducing blocking artifacts.

## 2. TDLT Defined

The Time-Domain Lapped Transform can be viewed as a set of pre and post filters to an existing block-based DCT transform. The idea is to place an invertible filter along the block boundaries outside an existing block-based DCT encoder.



The pre-filter P operates in the time domain, processing block boundaries and removing inter-block correlation. The blocks are then transformed by the DCT into the frequency domain, where the resulting coefficients are quantized and encoded. When decoding, the inverse operator  $P^{-1}$  is applied as a post-filter to the output of the inverse DCT. This has two benefits:

1. Quantization errors are spread over adjacent blocks via the post-filter  $P^{-1}$ , reducing blocking artifacts. This eliminates the need for a separate deblocking filter.
2. The increased support region of the transform allows it to take advantage of inter-block correlation to achieve a higher coding gain than a non-overlapped DCT. This allows it to more effectively code both smooth and textured regions.

The pre-filter  $P$  is defined in [Tran01] as follows:

$$P = \frac{1}{2} \begin{bmatrix} I & J \\ J & -I \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & V \end{bmatrix} \begin{bmatrix} I & J \\ J & -I \end{bmatrix}$$

Here  $I$  is the identity matrix and  $J$  is the "reversal matrix", obtained by simply re-ordering the rows of the identity matrix in reverse order. The  $V$  matrix is a free parameter, and as long as  $V$  is invertible, this filter structure guarantees perfect reconstruction, linear phase, and biorthogonality. If  $V$  is orthogonal, then the overall transform is also orthogonal instead of just biorthogonal.

For the case of the 4x8 TDLT, we use the following invertible matrix for V:

$$V = \begin{bmatrix} 1 & q_0 \\ & \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ & \\ p_0 & 1 \end{bmatrix} \begin{bmatrix} s_0 & 0 \\ & \\ 0 & s_1 \end{bmatrix}$$

Thus for the 4x8 case, the pre-filter and post-filter are completely described by the four parameters  $q_0$ ,  $p_0$ ,  $s_0$ , and  $s_1$ . In general, any invertible  $V$  matrix may be used. However, factoring  $V$  into a series of lifting steps ensures that it can be implemented efficiently, and can reduce the number of parameters required by the optimization process, since the full flexibility of an arbitrary invertible matrix is not required to achieve good coding gain.

[Tran01] proposes two reduced-parameter factorizations, dubbed Type III and Type IV. These are identical in the 4x8 case, but for larger transforms the differ in the order that the  $p_i$  and  $q_i$  steps are applied: interleaved for Type III and ascending and then descending order for Type IV. While Type III appears to give slightly higher coding gain when unconstrained, when coupled with the ramp constraint discussed below and the constraint that all coefficients be dyadic rationals, the number of feasible solutions is much smaller than with Type IV. The increased number of feasible solutions allows Type IV transforms to achieve higher coding gains than Type III when these constraints are imposed. This definition easily extends to the 8x16 and 16x32 TDLT case with similar parameterizations. In general, we use the Type IV factorization from [Tran01]. For a  $V$  matrix of size  $M$ , this has  $(M-1)$   $p_i$  and  $(M-1)$   $q_i$  parameters, and  $M$   $s_i$  parameters. For a transform of size  $N \times 2N$ , this gives a total of  $1.5N-2$  parameters. This is also the number of lifting steps that must be performed to implement the  $V$  portion of the pre- and post-filters.

### 3. Lapped-Transform Selection

We would like to find good candidate transform coefficients that perform well within a video coding framework. There are several metrics we can use for evaluating pre-filter parameters. Including

1. Coding Gain - how well energy is compacted into only a few coefficients
2. Side Band Attenuation - how much energy from frequencies outside the passband leaks into each basis function
3. Transform Width - how wide are the basis functions and how much ringing they will cause
4. Orthogonality - how linearly independent the basis functions are

Of these, the most important by far is coding gain as it allows us to directly measure the improvement in bits between different candidate transforms. At high bit rates using an efficient quantizer, every 6.02 dB improvement in coding gain saves a bit of entropy per coefficient.

### 3.1. Coding Gain

Coding gain is a useful metric for comparing different candidate transforms. Roughly speaking, it is the measure of how well energy is compacted into only a few coefficients. The formula for coding gain of the lapped transform can be found in [Terr12]. Using an AR(1) model with  $r=0.95$ , we have

$$C_g = 10 \cdot \text{Log}_{10} \left| \frac{1}{\prod_i ((G \cdot \text{AR}(1) \cdot G^T)[i,i] \cdot (H^T \cdot H)[i,i])} \right|$$

where  $G$  is the analysis filter of the lapped transform:

$$G = \begin{bmatrix} & & & \\ & & & \\ 0 & \text{DCT} & 0 & \\ & & & \end{bmatrix} \begin{bmatrix} P & 0 \\ & \\ & \\ 0 & P \end{bmatrix}$$

and  $H$  is the synthesis filter of the lapped transform:

$$H = \begin{bmatrix} & & & \\ & & & \\ & & & \\ 0 & P^{-1} & & \end{bmatrix} \begin{bmatrix} 0 \\ \text{idCT} \\ & \\ 0 \end{bmatrix}$$

In [Terr12] the coding gain of the non-lapped DCT is compared with the optimal non-lapped Karhunen-Loeve transform for the same AR(1) model with  $r=0.95$ .

	4 point	8 point	16 point
DCT	7.5701 dB	8.8259 dB	9.4555 dB
KLT	7.5825 dB	8.8462 dB	9.4781 dB

Similarly, in [Tran01] the coding gain of the TDLT using fast factorizations with real coefficients produced by unconstrained optimization are

	4x8	8x16	16x32
Type III TDLT	8.6349 dB	9.6115 dB	9.9496 dB
Type IV TDLT	8.6349 dB	9.6005 dB	9.9057 dB

### 3.2. Transform Width

In general, the wider the transform, the higher the coding gain: a 16-point DCT will always have a higher coding gain than a 4-point DCT. In the case of lapped transform, the width of the transform is more than just counting the number of points, it involves the shape of the basis functions. At equal coding gain, a narrower transform is better because it causes a smaller amount of ringing around edges. We define the width of the transform as

$$w = C * \left| \frac{\sum_{ij} (H[i,j]^2 * (j-N+1/2)^4)}{\sum_{ij} (H[i,j]^2)} \right|^{1/4},$$

where  $C=2.991$  is a constant calibrated such that the width of the 1024-point non-overlapped DCT is equal to 1024.

## 4. Optimal Transform Coefficients

Of the four metrics described in Section 3 we chose to optimize our transform parameters for the highest coding gain.

To avoid the use of floating point operations, we use dyadic rationals to represent the parameters of our TDLT. These are the  $p$ 's,  $q$ 's and  $s$ 's that describe the  $V$  matrix in the pre-filter. We chose a base of  $2^6$  because it offered enough resolution to find good approximations of the optimal values for the  $p$ 's,  $q$ 's, and  $s$ 's and still allowed us to fit the results of multiplications in a 16 bit word. Increasing the base to  $2^8$  improves the achievable coding gain of the 4x8 transform by less than 0.002 dB. On the other hand, dropping it even one bit to  $2^5$  lowers the coding gain by 0.037 dB.

### 4.1. Exhaustive Search

For the smaller lapped transforms, it is possible to simply do an exhaustive search and check all possible transform candidates to find the one with the best coding gain. The limitation that the  $p$ 's,  $q$ 's, and  $s$ 's all be dyadic rationals allows us to simply enumerate all reasonable values. Additional constraints allowed us to further reduce the search space. Because the  $p$ 's and  $q$ 's are liftings steps that represent rotations in the plane their values are between -1.0

and 1.0. Likewise the limitation that the pre- and post-filter steps be reversible requires that the scale factors be greater than or equal 1.0, otherwise information would be lost during the transform. Finally, all things equal we prefer smaller scale factors as it makes quantizing and encoding the coefficients cheaper. We thus cap the scale factors at 2.0. Based on some limited experimentation, scale factors larger than this do not appear to produce useful transforms according to our metrics, anyway.

With a dyadic rational base of  $2^6$ , the number of possible candidates to consider is

$$\begin{aligned} |C| &= (2*(2^6)+1)^{(|p|+|q|)} * (2^6+1)^{|s|} \\ &= (2*(2^6)+1)^{(2*(N/2-1))} * (2^6+1)^{(N/2)} \end{aligned}$$

Thus for the transform sizes we are interested in, the number of candidates is tractable only for the 4x8 case:

	N	C
4x8 TDLT	4	68161536
8x16 TDLT	8	$7.731400 * 10^{19}$
16x32 TDLT	16	$9.947082 * 10^{43}$

An exhaustive search for parameters that give the optimal coding gain for the 4x8 TDLT are below:

$p_0$	-11/64	$q_0$	36/64	$s_0$	91/64
				$s_1$	85/64

#### 4.2. Stochastic Search

For the larger lapped transforms, doing an exhaustive search is not possible. Instead we formulate the optimization problem as an integer programming problem and use a robust industrial solver to find optimal integer values for the p's, q's, and s's.

For the 8x16 TDLT, the parameters are below:

$p_0$	-23/64	$q_0$	48/64	$s_0$	90/64
$p_1$	-18/64	$q_1$	34/64	$s_1$	73/64
$p_2$	-6/64	$q_2$	20/64	$s_2$	72/64
				$s_3$	75/64

For the 16x32 TDLT, the parameters are below:

p_0	-24/64	q_0	50/64	s_0	90/64
p_1	-23/64	q_1	40/64	s_1	74/64
p_2	-17/64	q_2	31/64	s_2	73/64
p_3	-12/64	q_3	22/64	s_3	71/64
p_4	-14/64	q_4	18/64	s_4	67/64
p_5	-13/64	q_5	16/64	s_5	67/64
p_6	-7/64	q_6	11/64	s_6	67/64
				s_7	72/64

In order to confirm that the integer approximations found are in fact optimal, we can compare them with the optimal real valued coding gains for the three lapped-transforms we are proposing. In [Tran01] a numeric solver was used to find optimal values for a Type IV lapped transform.

	4x8	8x16	16x32
Real Valued	8.6349 dB	9.6005 dB	9.9057 dB
Approximate	8.63473 dB	9.60021 dB	9.89338 dB
Loss	0.00017 dB	0.00029 dB	0.01232 dB

#### 4.3. Ramp Constraint

It is also possible to constrain the lapped transform so that it is (1,2)-regular [DT03], i.e., that it has one vanishing moment in the analysis filter and two vanishing moments in the synthesis filter. This allows the synthesis filter to reconstruct any piecewise linear function solely from the DC coefficients. This causes the shape of the DC basis function to be a symmetric linear ramp. This can be particularly useful when it matches the shape of other windowing functions used in the codec. For example, a linear window is commonly used with Overlapped Block Motion Compensation (OBMC), which is one possible approach for avoiding blocking artifacts in the motion-compensation stage of the codec. More vanishing moments are possible, allowing reconstruction of piecewise quadratic or even higher-order functions, but these require additional overlap stages.

This regularity can be enforced solely by enforcing a series of constraints on the scale factors,  $s_i$ .



$$s_0 = N * (1 - q_0)$$

$$s_i = \frac{N}{2^i + 1} * \left\lfloor \frac{(q_{i-1} - 1) * p_{i-1} - q_i}{1} \right\rfloor, \text{ for } i > 0$$

Since  $2^i + 1$  is odd, but we want  $s_i$  to be a dyadic rational value, the remainder of the expression must be evenly divisible by  $(2^i + 1)$ . A similar set of constraints can be derived for Type III, but they involve more of the  $p$ 's and  $q$ 's per  $s_i$  value, and thus have far fewer admissible solutions when coupled with the dyadic rational constraint.

The additional restrictions described above greatly reduce the number of combinations to consider, both because there are fewer parameters (the  $s_i$ 's can no longer be chosen independently) and because there are fewer combinations of parameter values which produce dyadic rational coefficients. With these constraints, the number of combinations is small enough that an exhaustive search is now tractable for the 8x16 TDLT.

	N	C
4x8 TDLT	4	442
8x16 TDLT	8	331677320

An exhaustive search for parameters that give the optimal coding gain under the ramp and dyadic rational constraints for the 4x8 and 8x16 TDLT are below:

<table><tr><td>p_0</td><td>-16/64</td></tr></table>	p_0	-16/64	<table><tr><td>q_0</td><td>41/64</td></tr></table>	q_0	41/64	<table><tr><td>s_0</td><td>92/64</td></tr><tr><td>s_1</td><td>93/64</td></tr></table>	s_0	92/64	s_1	93/64												
p_0	-16/64																					
q_0	41/64																					
s_0	92/64																					
s_1	93/64																					
<table><tr><td>p_0</td><td>-24/64</td></tr><tr><td>p_1</td><td>-20/64</td></tr><tr><td>p_2</td><td>-4/64</td></tr></table>	p_0	-24/64	p_1	-20/64	p_2	-4/64	<table><tr><td>q_0</td><td>53/64</td></tr><tr><td>q_1</td><td>40/64</td></tr><tr><td>q_2</td><td>24/64</td></tr></table>	q_0	53/64	q_1	40/64	q_2	24/64	<table><tr><td>s_0</td><td>88/64</td></tr><tr><td>s_1</td><td>75/64</td></tr><tr><td>s_2</td><td>76/64</td></tr><tr><td>s_3</td><td>76/64</td></tr></table>	s_0	88/64	s_1	75/64	s_2	76/64	s_3	76/64
p_0	-24/64																					
p_1	-20/64																					
p_2	-4/64																					
q_0	53/64																					
q_1	40/64																					
q_2	24/64																					
s_0	88/64																					
s_1	75/64																					
s_2	76/64																					
s_3	76/64																					

Unfortunately, in the 16x32 TDLT case the number of combinations is still not tractable, even with these additional constraints. Again, we use an integer programming model to solve for the integer parameters that optimize coding gain in this context.

p_0	-32/64	q_0	59/64	s_0	80/64
p_1	-28/64	q_1	53/64	s_1	72/64
p_2	-24/64	q_2	46/64	s_2	73/64
p_3	-32/64	q_3	41/64	s_3	68/64
p_4	-24/64	q_4	35/64	s_4	72/64
p_5	-13/64	q_5	24/64	s_5	74/64
p_6	-2/64	q_6	12/64	s_6	74/64
				s_7	70/64

	4x8	8x16	16x32
Dyadic	8.63473 dB	9.60021 dB	9.89338 dB
Ramp + Dyadic	8.59886 dB	9.56161 dB	9.78294 dB
Loss	0.03587 dB	0.0386 dB	0.11044 dB

## 5. Intra Prediction

Since the final pixel values of a block are not available until after the post-filter runs, they cannot be used to predict neighboring blocks. There are a number of possible solutions to this. For example, one could simply use pixels from outside the overlap region. However, as these pixels are farther away, they are poorer predictors, and the extra distance reduces the range of prediction directions which have enough neighbors available to form an adequate extrapolation [OP11].

An alternate approach is to perform the prediction in the frequency domain. Initial experiments suggest that this is just as effective as prediction in the time domain, and has similar computational requirements [Egge13]. However, because the frequency domain coefficients of a neighboring block are impacted both by what size DCT was used, and the lapping across all four of its edges, directional predictors can only be so good. At low rates, this meant more bits were spent correcting an incorrect predictor than were saved by coding only a directional mode.

A signal free technique was developed for doing limited intra prediction in the frequency domain when using lapped transforms. Note that when the spatial prediction mode is exactly horizontal or vertical, applying the filters described in this draft along the orthogonal direction is the identity. Thus it is possible to look at the horizontal coefficients of the neighboring block to the left, and the vertical energy of the neighboring block above and simply use the coefficients where the energy is larger. When this technique is

coupled with a quantization and coefficient coder that makes signaling no predictor cheap [Vali15], this becomes an effective frequency domain intra predictor.

Finally, a technique was developed for intra predicting chroma frequency domain coefficients from decoded coincident luma coefficients [Egge15]. While this technique does not strictly require the use of lapped transforms, because the block size extent (and thus the lapping region) for both the chroma and luma planes is the same, the use of a lapped transform does not change the effectiveness of this technique.

## 6. Motion Compensation

There have been several lapped transform proposals that perform block-by-block motion compensation by simply expanding the size of the prediction region for each block [TT01], [OPT11]. However, in addition to increasing the amount of motion-compensated prediction pixels that must be computed by a factor of four, this also increases the number of applications of the pre- and post-filter by a factor of four, since this must now be done separately for each block, using the motion-compensated frame difference for that block.

An alternate approach is simply perform motion compensation of the frame in a completely separate step, prior to any transform, using any method desired [Terr15]. The lapping can then be applied to this motion-compensated prediction, producing per-block predictors. This still allows the prediction mode (inter, intra, bi-prediction, etc.) to be chosen on a block-by-block basis. It also interacts well with other techniques designed to operate in the frequency domain, such as the Pyramid Vector Quantization (PVQ) proposed elsewhere.

The downside is that motion estimation in the encoder needs to be performed for regions slightly beyond the current block. However, this is already required by blocking-artifact-free motion compensation techniques, such as Overlapped Block Motion Compensation (OBMC). Experience with OBMC has shown that an encoder can mostly ignore look-ahead and still get acceptable results, unlike other techniques, such as control-grid interpolation (CGI).

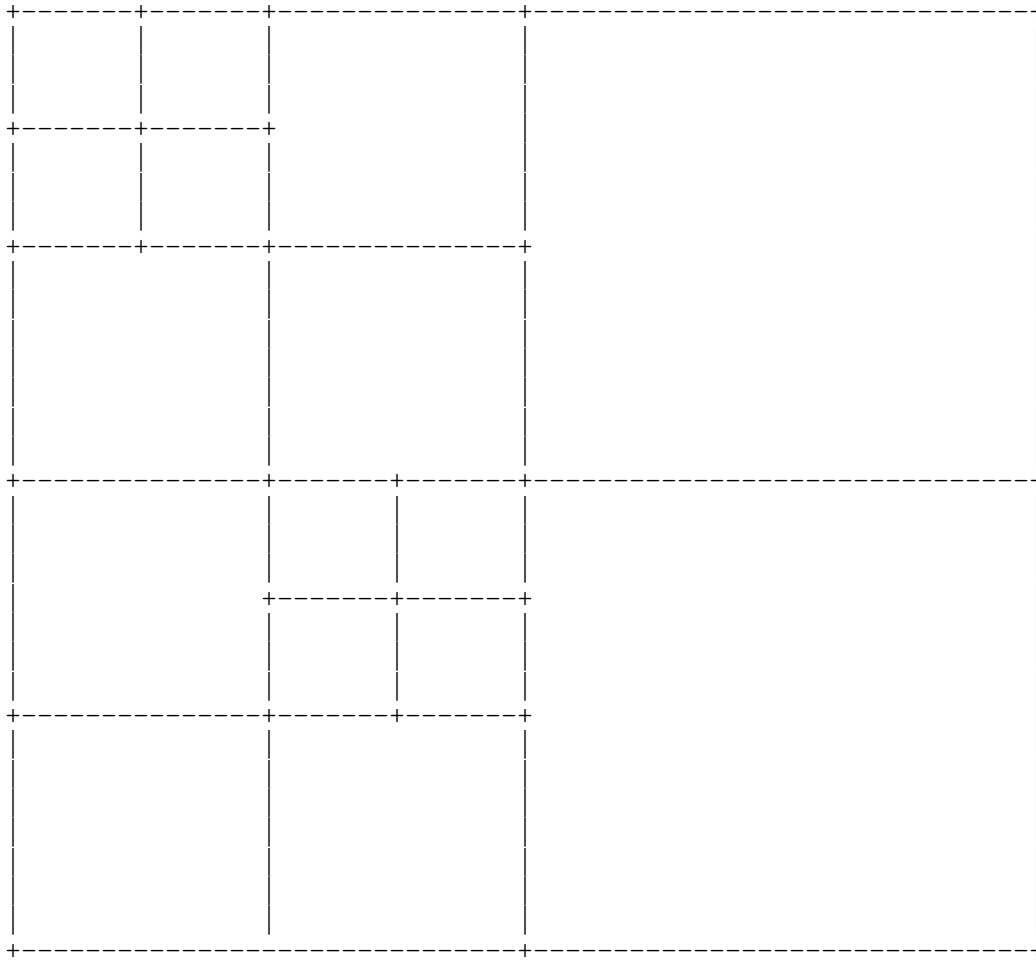
## 7. Multiple Block Sizes

Multiple block size support is important for lapped transforms, since the larger support region increases their susceptibility to ringing artifacts compared to a non-overlapped transform with the same number of coefficients (though it is greatly reduced compared to a non-overlapped transform with a support region of the same size).

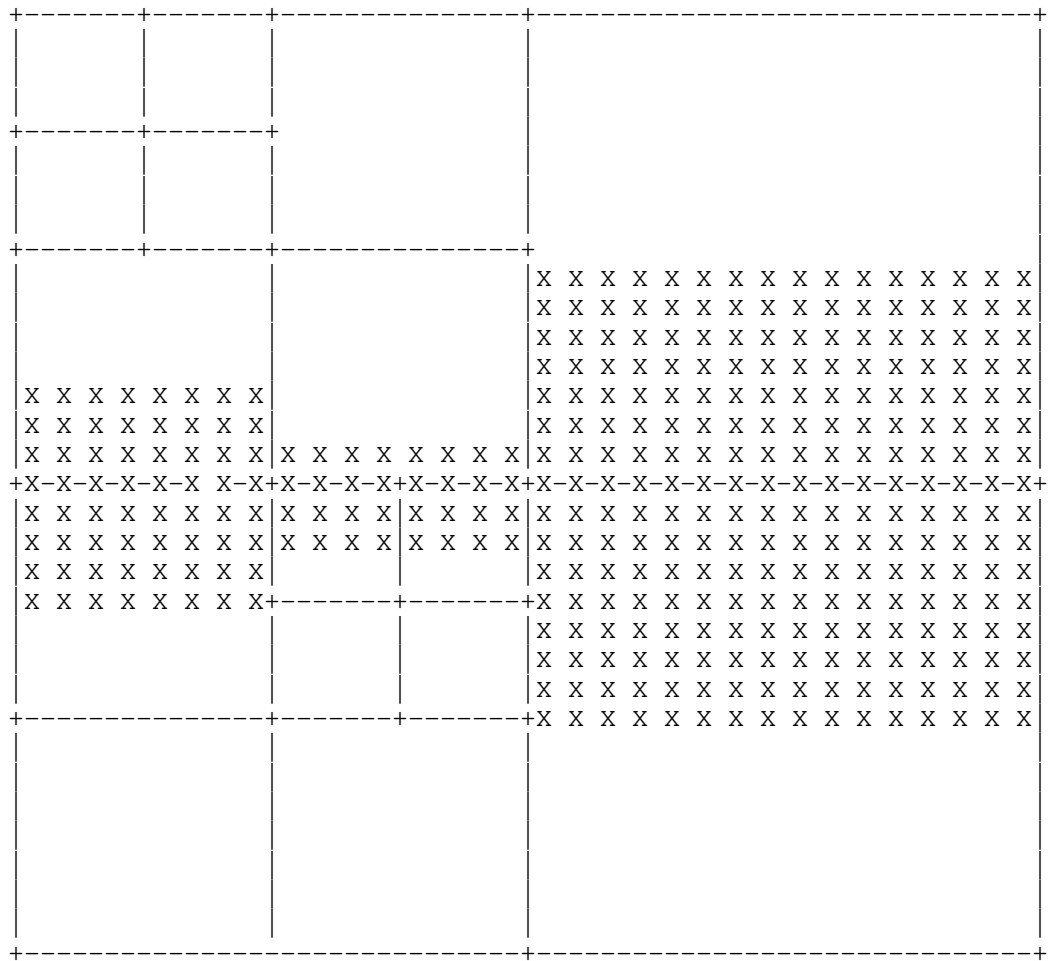
### 7.1. Variable Sized Lapping

The most obvious approach is to require that the size of the overlap filter be constrained by the smallest block adjacent to a given edge. This requires some amount of look-ahead in the encoder, but has the benefit of using the largest lapping possible in regions where all blocks are the same size while not introducing discontinuities where blocks of different sizes meet. Note that this has an effect on the coding syntax, as the block size decision for the block below the one being coded must be made and communicated to the decoder prior to coding. Using this convention no additional information need to be communicated other than the block size decision to completely describe how the variable sized lapping should be applied.

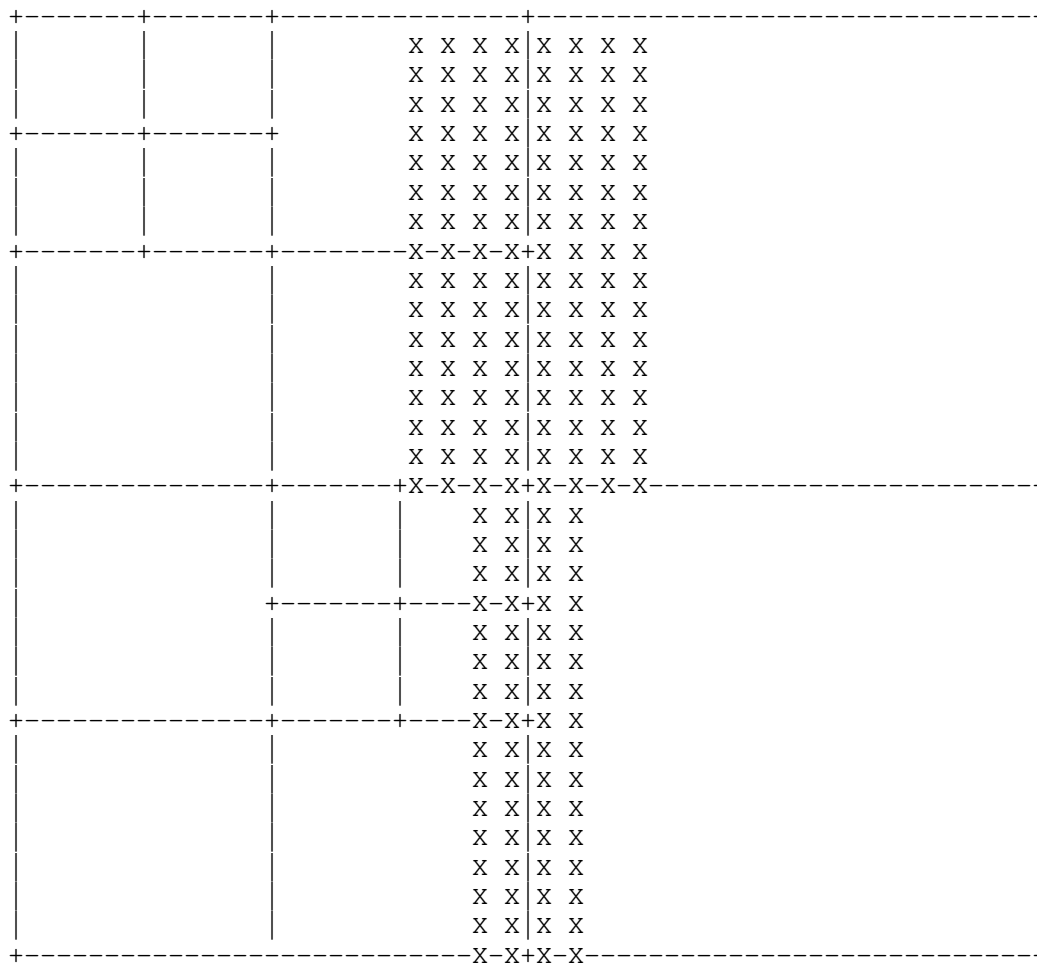
Consider an example image that is 32x32 with the following block size decisions. We apply the lapping recursively to blocks of 32x32 at a time until we reach a block that is not subdivided into smaller blocks. At each step in the recursions, we apply a filter vertically across the block edges that run left to right splitting the block in half. We then apply a horizontal filter across the block edges that split the block in half top to bottom.



Block size decision for a 32x32 frame.



Apply the filter vertically across the horizontal internal edge.



Apply the filter horizontally across the vertical internal edge.

The filters are then applied recursively in this manner to the four quadrants of the block. By applying the filters recursively this way, we have prevented any discontinuities from appearing where block is split but its neighbor is not.

## 7.2. Fixed Sized Lapping

One of the challenges using variable sized lapping is that changing the block size decision (either splitting a block into four blocks a quarter as big, or merging four blocks into one four times the size) can have an impact on the coding performance outside the block considered. This makes computing the optimal block size decision for

a frame computationally difficult as traditional rate-distortion optimization (RDO) algorithms exploit this locality to iteratively improve an initial decision.

One way to simplify the problem is to assume a fixed sized lapping across the entire image. If only the 4-point filter is used across block boundaries, then it is possible to compare the distortion of an 8x8 block with that of four 4x4 blocks by simply computing the mean squared error (MSE) of the 64 spatial domain coefficients after applying the inverse lapped transform. Because changing the block size decision, and thus the interior lapping has no impact on the lap decision on the border of the 8x8 block, then just looking at the rate and distortion of the interior coefficients is sufficient.

This approach has does not leverage the additional coding gain and deblocking achieved by using larger lapping filters but may make up for this by allowing computationally cheap block size decision heuristics in real-time encoding environments.

#### 8. IANA Considerations

This document has no actions for IANA.

#### 9. Security Considerations

This draft has no security considerations.

#### 10. Acknowledgments

Thanks to Greg Maxwell and Jean-Marc Valin for their assistance in the experimentation and other valuable contributions to this document.

#### 11. Informative References

- [DT03] Dai, W. and T. Tran, "Regularity-Constrained Pre- and Post-Filtering for Block DCT Based Systems", IEEE Transactions on Signal Processing 51(10):2568--2581, October 2003.
- [Egge13] Egge, N., "Intra-Prediction in Daala", October 2013, <<http://people.xiph.org/~unlord/Daala-Intra.pdf>>.
- [Egge15] Egge, N. and J. Valin, "Predicting Chroma from Luma with Frequency Domain Intra Prediction", February 2015, <[http://people.xiph.org/~unlord/spie\\_cfl.pdf](http://people.xiph.org/~unlord/spie_cfl.pdf)>.



- [Malv89] Malvar, H. and D. Staelin, "The LOT: Transform Coding Without Blocking Effects", IEEE Transactions on Acoustics, Speech, and Signal Processing , April 1989, <<http://research.microsoft.com/apps/pubs/default.aspx?id=102073>>.
- [OP11] de Oliveria, R. and B. Pesquet-Popescu, "Intra-Frame Prediction with Lapped Transforms for Image Coding", Proc. of the 36th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'11) pp. 805--808, May 2011.
- [OPT11] de Oliveria, R., Pesquet-Popescu, B., and M. Trocan, "Inter Prediction Using Lapped Transforms for Advanced Video Coding", Proc. of the 18th IEEE International Conference on Image Processing (ICIP'11) pp. 3705--3708, September 2011.
- [Tran01] Tran, T., "Lapped Transform via Time-Domain Pre- and Post-Processing", IEEE Transactions on Signal Processing , October 2001.
- [TT01] Tran, T. and C. Tu, "Lapped Transform Based Video Coding", Proc. of the 24th SPIE Conference on Applications of Digital Image Processing vol. 4472, pp. 319--333, July 2001.
- [Terr12] Terriberry, T., "Introduction to Video Coding Part 1: Transform Coding", February 2012.
- [Terr15] Terriberry, T., "Adaptive Motion Compensation Without Blocking Artifacts", February 2015, <<https://people.xiph.org/~tterribe/daala/vbsobmc.pdf>>.
- [Vali15] Valin, J., "Perceptual Vector Quantization for Video Coding", February 2015, <[http://jmvalin.ca/video/spie\\_pvq.pdf](http://jmvalin.ca/video/spie_pvq.pdf)>.

## Authors' Addresses

Nathan E. Egge  
Mozilla Corporation  
331 E. Evelyn Avenue  
Mountain View, CA 94041  
USA

Phone: +1 650 903-0800  
Email: [negge@xiph.org](mailto:negge@xiph.org)

Timothy B. Terriberry  
Mozilla Corporation  
331 E. Evelyn Avenue  
Mountain View, CA 94041  
USA

Phone: +1 650 903-0800  
Email: tterribe@xiph.org

Network Working Group  
Internet Draft  
Intended status: Informational  
Expires: April 19, 2016

A. Filippov  
Huawei Technologies  
October 19, 2015

<Video Codec Requirements and Evaluation Methodology>  
draft-filippov-netvc-requirements-02.txt

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts can be accessed at <http://datatracker.ietf.org/drafts/current/>

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>

This Internet-Draft will expire on April 19, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in

Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Abstract

This document provides requirements for a video codec designed mainly for use over the Internet. In addition, an evaluation methodology needed for measuring the parameters (compression efficiency, computational complexity, etc.) to ensure whether the stated requirements are fulfilled or not.

## Table of Contents

1. Introduction.....	3
2. Applications.....	3
2.1. Internet Protocol Television (IPTV) / IP-based over-the-top (OTT) video transmission.....	4
2.2. Video conferencing.....	5
2.3. Video sharing.....	6
2.4. Screencasting.....	7
2.5. Game streaming.....	8
2.6. Video monitoring / surveillance.....	8
3. Requirements.....	9
3.1. Basic requirements.....	10
3.1.1. Input source formats:.....	10
3.1.2. Coding delay:.....	11
3.1.3. Complexity:.....	11
3.1.4. Scalability:.....	11
3.1.5. Error resilience:.....	11
3.2. Optional requirements.....	11
3.2.1. Input source formats.....	11
3.2.2. Scalability:.....	11
3.2.3. Complexity:.....	12
4. Evaluation methodology.....	12
4.1. Compression performance evaluation.....	12
5. Security Considerations.....	14
6. Conclusions.....	14
7. References.....	14
7.1. Normative References.....	14
7.2. Informative References.....	14
8. Acknowledgments.....	15
Appendix A. Abbreviations used in the text of this document.....	16
Appendix B. Used terms.....	17

## 1. Introduction

In this document, the requirements for a video codec designed mainly for use over the Internet are presented. The requirements encompass a wide range of applications that use data transmission over the Internet including IPTV (broadcasting over IP-based networks), peer-to-peer video conferencing, video sharing, screencasting, and video monitoring/ surveillance. For each application, typical resolutions, frame-rates and picture access modes are presented. Specific requirements related to data transmission over packet-loss networks are considered as well. In this document, when we discuss data protection techniques we only refer to methods designed and implemented to protect data inside the video codec since there are many existing techniques that protect generic data transmitted over packet-loss networks. From the theoretical point of view, both packet-loss and bit-error robustness can be beneficial for video codecs. In practice, packet losses are a more significant problem than bit corruption in IP networks. It is worth noting that there is an evident interdependence between possible amount of delay and the necessity of error robust video streams:

- o If an amount of delay is not crucial for an application, then reliable transport protocols such as TCP that resends undelivered packets can be used to guarantee correct decoding of transmitted data.
- o If the amount of delay must be kept low, then either data transmission should be error free (e.g., by using managed networks) or compressed video stream should be error resilient.

Thus, error resilience can be useful for delay-critical applications to provide low delay in packet-loss environment.

## 2. Applications

In this chapter, an overview of video codec applications that are currently available on the Internet market is presented. It is worth noting that there are different use cases for each application that define a target platform, and hence there are different types of communication channels involved (e.g., wired or wireless channels) that are characterized by different quality of service as well as bandwidth; for instance, wired channels are considerably more error-free than wireless channels and therefore require different QoS approaches. The target platform, the channel bandwidth and the channel quality determine resolutions, frame-rates and quality or bit-rates for video streams to be encoded or decoded. By default,

color format YUV 4:2:0 is assumed for the application scenarios listed below.

## 2.1. Internet Protocol Television (IPTV) / IP-based over-the-top (OTT) video transmission

This is a service for delivering television content over IP-based networks. IPTV may be classified into two main groups based on the type of delivery, as follows:

- o unicast (e.g., for video on demand), where delay is not crucial and, hence, error resilience is not needed;
- o multicast/broadcast (e.g., for transmitting news) where zapping, i.e. stream changing, delay is important and, therefore, error resilience is required in the case of unmanaged networks like the Internet.

The main difference between IPTV and IP-based OTT video transmission is that traffic is transmitted over managed (QoS-based) and unmanaged networks in the above cases, respectively. Typical content used in this application is news, movies, cartoons, series, TV shows, etc. One important requirement for both groups is Random access to pictures, i.e. random access period (RAP) should be kept small enough (approximately, 1-15 seconds). For the second group, two further requirements should be met:

- o Temporal (frame-rate) scalability;
- o Error robustness (only for IP-based OTT video transmission).

For the first use case, the two above-mentioned requirements are optional. Support of resolution and quality (SNR) scalability is highly desirable for the both groups. For this application, typical values of resolutions, frame-rates, and RAPs are presented in Table 1.

Resolution *	Frame-rate, fps	PAM
2160p (4K), 3840x2160	60	RA

1080p, 1920x1080	24, 50, 60	RA
1080i, 1920x1080 *	30 (60 fields per second)	RA
720p, 1280x720	50, 60	RA
576p (EDTV), 720x576	25, 50	RA
576i (SDTV), 720x576	25, 30	RA
480p (EDTV), 720x480	50, 60	RA
480i (SDTV), 720x480	25, 30	RA

Table 1. IPTV: typical values of resolutions, frame-rates, and RAPs

NB \*: interlaced content can be handled at the higher system level and not necessarily by using specialized video coding tools. It is included in this table only for the sake of completeness as most video content today is in progressive format.

## 2.2. Video conferencing

This is a form of video connection over the Internet. This form allows users to establish connections to two or more people by two-way video and audio transmission for communication in real-time. For this application, both stationary and mobile devices can be used. The main requirements are as follows:

- o Delay should be kept as low as possible (the preferable and maximum end-to-end delay values should be less than 100 ms [6] and 320 ms [1], respectively);
- o Temporal (frame-rate) scalability;
- o Error robustness.

Support of resolution and quality (SNR) scalability is highly desirable. For this application, typical values of resolutions, frame-rates, and RAPs are presented in Table 2.

Resolution	Frame-rate, fps	PAM
------------	-----------------	-----

1080p, 1920x1080	15, 30	JFPIC
720p, 1280x720	30, 60	JFPIC
4CIF, 704x576	30, 60	JFPIC
4SIF, 704x480	30, 60	JFPIC
VGA, 640x480	30, 60	JFPIC
360p, 640x360	30, 60	JFPIC

Table 2. Video conferencing: typical values of resolutions, frame-rates, and RAPs

### 2.3. Video sharing

This is a service that allows people to upload and share video data (using live streaming or not) and to watch them. It is also known as video hosting. A typical User-generated Content (UGC) scenario for this application is to capture video using mobile cameras such as GoPro or cameras integrated into smartphones (amateur video). The main requirements are as follows:

- o Random access to pictures for downloaded video data;
- o Temporal (frame-rate) scalability;
- o Error robustness.

Support of resolution and quality (SNR) scalability is highly desirable. For this application, typical values of resolutions, frame-rates, and RAPs are presented in Table 3.

Resolution	Frame-rate, fps	PAM
2160p (4K), 3840x2160	24, 25, 30, 48, 50, 60	RA
1440p (2K), 2560x1440	24, 25, 30, 48, 50, 60	RA
1080p, 1920x1080	24, 25, 30, 48, 50, 60	RA
720p, 1280x720	24, 25, 30, 48, 50, 60	RA



480p, 854x480	24, 25, 30, 48, 50, 60	RA	
+-----+	+-----+	+-----+	+-----+
360p, 640x360	24, 25, 30, 48, 50, 60	RA	
+-----+	+-----+	+-----+	+-----+

Table 3. Video sharing: typical values of resolutions, frame-rates [7], and RAPs

#### 2.4. Screencasting

This is a service that allows users to record and distribute computer desktop screen output. This service requires efficient compression of computer-generated content with high visual quality (up to visually and mathematically lossless) [8]. Currently, this application includes business presentations (powerpoint, word documents, email messages, etc.), animation (cartoons), gaming content, data visualization, i.e. such type of content that is characterized by fast motion, rotation, smooth shade, 3D effect, highly saturated colors with full resolution, clear textures and sharp edges with distinct colors [8]), virtual desktop infrastructure (VDI), screen/desktop sharing and collaboration, supervisory control and data acquisition (SCADA) display, automotive/navigation display, cloud gaming, factory automation display, wireless display, display wall, digital operating room (DiOR), etc. For this application, an important requirement is the support of a wide range of video formats (e.g., RGB) in addition to YUV 4:2:0 and YUV 4:4:4 [8]. For this application, typical values of resolutions, frame-rates, and RAPs are presented in Table 4.

Resolution	Frame-rate, fps	PAM
Input color format: RGB		
WQXGA, 2560x1600	15, 30, 60	AI, RA, JFPIC
WUXGA, 1920x1200	15, 30, 60	AI, RA, JFPIC
WSXGA+, 1680x1050	15, 30, 60	AI, RA, JFPIC
WXGA, 1280x800	15, 30, 60	AI, RA, JFPIC
XGA, 1024x768	15, 30, 60	AI, RA, JFPIC
SVGA, 800x600	15, 30, 60	AI, RA, JFPIC

VGA, 640x480	15, 30, 60	AI, RA, JFPIC
Input color format: YUV 4:4:4		
1440p (2K), 2560x1440	15, 30, 60	AI, RA, JFPIC
1080p, 1920x1080	15, 30, 60	AI, RA, JFPIC
720p, 1280x720	15, 30, 60	AI, RA, JFPIC

Table 4. Screencasting for RGB and YUV 4:4:4 format: typical values of resolutions, frame-rates, and RAPs

## 2.5. Game streaming

This is a service that provides game content over the Internet to different local devices such as notebooks, gaming tablets, etc. In this category of applications, server renders 3D games in cloud server, and streams the game to any device with a wired or wireless broadband connection [9]. There are low latency requirements for transmitting user interactions and receiving game data in less than a turn-around delay of 100 ms. This allows anyone to play (or resume) full featured games from anywhere in the Internet [9]. An example of this application is Nvidia Grid [9]. Another category application is broadcast of video games played by people over the Internet in real time or for later viewing [9]. There are many companies such as Twitch, YY in China enable game broadcasting [9]. Games typically contain a lot of sharp edges and large motion [9]. The main requirements are as follows:

- o Random access to pictures for game broadcasting;
- o Temporal (frame-rate) scalability;
- o Error robustness.

Support of resolution and quality (SNR) scalability is highly desirable. For this application, typical values of resolutions, frame-rates, and RAPs are similar to ones presented in Table 4.

## 2.6. Video monitoring / surveillance

This is a type of live broadcasting over IP-based networks. Video streams are sent to many receivers at the same time. A new receiver may connect to the stream at an arbitrary moment, so random access period should be kept small enough (approximately, ~1-5 seconds).

Data are transmitted publicly in the case of video monitoring and privately in the case of video surveillance, respectively. For IP-cameras that have to capture, process and encode video data, complexity including computational and hardware complexity as well as memory bandwidth should be kept low to allow real-time processing. In addition, support of high dynamic range as well as resolution and quality (SNR) scalability is an essential requirement for video surveillance. For this application, typical values of resolutions, frame-rates, and RAPs are presented in Table 5.

Resolution	Frame-rate, fps	PAM
2160p (4K), 3840x2160	12	RA, JFPIC
5Mpixels, 2560x1920	12	RA, JFPIC
1080p, 1920x1080	25	RA, JFPIC
1.3Mpixels, 1280x960	25, 30	RA, JFPIC
720p, 1280x720	25, 30	RA, JFPIC
SVGA, 800x600	25, 30	RA, JFPIC

Table 5. Video monitoring / surveillance: typical values of resolutions, frame-rates, and RAPs

### 3. Requirements

Taking the requirements discussed above for specific video applications, this chapter proposes requirements for an internet video codec. The most basic requirement is coding efficiency, i.e. compression performance. It should be better than for state-of-the-art video codecs such as HEVC/H.265 and VP9. Levels to be supported by the new codec are presented in Table 6.

Level	Example picture resolution at highest frame rate
1	128x96@30.0 176x144@15.0

2	176x144@100.0 352x288@30.0
3	352x288@60.0 640x360@30.0
4	640x360@60.0 960x540@30.0
5	720x576@75.0 960x540@60.0 1280x720@30.0
6	1,280x720@68.0 2,048x1,080@30.0
7	1,280x720@120.0 2,048x1,080@60.0
8	1,920x1,080@120.0 3,840x2,160@30.0 4,096x2,160@30.0
9	1,920x1,080@250.0 4,096x2,160@60.0
10	1,920x1,080@300.0 4,096x2,160@120.0
11	3,840x2,160@120.0 8,192x4,320@30.0
12	3,840x2,160@250.0 8,192x4,320@60.0
13	3,840x2,160@300.0 8,192x4,320@120.0

Table 6. Codec levels

## 3.1. Basic requirements

## 3.1.1. Input source formats:

- o Bit depth: 8- and 10-bits per color component;

- o Color sampling formats: YUV 4:2:0, YUV 4:4:4

- o Support of arbitrary resolution

#### 3.1.2. Coding delay:

- o Support of "low-delay" configurations (end-to-end delay should be up to 320 ms [1] but it's preferable value should be less than 100 ms [6])

#### 3.1.3. Complexity:

- o Feasible real-time implementation of both an encoder and a decoder for hardware and software implementation based on a wide range of state-of-the-art platforms

#### 3.1.4. Scalability:

- o Temporal (frame-rate) scalability

#### 3.1.5. Error resilience:

- o Error resilience tools that are complementary to the error protection mechanisms implemented on transport level.

### 3.2. Optional requirements

#### 3.2.1. Input source formats

- o Bit depth: up to 16-bits per color component;
- o Color sampling formats: RGB and YUV 4:2:2;
- o Auxiliary channel (e.g., alpha channel) support;
- o Support of high dynamic range

#### 3.2.2. Scalability:

- o Resolution and quality (SNR) scalability;
- o Computational complexity scalability, i.e. computational complexity is decreasing along with degrading picture quality

### 3.2.3. Complexity:

Tools that enable parallel processing (e.g., slices, tiles, wave front propagation processing) at both encoder and decoder sides are highly desirable for many applications.

- o High-level multi-core parallelism: encoder and decoder operation, especially entropy encoding and decoding, should allow multiple frames or sub-frame regions (e.g. 1D slices, 2D tiles, or partitions) to be processed concurrently, either independently or with deterministic dependencies that can be efficiently pipelined
- o Low-level instruction set parallelism: favor algorithms that are SIMD/GPU friendly over inherently serial algorithms

## 4. Evaluation methodology

### 4.1. Compression performance evaluation

As shown in Fig.1, compression performance testing is performed in 3 ranges that encompass 12 different bit-rate values:

- o Low bit-rate range (LBR) is the range that contains the 4 lowest bit-rates of the 12 specified bit-rates;
- o Medium bit-rate range (MBR) is the range that contains the 4 medium bit-rates of the 12 specified bit-rates;
- o High bit-rate range (HBR) is the range that contains the 4 highest bit-rates of the 12 specified bit-rates.

To avoid any rate control mechanisms that can significantly impact evaluation results, just nominal values of bit-rates should be specified in a separate document on Internet video codec testing. The deviation between nominal and actual values of bit-rates obtained for both reference and tested codecs should be less than the threshold value defined in the above-mention document on Internet video codec testing. This deviation is calculated as follows:

$$D = \text{abs}((\text{BRa} - \text{BRn}) / \text{BRn}) * 100\%,$$

where BRn is a nominal value of bit-rate, BRa is an actual value of bit-rate obtained for either reference or tested codecs.

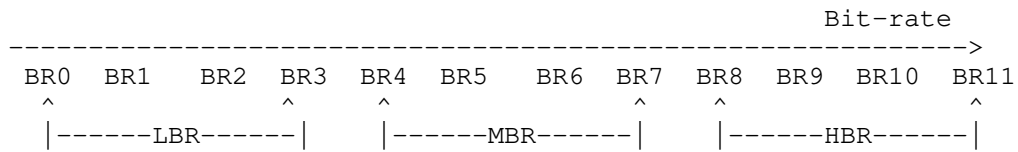


Figure 1 Bit-rate ranges for the CBR mode

To assess the quality of output (decoded) sequences, two indexes, PSNR [2] and MS-SSIM [2,10], should be separately calculated for each color plane. For obtaining an integral estimation, BD-rate [11] should be computed for each range and each quality index. Finally, 18 values should be obtained for a color format, which contains 3 color planes (e.g., for YUV or RGB). A list of video sequences that should be used for testing as well as the 6 values of bit-rates are defined in a separate document. Testing processes should use the information on the codec applications presented in this document. As the reference for evaluation, the HEVC/H.265 codec [3,4] must be used. The reference source code of the HEVC/H.265 codec can be found at [5]. The HEVC/H.265 codec must be configured according to [12] and Table 9.

Intra-period, second	HEVC/H.265 encoding mode according to [12]
AI	Intra Main or Intra Main10
RA	Random access Main or Random access Main10
JFPIC	Low delay Main or Low delay Main10

Table 9. Intra-periods for different HEVC/H.265 encoding modes according to [12]

In addition to the objective quality measures defined above, subjective evaluation must also be performed before adopting any new tool and a final codec standard. For subjective tests, the MOS-based evaluation procedure must be used as described in section 2.1 of [2]. For perception-oriented tools that primarily impact subjective quality, additional tests may also be individually assigned even for intermediate evaluation, subject to a decision of the NETVC WG.

## 5. Security Considerations

This document itself does not address any security considerations. However, it is worth noting that a codec implementation (for both an encoder and a decoder) should cover the worst case of computational complexity, memory bandwidth, and physical memory size (e.g., for decoded pictures used as references). Otherwise, it can be considered as a security vulnerability and lead to denial-of-service (DoS) in the case of attacks.

## 6. Conclusions

In this document, an overview of Internet video codec applications and typical use cases as well as a prioritized list of requirements for an Internet video codec are presented. An evaluation methodology for this codec is also proposed.

## 7. References

### 7.1. Normative References

- [1] Recommendation ITU-T G.1091: Quality of Experience requirements for telepresence services, 2014.
- [2] ISO/IEC PDTR 29170-1: Information technology -- Advanced image coding and evaluation methodologies -- Part 1: Guidelines for codec evaluation.
- [3] ISO/IEC 23008-2:2015. Information technology -- High efficiency coding and media delivery in heterogeneous environments -- Part 2: High efficiency video coding
- [4] Recommendation ITU-T H.265: High efficiency video coding, 2013.
- [5] [https://hevc.hhi.fraunhofer.de/svn/svn\\_HEVCSoftware/](https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/)

### 7.2. Informative References

- [6] S. Wenger, "The case for scalability support in version 1 of Future Video Coding," contribution COM 16-C 988 R1-E to ITU-T SG16/Q6, September 2015. "Recommended upload encoding settings (Advanced) "
- [7] "Recommended upload encoding settings (Advanced) "  
<https://support.google.com/youtube/answer/1722171?hl=en>



- [8] H. Yu, K. McCann, R. Cohen, and P. Amon, "Requirements for future extensions of HEVC in coding screen content", ISO/IEC JTC1/SC29/WG11 MPEG2013/N14174, San Jose, USA, Jan. 2014
- [9] Manindra Parhy, "Game streaming requirement for Future Video Coding," MPEG Contribution m36771, June 2015, Warsaw, Poland.
- [10] Z. Wang, E. P. Simoncelli, and A. C. Bovik, "Multi-scale structural similarity for image quality assessment," Invited Paper, IEEE Asilomar Conference on Signals, Systems and Computers, Nov. 2003, Vol. 2, pp. 1398-1402.
- [11] G. Bjontegaard, "Calculation of average PSNR differences between RD-curves (VCEG-M33)," in VCEG Meeting (ITU-T SG16 Q.6), Austin, Texas, USA, Apr. 2-4 2001.
- [12] F. Bossen, "Common test conditions and software reference configurations," JCTVC-L1100, Geneva, Switzerland, Jan. 2013.
- [13] <http://www.digitizationguidelines.gov/term.php?term=compressionvisuallylossless>

## 8. Acknowledgments

The author would like to thank Mr. Jiantong Zhou, Mr. Paul Coverdale, Mr. Jose Alvarez, Mr. Vasily Rufitskiy, Dr. Haitao Yang, Mr. Viktor Stepin, Mr. Maxim Sychev, and Mr. Sergey Ikonin for many useful discussions on this document and their help while preparing it as well as Mr. Mo Zanaty, Dr. Ali Begen, Mr. Thomas Daede, Dr. Jean-Marc Valin, Mr. Jack Moffitt and Mr. Greg Coppa for their valuable comments on the initial or/and first revisions of this document.

This document was prepared using 2-Word-v2.0.template.dot.

## Appendix A.                      Abbreviations used in the text of this document

Abbreviation	Meaning
AI	All-Intra (each picture is intra-coded)
BD-Rate	Bjontegaard Delta Rate
GOP	Group of Picture
HBR	High Bit-rate Range
PAM	Picture Access Mode
RA	Random Access
RAP	Random Access Period
IPTV	Internet Protocol Television
JFPIC	Just the First Picture is Intra-Coded
LBR	Low Bit-rate Range
MBR	Medium Bit-rate Range
MOS	Mean Opinion Score
MS-SSIM	Multi-Scale Structural Similarity quality index
OTT	Over-The-Top
PSNR	Peak Signal-to-Noise Ratio
QoS	Quality of Service
UGC	User-Generated Content
VDI	Virtual Desktop Infrastructure

## Appendix B.                      Used terms

Term	Meaning
Random access period	is the period of time between two closest independently decodable frames (pictures).
Visually lossless compression	is a form or manner of lossy compression where the data that are lost after the file is compressed and decompressed is not detectable to the eye; the compressed data appearing identical to the uncompressed data [13].

Authors' Addresses

Alexey Filippov  
Huawei Technologies

Email: alexey.filippov@huawei.com



Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: May 4, 2017

A. Fuldseth  
G. Bjontegaard  
S. Midtskogen  
T. Davies  
M. Zanaty  
Cisco  
October 31, 2016

Thor Video Codec  
draft-fuldseth-netvc-thor-03

Abstract

This document provides a high-level description of the Thor video codec. Thor is designed to achieve high compression efficiency with moderate complexity, using the well-known hybrid video coding approach of motion-compensated prediction and transform coding.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 4, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Definitions . . . . .	5
2.1. Requirements Language . . . . .	5
2.2. Terminology . . . . .	6
3. Block Structure . . . . .	6
3.1. Super Blocks and Coding Blocks . . . . .	6
3.2. Special Processing at Frame Boundaries . . . . .	7
3.3. Transform Blocks . . . . .	8
3.4. Prediction Blocks . . . . .	8
4. Intra Prediction . . . . .	8
5. Inter Prediction . . . . .	9
5.1. Multiple Reference Frames . . . . .	9
5.2. Bi-Prediction . . . . .	10
5.3. Improved chroma prediction . . . . .	10
5.4. Reordered Frames . . . . .	10
5.5. Interpolated Reference Frames . . . . .	10
5.6. Sub-Pixel Interpolation . . . . .	10
5.6.1. Luma Poly-phase Filter . . . . .	10
5.6.2. Luma Special Filter Position . . . . .	12
5.6.3. Chroma Poly-phase Filter . . . . .	13
5.7. Motion Vector Coding . . . . .	13
5.7.1. Inter0 and Inter1 Modes . . . . .	13
5.7.2. Inter2 and Bi-Prediction Modes . . . . .	15
5.7.3. Motion Vector Direction . . . . .	16
6. Transforms . . . . .	16
7. Quantization . . . . .	16
7.1. Quantization matrices . . . . .	17
7.1.1. Quantization matrix selection . . . . .	17
7.1.2. Quantization matrix design . . . . .	18
8. Loop Filtering . . . . .	18
8.1. Deblocking . . . . .	18
8.1.1. Luma deblocking . . . . .	18
8.1.2. Chroma Deblocking . . . . .	19
8.2. Constrained Low Pass Filter (CLPF) . . . . .	20
9. Entropy coding . . . . .	20
9.1. Overview . . . . .	20
9.2. Low Level Syntax . . . . .	21
9.2.1. CB Level . . . . .	21
9.2.2. PB Level . . . . .	21
9.2.3. TB Level . . . . .	22
9.2.4. Super Mode . . . . .	22
9.2.5. CBP . . . . .	23
9.2.6. Transform Coefficients . . . . .	23

10. High Level Syntax . . . . .	25
10.1. Sequence Header . . . . .	25
10.2. Frame Header . . . . .	26
11. IANA Considerations . . . . .	27
12. Security Considerations . . . . .	27
13. Normative References . . . . .	27
Authors' Addresses . . . . .	27

## 1. Introduction

This document provides a high-level description of the Thor video codec. Thor is designed to achieve high compression efficiency with moderate complexity, using the well-known hybrid video coding approach of motion-compensated prediction and transform coding.

The Thor video codec is a block-based hybrid video codec similar in structure to widespread standards. The high level encoder and decoder structures are illustrated in Figure 1 and Figure 2 respectively.



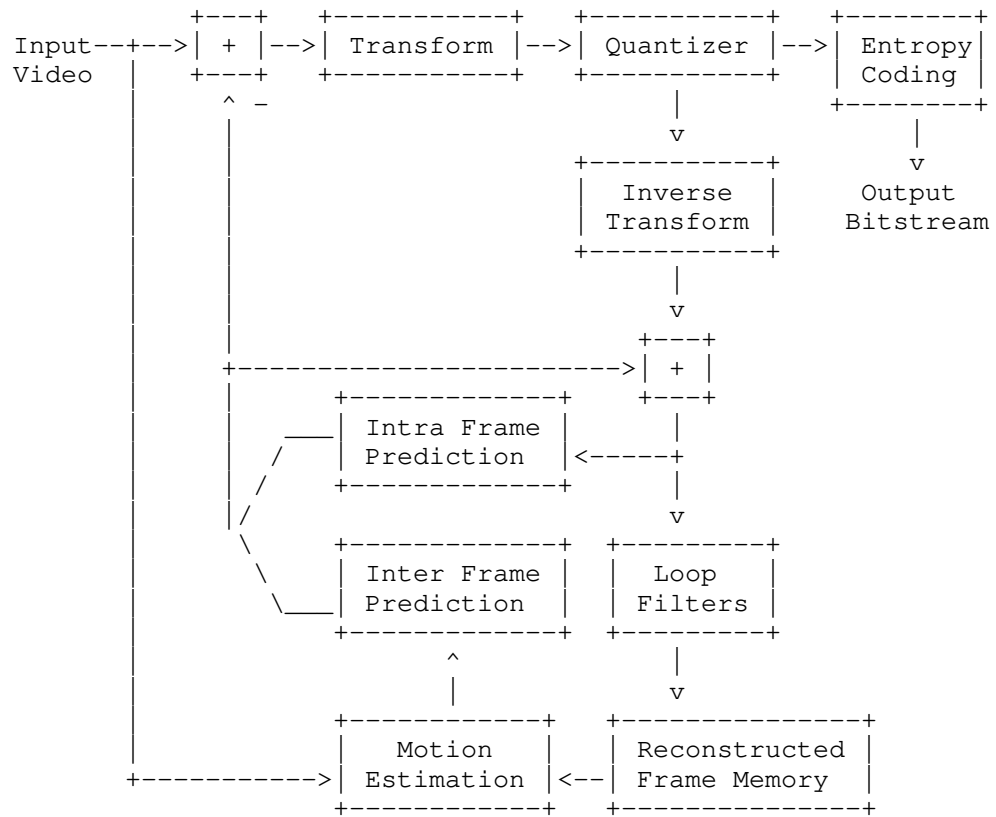


Figure 1: Encoder Structure

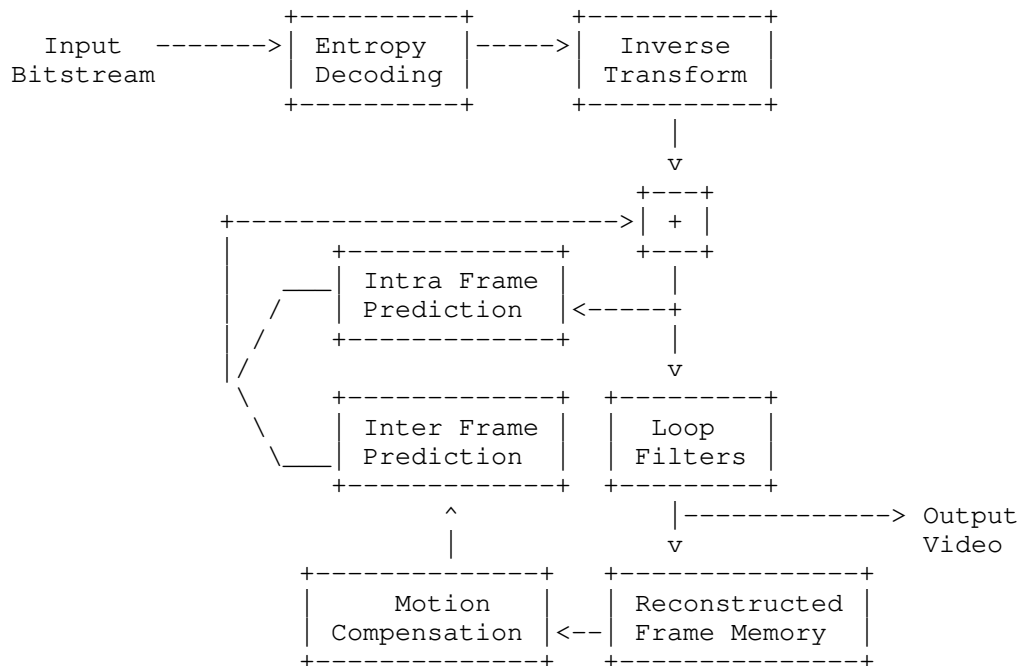


Figure 2: Decoder Structure

The remainder of this document is organized as follows. First, some requirements language and terms are defined. Block structures are described in detail, followed by intra-frame prediction techniques, inter-frame prediction techniques, transforms, quantization, loop filters, entropy coding, and finally high level syntax.

An open source reference implementation is available at [github.com/cisco/thor](https://github.com/cisco/thor).

## 2. Definitions

### 2.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

## 2.2. Terminology

This document frequently uses the following terms.

SB: Super Block - 64x64 or 128x128 block (luma pixels) which can be divided into CBs.

CB: Coding Block - Subdivision of a SB, down to 8x8 (luma pixels).

PB: Prediction Block - Subdivision of a CB, into 1, 2 or 4 equal blocks.

TB: Transform Block - Subdivision of a CB, into 1 or 4 equal blocks.

## 3. Block Structure

### 3.1. Super Blocks and Coding Blocks

Input frames with bitdepths of 8, 10 or 12 are supported. The internal bitdepth can be 8, 10 or 12 regardless of input bitdepth. The bitdepth of the output frames always follows the input frames. Chroma can be subsampled in both directions (4:2:0) or have full resolution (4:4:4).

Each frame is divided into 64x64 or 128x128 Super Blocks (SB) which are processed in raster-scan order. The SB size is signaled in the sequence header. Each SB can be divided into Coding Blocks (CB) using a quad-tree structure. The smallest allowed CB size is 8x8 luma pixels. The four CBs of a larger block are coded/signaled in the following order; upleft, downleft, upright, and downright.

The following modes are signaled at the CB level:

- o Intra
- o Inter0 (skip): MV index, no residual information
- o Inter1 (merge): MV index, residual information
- o Inter2 (uni-pred): explicit motion information, residual information
- o Inter3 (bi-pred): explicit motion information, residual information

### 3.2. Special Processing at Frame Boundaries

At frame boundaries some square blocks might not be complete. For example, for 1920x1080 resolutions, the bottom row would consist of rectangular blocks of size 64x56. Rectangular blocks at frame boundaries are handled as follows. For each rectangular block, send one bit to choose between:

- o A rectangular inter0 block and
- o Further split.

For the bottom part of a 1920x1080 frame, this implies the following:

- o For each 64x56 block, transmit one bit to signal a 64x56 inter0 block or a split into two 32x32 blocks and two 32x24 blocks.
- o For each 32x24 block, transmit one bit to signal a 32x24 inter0 block or a split into two 16x16 blocks and two 16x8 blocks.
- o For each 16x8 block, transmit one bit to signal a 16x8 inter0 block or a split into two 8x8 blocks.

Two examples of handling 64x56 blocks at the bottom row of a 1920x1080 frame are shown in Figure 3 and Figure 4 respectively.

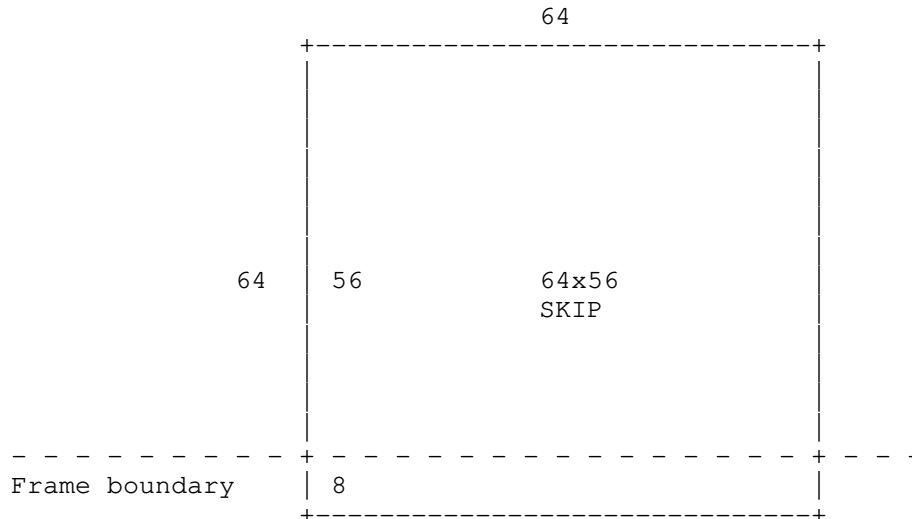


Figure 3: Super block at frame boundary

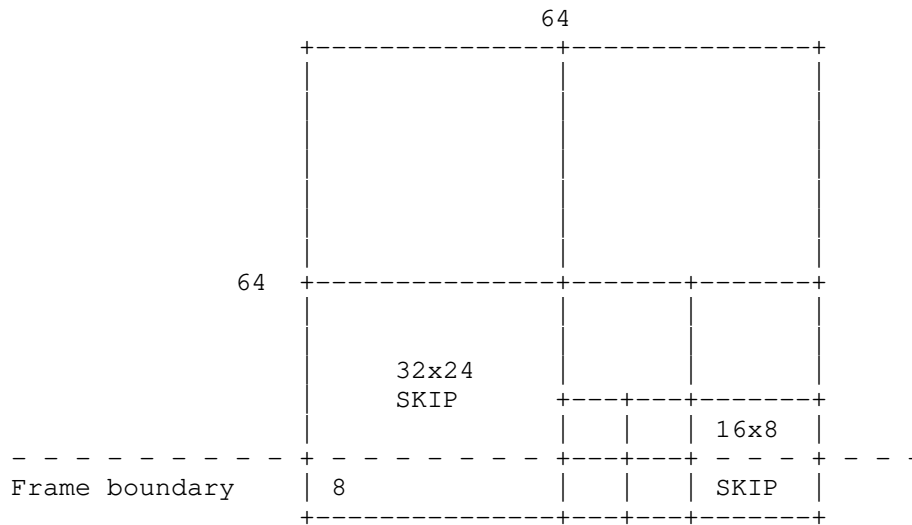


Figure 4: Coding block at frame boundary

### 3.3. Transform Blocks

A coding block (CB) can be divided into four smaller transform blocks (TBs).

### 3.4. Prediction Blocks

A coding block (CB) can also be divided into smaller prediction blocks (PBs) for the purpose of motion-compensated prediction. Horizontal, vertical and quad split are used.

#### 4. Intra Prediction

8 intra prediction modes are used:

1. DC
2. Vertical (V)
3. Horizontal (H)
4. Upupright (north-northeast)
5. Upupleft (north-northwest)
6. Upleft (northwest)

7. Upleftleft (west-northwest)

8. Downleftleft (west-southwest)

The definition of DC, vertical, and horizontal modes are straightforward.

The upleft direction is exactly 45 degrees.

The upupright, upupleft, and upleftleft directions are equal to  $\arctan(1/2)$  from the horizontal or vertical direction, since they are defined by going one pixel horizontally and two pixels vertically (or vice versa).

For the 5 angular intra modes (i.e. angle different from 90 degrees), the pixels of the neighbor blocks are filtered before they are used for prediction:

$$y(n) = (x(n-1) + 2*x(n) + x(n+1) + 2)/4$$

For the angular intra modes that are not 45 degrees, the prediction sometimes requires sample values at a half-pixel position. These sample values are determined by an additional filter:

$$z(n + 1/2) = (y(n) + y(n+1))/2$$

## 5. Inter Prediction

### 5.1. Multiple Reference Frames

Multiple reference frames are currently implemented as follows.

- o Use a sliding-window process to keep the N most recent reconstructed frames in memory. The value of N is signaled in the sequence header.
- o In the frame header, signal which of these frames shall be active for the current frame.
- o For each CB, signal which of the active frames to be used for MC.

Combined with re-ordering, this allows for MPEG-1 style B frames.

A desirable future extension is to allow long-term reference frames in addition to the short-term reference frames defined by the sliding-window process.

## 5.2. Bi-Prediction

In case of bi-prediction, two reference indices and two motion vectors are signaled per CB. In the current version, PB-split is not allowed in bi-prediction mode. Sub-pixel interpolation is performed for each motion vector/reference index separately before doing an average between the two predicted blocks:

$$p(x,y) = (p0(x,y) + p1(x,y))/2$$

## 5.3. Improved chroma prediction

If specified in the sequence header, the chroma prediction, both intra and inter, or either, is improved by using the luma reconstruction if certain criteria are met. The process is described in the separate CLPF draft [I-D.midtskogen-netvc-chromapred].

## 5.4. Reordered Frames

Frames may be transmitted out of order. Reference frames are selected from the sliding window buffer as normal.

## 5.5. Interpolated Reference Frames

A flag is sent in the sequence header indicating that interpolated reference frames may be used.

If a frame is using an interpolated reference frame, it will be the first reference in the reference list, and will be interpolated from the second and third reference in the list. It is indicated by a reference index of -1 and has a frame number equal to that of the current frame.

The interpolated reference is created by a deterministic process common to the encoder and decoder, and described in the separate IRFVC draft [I-D.davies-netvc-irfvc].

## 5.6. Sub-Pixel Interpolation

### 5.6.1. Luma Poly-phase Filter

Inter prediction uses traditional block-based motion compensated prediction with quarter pixel resolution. A separable 6-tap poly-phase filter is the basis method for doing MC with sub-pixel accuracy. The luma filter coefficients are as follows:

When bi-prediction is enabled in the sequence header:

1/4 phase: [2,-10,59,17,-5,1]/64

2/4 phase: [1,-8,39,39,-8,1]/64

3/4 phase: [1,-5,17,59,-10,2]/64

When bi-prediction is disabled in the sequence header:

1/4 phase: [1,-7,55,19,-5,1]/64

2/4 phase: [1,-7,38,38,-7,1]/64

3/4 phase: [1,-5,19,55,-7,1]/64

With reference to Figure 5, a fractional sample value, e.g.  $i_{0,0}$  which has a phase of 1/4 in the horizontal dimension and a phase of 1/2 in the vertical dimension is calculated as follows:

$$a_{0,j} = 2*A_{-2,i} - 10*A_{-1,i} + 59*A_{0,i} + 17*A_{1,i} - 5*A_{2,i} + 1*A_{3,i}$$

where  $j = -2, \dots, 3$

$$i_{0,0} = (1*a_{0,-2} - 8*a_{0,-1} + 39*a_{0,0} + 39*a_{0,1} - 8*a_{0,2} + 1*a_{0,3} + 2048)/4096$$

The minimum sub-block size is 8x8.



A				A	a	b	c	A
-1,-1				0,-1	0,-1	0,-1	0,-1	1,-1
A				A	a	b	c	A
-1,0				0,0	0,0	0,0	0,0	1,0
d				d	e	f	g	d
-1,0				0,0	0,0	0,0	0,0	1,0
h				h	i	j	k	h
-1,0				0,0	0,0	0,0	0,0	1,0
l				l	m	n	o	l
-1,0				0,0	0,0	0,0	0,0	1,0
A				A	a	b	c	A
-1,1				0,1	0,1	0,1	0,1	1,1

Figure 5: Sub-pixel positions

#### 5.6.2. Luma Special Filter Position

For the fractional pixel position having exactly 2 quarter pixel offsets in each dimension, a non-separable filter is used to calculate the interpolated value. With reference to Figure 5, the center position  $j0,0$  is calculated as follows:

$j0,0 =$

$[0 \cdot A_{-1,-1} + 1 \cdot A_{0,-1} + 1 \cdot A_{1,-1} + 0 \cdot A_{2,-1} +$   
 $1 \cdot A_{-1,0} + 2 \cdot A_{0,0} + 2 \cdot A_{1,0} + 1 \cdot A_{2,0} +$   
 $1 \cdot A_{-1,1} + 2 \cdot A_{0,1} + 2 \cdot A_{1,1} + 1 \cdot A_{2,1} +$

$$0*A_{-1,2} + 1*A_{0,2} + 1*A_{1,2} + 0*A_{2,2} + 8]/16$$

#### 5.6.3. Chroma Poly-phase Filter

Chroma interpolation is performed with 1/8 pixel resolution using the following poly-phase filter.

1/8 phase: [-2, 58, 10, -2]/64

2/8 phase: [-4, 54, 16, -2]/64

3/8 phase: [-4, 44, 28, -4]/64

4/8 phase: [-4, 36, 36, -4]/64

5/8 phase: [-4, 28, 44, -4]/64

6/8 phase: [-2, 16, 54, -4]/64

7/8 phase: [-2, 10, 58, -2]/64

#### 5.7. Motion Vector Coding

##### 5.7.1. Inter0 and Inter1 Modes

Inter0 and inter1 modes imply signaling of a motion vector index to choose a motion vector from a list of candidate motion vectors with associated reference frame index. A list of motion vector candidates are derived from at most two different neighbor blocks, each having a unique motion vector/reference frame index. Signaling of the motion vector index uses 0 or 1 bit, dependent on the number of unique motion vector candidates. If the chosen neighbor block is coded in bi-prediction mode, the inter0 or inter1 block inherits both motion vectors, both reference indices and the bi-prediction property of the neighbor block.

For block sizes less than 64x64, inter0 has only one motion vector candidate, and its value is always zero.

Which neighbor blocks to use for motion vector candidates depends on the availability of the neighbor blocks (i.e. whether the neighbor blocks have already been coded, belong to the same slice and are not outside the frame boundaries). Four different availabilities, U, UR, L, and LL, are defined as illustrated in Figure 6. If the neighbor block is intra it is considered to be available but with a zero motion vector.

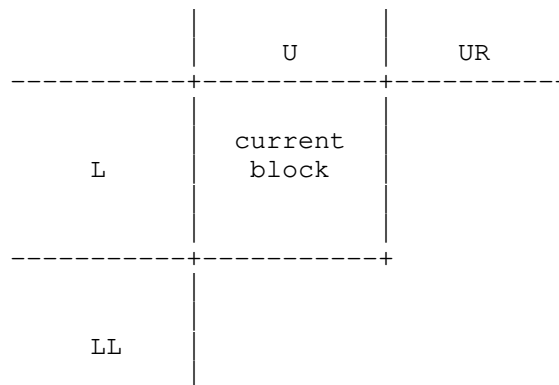


Figure 6: Availability of neighbor blocks

Based on the four availabilities defined above, each of the motion vector candidates is derived from one of the possible neighbor blocks defined in Figure 7.

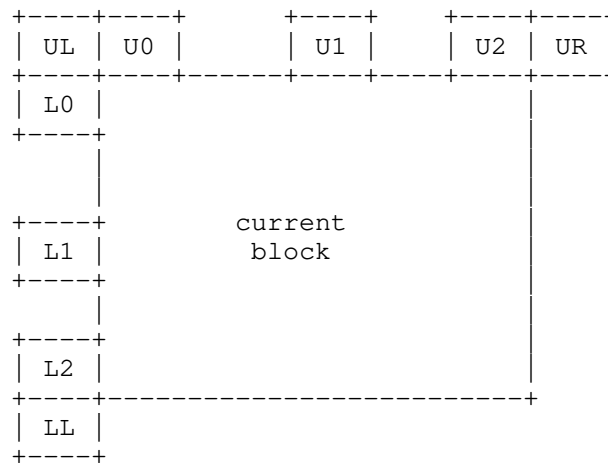


Figure 7: Motion vector candidates

The choice of motion vector candidates depends on the availability of neighbor blocks as shown in Table 1.

U	UR	L	LL	Motion vector candidates
0	0	0	0	zero vector
1	0	0	0	U2, zero vector
0	1	0	0	NA
1	1	0	0	U2, zero vector
0	0	1	0	L2, zero vector
1	0	1	0	U2, L2
0	1	1	0	NA
1	1	1	0	U2, L2
0	0	0	1	NA
1	0	0	1	NA
0	1	0	1	NA
1	1	0	1	NA
0	0	1	1	L2, zero vector
1	0	1	1	U2, L2
0	1	1	1	NA
1	1	1	1	U2, L2

Table 1: Motion vector candidates for different availability of neighbor blocks

#### 5.7.2. Inter2 and Bi-Prediction Modes

Motion vectors are coded using motion vector prediction. The motion vector predictor is defined as the median of the motion vectors from three neighbor blocks. Definition of the motion vector predictor uses the same definition of availability and neighbors as in Figure 6 and Figure 7 respectively. The three vectors used for median filtering depends on the availability of neighbor blocks as shown in Table 2. If the neighbor block is coded in bi-prediction mode, only the first motion vector (in transmission order), MV0, is used as input to the median operator.

U	UR	L	LL	Motion vectors for median filtering
0	0	0	0	3 x zero vector
1	0	0	0	U0,U1,U2
0	1	0	0	NA
1	1	0	0	U0,U2,UR
0	0	1	0	L0,L1,L2
1	0	1	0	UL,U2,L2
0	1	1	0	NA
1	1	1	0	U0,UR,L2,L0
0	0	0	1	NA
1	0	0	1	NA
0	1	0	1	NA
1	1	0	1	NA
0	0	1	1	L0,L2,LL
1	0	1	1	U2,L0,LL
0	1	1	1	NA
1	1	1	1	U0,UR,L0

Table 2: Neighbor blocks used to define motion vector predictor through median filtering

### 5.7.3. Motion Vector Direction

Motion vectors referring to reference frames later in time than the current frame are stored with their sign reversed, and these reversed values are used for coding and motion vector prediction.

## 6. Transforms

Transforms are applied at the TB or CB level, implying that transform sizes range from 4x4 to 128x128. The transforms form an embedded structure meaning the transform matrix elements of the smaller transforms can be extracted from the larger transforms.

## 7. Quantization

For the 32x32, 64x64 and 128x128 transform sizes, only the 16x16 low frequency coefficients are quantized and transmitted.

The 64x64 inverse transform is defined as a 32x32 transform followed by duplicating each output sample into a 2x2 block. The 128x128 inverse transform is defined as a 32x32 transform followed by duplicating each output sample into a 4x4 block.

### 7.1. Quantization matrices

A flag is transmitted in the sequence header to indicate whether quantization matrices are used. If this flag is true, a 6 bit value `qmtx_offset` is transmitted in the sequence header to indicate matrix strength.

If used, then in dequantization a separate scaling factor is applied to each coefficient, so that the dequantized value of a coefficient `ci` at position `i` is:

$$(ci * d(q) * IW(i,c,s,t,q) + 2^{(k + 5)}) >> (k + 6)$$

Figure 8: Equation 1

where `IW` is the scale factor for coefficient position `i` with size `s`, frame type (inter/inter) `t`, component (Y, Cb or Cr) `c` and quantizer `q`; and `k=k(s,q)` is the dequantization shift. `IW` has scale 64, that is, a weight value of 64 is no different to unweighted dequantization.

#### 7.1.1. Quantization matrix selection

The current luma `qp` value `qpY` and the offset value `qmtx_offset` determine a quantisation matrix set by the formula:

$$qmlevel = \max(0, \min(11, ((qpY + qmtx\_offset) * 12) / 44))$$

Figure 9: Equation 2

This selects one of the 12 different sets of default quantization matrix, with increasing `qmlevel` indicating increasing flatness.

For a given value of `qmlevel`, different weighting matrices are provided for all combinations of transform block size, type (intra/inter), and component (Y, Cb, Cr). Matrices at low `qmlevel` are flat (constant value 64). Matrices for inter frames have unity DC gain (i.e. value 64 at position 0), whereas those for intra frames are designed such that the inverse weighting matrix has unity energy gain (i.e. normalized sum-squared of the scaling factors is 1).

### 7.1.2. Quantization matrix design

Further details on the quantization matrix and implementation can be found in the separate QMTX draft [I-D.davies-netvc-qmtx].

## 8. Loop Filtering

### 8.1. Deblocking

#### 8.1.1. Luma deblocking

Luma deblocking is performed on an 8x8 grid as follows:

1. For each vertical edge between two 8x8 blocks, calculate the following for each of line 2 and line 5 respectively:

$$d = \text{abs}(a-b) + \text{abs}(c-d),$$

where  $a$  and  $b$ , are on the left hand side of the block edge and  $c$  and  $d$  are on the right hand side of the block edge:

a b | c d

2. For each line crossing the vertical edge, perform deblocking if and only if all of the following conditions are true:

- \*  $d2+d5 < \text{beta}(\text{QP})$

- \* The edge is also a transform block edge

- \*  $\text{abs}(\text{mvx}(\text{left})) > 2$ , or  $\text{abs}(\text{mvx}(\text{right})) > 2$ , or

- $\text{abs}(\text{mvy}(\text{left})) > 2$ , or  $\text{abs}(\text{mvy}(\text{right})) > 2$ , or

- One of the transform blocks on each side of the edge has non-zero coefficients, or

- One of the transform blocks on each side of the edge is coded using intra mode.

3. If deblocking is performed, calculate a delta value as follows:

$$\text{delta} = \text{clip}((18*(c-b) - 6*(d-a) + 16)/32, \text{tc}, -\text{tc}),$$

where  $\text{tc}$  is a QP-dependent value.

4. Next, modify two pixels on each side of the block edge as follows:

$$a' = a + \text{delta}/2$$

$$b' = b + \text{delta}$$

$$c' = c + \text{delta}$$

$$d' = d + \text{delta}/2$$

5. The same procedure is followed for horizontal block edges.

The relative positions of the samples,  $a$ ,  $b$ ,  $c$ ,  $d$  and the motion vectors,  $MV$ , are illustrated in Figure 10.

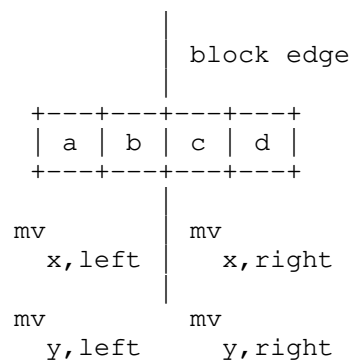


Figure 10: Deblocking filter pixel positions

#### 8.1.2. Chroma Deblocking

Chroma deblocking is performed on a 4x4 grid as follows:

1. Delocking of the edge between two 4x4 blocks is performed if and only if:
  - \* The pixels on either side of the block edge belongs to an intra block.
  - \* The block edge is also an edge between two transform blocks.
2. If deblocking is performed, calculate a delta value as follows:
 
$$\text{delta} = \text{clip}((4*(c-b) + (d-a) + 4)/8, \text{tc}, -\text{tc}),$$
 where  $\text{tc}$  is a QP-dependent value.



3. Next, modify one pixel on each side of the block edge as follows:

$$b' = b + \text{delta}$$

$$c' = c + \text{delta}$$

## 8.2. Constrained Low Pass Filter (CLPF)

A low-pass filter is applied after the deblocking filter if signaled in the sequence header. It can still be switched off for individual frames in the frame header. Also signaled in the frame header is whether to apply the filter for all qualified 128x128 blocks or to transmit a flag for each such block. A super block does not qualify if it only contains Inter0 (skip) coding block and no signal is transmitted for these blocks.

The filter is described in the separate CLPF draft [I-D.midtskogen-netvc-clpf].

## 9. Entropy coding

### 9.1. Overview

The following information is signaled at the sequence level:

- o Sequence header

The following information is signaled at the frame level:

- o Frame header

The following information is signaled at the CB level:

- o Super-mode (mode, split, reference index for uni-prediction)
- o Intra prediction mode
- o PB-split (none, hor, ver, quad)
- o TB-split (none or quad)
- o Reference frame indices for bi-prediction
- o Motion vector candidate index
- o Transform coefficients if TB-split=0

The following information is signaled at the TB level:

- o CBP (8 combinations of CBPY, CBPU, and CBPV)
- o Transform coefficients

The following information is signaled at the PB level:

- o Motion vector differences

## 9.2. Low Level Syntax

### 9.2.1. CB Level

super-mode (inter0/split/inter1/inter2-ref0/intra/inter2-ref1/inter2-ref2/inter2-ref3,...)

if (mode == inter0 || mode == inter1)

mv\_idx (one of up to 2 motion vector candidates)

else if (mode == INTRA)

intra\_mode (one of up to 8 intra modes)

tb\_split (NONE or QUAD, coded jointly with CBP for tb\_split=NONE)

else if (mode == INTER)

pb\_split (NONE, VER, HOR, QUAD)

tb\_split\_and\_cbp (NONE or QUAD and CBP)

else if (mode == BIPRED)

mvd\_x0, mvd\_y0 (motion vector difference for first vector)

mvd\_x1, mvd\_y1 (motion vector difference for second vector)

ref\_idx0, ref\_idx1 (two reference indices)

### 9.2.2. PB Level

if (mode == INTER2 || mode == BIPRED)

mvd\_x, mvd\_y (motion vector differences)

## 9.2.3. TB Level

```

    if (mode != INTER0 and tb_split == 1)

        cbp                                (8 possibilities for CBPY/CBPU/CBPV)

    if (mode != INTER0)

        transform coefficients

```

## 9.2.4. Super Mode

For each block of size  $N \times N$  ( $64 \geq N > 8$ ), the following mutually exclusive events are jointly encoded using a single VLC code as follows (example using 4 reference frames):

If there is no interpolated reference frame:

INTER0	1
SPLIT	01
INTER1	001
INTER2-REF0	0001
BIPRED	00001
INTRA	000001
INTER2-REF1	0000001
INTER2-REF2	00000001
INTER2-REF3	00000000

If there is an interpolated reference frame:

INTER0	1
SPLIT	01
INTER1	001
BIPRED	0001
INTRA	00001
INTER2-REF1	000001
INTER2-REF2	0000001
INTER2-REF3	00000001
INTER2-REF0	00000000

If less than 4 reference frames is used, a shorter VLC table is used. If bi-pred is not possible, or split is not possible, they are omitted from the table and shorter codes are used for subsequent elements.

Additionally, depending on information from the blocks to the left and above (meta data and CBP), a different sorting of the events can be used, e.g.:

SPLIT	1
INTER1	01
INTER2-REF0	001
INTER0	0001
INTRA	00001
INTER2-REF1	000001
INTER2-REF2	0000001
INTER2-REF3	00000001
BIPRED	00000000

#### 9.2.5. CBP

Calculate code as follows:

```
if (tb-split == 0)

    N = 4*CBPV + 2*CBPU + CBPY

else

    N = 8
```

Map the value of N to code through a table lookup:

```
code = table[N]
```

where the purpose of the table lookup is the sort the different values of code according to decreasing probability (typically CBPY=1, CBPU=0, CBPV=0 having the highest probability).

Use a different table depending on the values of CBPY in neighbor blocks (left and above).

Encode the value of code using a systematic VLC code.

#### 9.2.6. Transform Coefficients

Transform coefficient coding uses a traditional zig-zag scan pattern to convert a 2D array of quantized transform coefficients, *coeff*, to a 1D array of samples. VLC coding of quantized transform coefficients starts from the low frequency end of the 1D array using two different modes; level-mode and run-mode, starting in level-mode:

- o Level-mode
  - \* Encode each coefficient, *coeff*, separately
  - \* Each coefficient is encoded by:

- + The absolute value,  $\text{level}=\text{abs}(\text{coeff})$ , using a VLC code and
  - + If  $\text{level} > 0$ , the sign bit ( $\text{sign}=0$  or  $\text{sign}=1$  for  $\text{coeff}>0$  and  $\text{coeff}<0$  respectively).
  - \* If coefficient N is zero, switch to run-mode, starting from coefficient N+1.
- o Run-mode
- \* For each non-zero coefficient, encode the combined event of:
    1. Length of the zero-run, i.e. the number of zeros since the last non-zero coefficient.
    2. Whether or not  $\text{level}=\text{abs}(\text{coeff})$  is greater than 1.
    3. End of block (EOB) indicating that there are no more non-zero coefficients.
  - \* Additionally, if  $\text{level} = 1$ , code the sign bit.
  - \* Additionally, if  $\text{level} > 1$  define  $\text{code} = 2 \cdot (\text{level}-2) + \text{sign}$ ,
  - \* If the absolute value of coefficient N is larger than 1, switch to level-mode, starting from coefficient N+1.

#### Example

Figure 11 illustrates an example where 16 quantized transform coefficients are encoded.

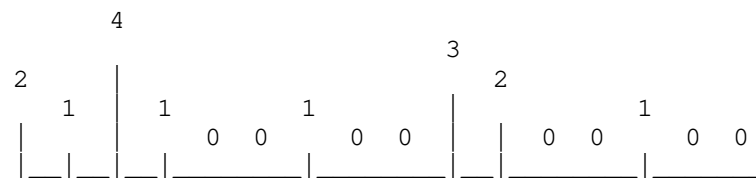


Figure 11: Coefficients to encode

Table 3 shows the mode, VLC number and symbols to be coded for each coefficient.

Index	abs (coeff)	Mode	Encoded symbols
0	2	level-mode	level=2, sign
1	1	level-mode	level=1, sign
2	4	level-mode	level=4, sign
3	1	level-mode	level=1, sign
4	0	level-mode	level=0
5	0	run-mode	
6	1	run-mode	(run=1, level=1)
7	0	run-mode	
8	0	run-mode	
9	3	run-mode	(run=1, level>1), 2*(3-2)+sign
10	2	level-mode	level=2, sign
11	0	level-mode	level=0
12	0	run-mode	
13	1	run-mode	(run=1, level=1)
14	0	run-mode	EOB
15	0	run-mode	

Table 3: Transform coefficient encoding for the example above.

## 10. High Level Syntax

High level syntax is currently very simple and rudimentary as the primary focus so far has been on compression performance. It is expected to evolve as functionality is added.

### 10.1. Sequence Header

- o Width - 16 bits
- o Height - 16 bits
- o Enable/disable PB-split - 1 bit
- o SB size - 3 bits
- o Enable/disable TB-split - 1 bit
- o Number of active reference frames (may go into frame header) - 2 bits (max 4)
- o Enable/disable interpolated reference frames - 1 bit
- o Enable/disable delta qp - 1 bit

- o Enable/disable deblocking - 1 bit
- o Constrained low-pass filter (CLPF) enable/disable - 1 bit
- o Enable/disable block context coding - 1 bit
- o Enable/disable bi-prediction - 1 bit
- o Enable/disable quantization matrices - 1 bit
- o If quantization matrices enabled: quantization matrix offset - 6 bits
- o Select 420 or 444 input - 1 bit
- o Number of reordered frames - 4 bits
- o Enable/disable chroma intra prediction from luma - 1 bit
- o Enable/disable chroma inter prediction from luma - 1 bit
- o Internal frame bitdepth (8, 10 or 12 bits) - 2 bits
- o Input video bitdepth (8, 10 or 12 bits) - 2 bits

#### 10.2. Frame Header

- o Frame type - 1 bit
- o QP - 8 bits
- o Identification of active reference frames - num\_ref\*4 bits
- o Number of intra modes - 4 bits
- o Number of active reference frames - 2 bits
- o Active reference frames - number of active reference frames \* 6 bits
- o Frame number - 16 bits
- o If CLPF is enabled in the sequence header: Constrained low-pass filter (CLPF) strength - 2 bits (00 = off, 01 = strength 1, 10 = strength 2, 11 = strength 4)
- o IF CLPF is enabled in the sequence header: Enable/disable CLPF signal for each qualified filter block

## 11. IANA Considerations

This document has no IANA considerations yet. TBD

## 12. Security Considerations

This document has no security considerations yet. TBD

## 13. Normative References

[I-D.davies-netvc-irfvc]

Davies, T., "Interpolated reference frames for video coding", draft-davies-netvc-irfvc-00 (work in progress), October 2015.

[I-D.davies-netvc-qmtx]

Davies, T., "Quantisation matrices for Thor video coding", draft-davies-netvc-qmtx-00 (work in progress), March 2016.

[I-D.midtskogen-netvc-chromapred]

Midtskogen, S., "Improved chroma prediction", draft-midtskogen-netvc-chromapred-02 (work in progress), October 2016.

[I-D.midtskogen-netvc-clpf]

Midtskogen, S., Fuldseth, A., and M. Zanaty, "Constrained Low Pass Filter", draft-midtskogen-netvc-clpf-02 (work in progress), April 2016.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.

## Authors' Addresses

Arild Fuldseth  
Cisco  
Lysaker  
Norway

Email: [arilfuld@cisco.com](mailto:arilfuld@cisco.com)



Gisle Bjontegaard  
Cisco  
Lysaker  
Norway

Email: gbjonteg@cisco.com

Steinar Midtskogen  
Cisco  
Lysaker  
Norway

Email: stemidts@cisco.com

Thomas Davies  
Cisco  
London  
UK

Email: thdavies@cisco.com

Mo Zanaty  
Cisco  
RTP, NC  
USA

Email: mzanaty@cisco.com

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: September 11, 2017

S. Midtskogen  
A. Fuldseth  
M. Zanaty  
Cisco  
March 10, 2017

Constrained Low Pass Filter  
draft-midtskogen-netvc-clpf-04

Abstract

This document describes a low complexity filtering technique which is being used as a low pass loop filter in the Thor video codec.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 11, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Definitions . . . . .	2
2.1. Requirements Language . . . . .	2
2.2. Terminology . . . . .	2
3. Filtering Process . . . . .	3
4. Further complexity considerations . . . . .	6
5. Performance . . . . .	6
6. IANA Considerations . . . . .	7
7. Security Considerations . . . . .	7
8. Acknowledgements . . . . .	7
9. References . . . . .	8
9.1. Normative References . . . . .	8
9.2. Informative References . . . . .	8
Authors' Addresses . . . . .	8

## 1. Introduction

Modern video coding standards such as Thor [I-D.fuldseth-netvc-thor] include in-loop filters which correct artifacts introduced in the encoding process. Thor includes a deblocking filter which corrects artifacts introduced by the block based nature of the encoding process, and a low pass filter correcting artifacts not corrected by the deblocking filter, in particular artifacts introduced by quantisation errors of transform coefficients and by the interpolation filter. Since in-loop filters have to be applied in both the encoder and decoder, it is highly desirable that these filters have low computational complexity.

## 2. Definitions

## 2.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

## 2.2. Terminology

This document will refer to a pixel X and eight of its neighbouring pixels A - H ordered in the following pattern.

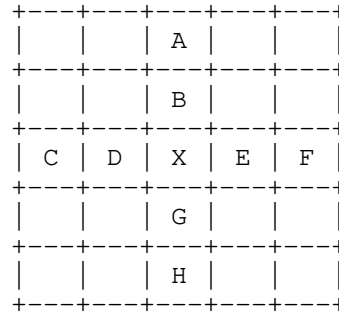


Figure 1: Filter pixel positions

In Thor the frames are divided into filter blocks (FB) of 128x128, 64x64 or 32x32 pixels, which is signalled for each frame to be filtered. Also, each frame is divided into coding blocks (CB) which range from 8x8 to 128x128 independent of the FB size. The filter described in this draft can be switched on or off for the entire frame or optionally on or off for each FB. CB's that have been coded using the skip mode are not filtered, and if a FB only contains CB's that have been coded in skip mode, the FB will not be filtered and no signal will be transmitted for this FB.

If the frame can't fit a whole number of FB's, the FB's at the right and bottom edges are clipped to fit. For instance, if the frame resolution is 1920x1080 and the FB size is 128x128, the size of the FB's at the bottom of the frame becomes 128x56.

### 3. Filtering Process

Given a pixel X and its neighbouring pixels described above we can define a general non-linear filter as:

$$X' = X + a*\text{constrain}(A-X) + b*\text{constrain}(B-X) + c*\text{constrain}(C-X) + d*\text{constrain}(D-X) + e*\text{constrain}(E-X) + f*\text{constrain}(F-X) + g*\text{constrain}(G-X) + h*\text{constrain}(H-X)$$

Figure 2: Equation 1

where constrain(x) is a function limiting the range of x.

If a neighbour pixel is outside the image frame, it is given the same value as the closest pixel within the frame. To avoid dependencies

prohibiting parallel processing, all neighbour pixels must be the unfiltered pixels of the frame being filtered.

Experiments in Thor have shown that a good compromise between complexity and performance is  $a=c=f=h=1/16$ ,  $b=d=e=g=3/16$ , and a good constrain function has been found to be:

```
constrain(x, s, d) =
    sign(x) * max(0, abs(x) - max(0, abs(x) - s +
                                (abs(x) >> (d - log2(s))))))
```

Figure 3: Equation 2

where  $\text{sign}(x)$  returns 1 or -1 if  $x$  is positive or negative respectively,  $s$  denotes the strength of the filter by which  $x$  will be clipped, and  $d$  further constrains the range of  $x$  so that the output of the function will linearly approach 0 as  $\text{abs}(x)$  approaches  $2^d$ .  $d$  depends on the frame quality ( $qp$ ) and is computed as

$$d = \text{bitdepth} - 4 + qp/16$$

Figure 4: Equation 4

for the luma plane and

$$d = \text{bitdepth} - 5 + qp/16$$

Figure 5: Equation 5

for the chroma planes.

The constrain function can be visualised as follows

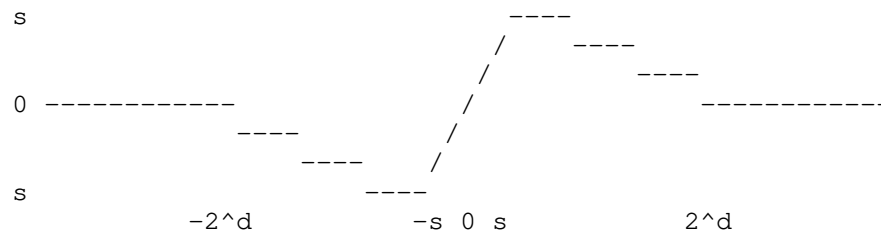


Figure 6: Graph 1

The filter strength  $s$  can be 1, 2 or 4 signalled at frame level when the bitdepth is 8. The strengths are scaled according to the bitdepth, so they become 4, 8 and 16 when the bitdepth is 10, and 16, 32 and 64 when the bitdepth is 12. The rounding is to the nearest integer.

This gives us the equation:

$$X' = X + (1 \cdot \text{constrain}(A-X, s, d) + 3 \cdot \text{constrain}(B-X, s, d) + 1 \cdot \text{constrain}(C-X, s, d) + 3 \cdot \text{constrain}(D-X, s, d) + 3 \cdot \text{constrain}(E-X, s, d) + 1 \cdot \text{constrain}(F-X, s, d) + 3 \cdot \text{constrain}(G-X, s, d) + 1 \cdot \text{constrain}(H-X, s, d))$$

Figure 7: Equation 6

The filter leaves the encoder 13 different choices for a frame. The filter can be disabled for the entire frame, or the frame is filtered using all distinct combinations of strength (1, 2 or 4 scaled for bitdepth), non-skip FB signal (enabled/disabled) and FB size (32x32, 64x64 or 128x128). Note that the FB size only matters when FB signalling is in use.

The decisions at both frame level and FB level may be based on rate-distortion optimisation (RDO), but an encoder running in a low-complexity mode, or possibly a low-delay mode, may instead assume that a fixed mode will be beneficial. In general, using  $s=2$ , a QP dependent FB size and RDO only at the FB level gives good results.

However, because of the low complexity of the filter, fully RDO based decisions are not costly. The distortion of the 13 configurations of the filter can easily be computed in a single pass by keeping track of the distortions of the three different strengths and the bit costs for different FB sizes.

The filter is applied after the deblocking filter.

#### 4. Further complexity considerations

The filter has been designed to offer the best compromise between low complexity and performance. All operations are easily vectorised with SIMD instructions and if the video input is 8 bit, all SIMD operations can have 8 bit lanes in architectures such as x86/SSE4 and ARM/NEON. Clipping at frame borders can be implemented using shuffle instructions.

#### 5. Performance

The table below shows filters effect on the bandwidth for a selection of 10 second video sequences encoded in Thor with uni-prediction only. The numbers have been computed using the Bjontegaard Delta Rate (BDR). BDR-low and BDR-high indicate the effect at low and high bitrates, respectively, as described in BDR [BDR].

The effect of the filter was tested in two encoder low-delay configurations: high complexity in which the encoder strongly favours compression efficiency over CPU usage, and medium complexity which is more suited for real-time applications. The bandwidth reduction is somewhat less in the high complexity configuration.

Sequence	MEDIUM COMPLEXITY			HIGH COMPLEXITY		
	BDR	BDR-low	BDR-high	BDR	BDR-low	BDR-high
Kimono	-2.6%	-2.3%	-3.2%	-1.7%	-1.7%	-1.9%
BasketballDrive	-3.9%	-3.1%	-5.0%	-2.8%	-2.4%	-3.5%
BQTerrace	-7.4%	-4.0%	-10.2%	-4.8%	-2.4%	-6.8%
FourPeople	-5.5%	-3.9%	-8.2%	-3.8%	-2.9%	-5.1%
Johnny	-5.2%	-3.5%	-8.2%	-3.3%	-2.7%	-4.5%
ChangeSeats	-6.4%	-3.9%	-10.2%	-4.7%	-2.6%	-6.9%
HeadAndShoulder	-9.3%	-3.4%	-19.6%	-6.2%	-2.6%	-11.8%
TelePresence	-5.8%	-3.6%	-10.0%	-4.5%	-3.3%	-6.6%
Average	-5.8%	-3.5%	-9.3%	-4.0%	-2.6%	-5.9%

Figure 8: Compression Performance without Biprediction

While the filter objectively performs better at relatively high bitrates, the subjective effect seems better at relatively low bitrates, and overall the subjective effect seems better than what the objective numbers suggest.

If biprediction is allowed, there is generally less bandwidth reduction as the table below shows. These results reflect low-delay biprediction without frame reordering.

	MEDIUM COMPLEXITY			HIGH COMPLEXITY		
Sequence	BDR	BDR-low	BDR-high	BDR	BDR-low	BDR-high
Kimono	-2.2%	-2.0%	-2.7%	-1.4%	-1.3%	-1.5%
BasketballDrive	-3.1%	-3.0%	-3.3%	-1.9%	-2.0%	-1.7%
BQTerrace	-5.4%	-4.3%	-6.5%	-3.9%	-3.6%	-3.8%
FourPeople	-3.8%	-2.8%	-5.2%	-2.4%	-1.8%	-3.0%
Johnny	-3.8%	-3.1%	-4.8%	-2.4%	-2.2%	-2.7%
ChangeSeats	-4.4%	-3.1%	-6.5%	-3.2%	-2.6%	-3.9%
HeadAndShoulder	-4.8%	-3.0%	-8.1%	-3.0%	-2.7%	-3.7%
TelePresence	-3.4%	-2.3%	-5.5%	-2.2%	-1.7%	-3.1%
Average	-3.9%	-2.9%	-5.3%	-2.5%	-2.2%	-2.9%

Figure 9: Compression Performance with Biprediction

## 6. IANA Considerations

This document has no IANA considerations yet. TBD

## 7. Security Considerations

This document has no security considerations yet. TBD

## 8. Acknowledgements

The authors would like to thank Gisle Bjontegaard for reviewing this document and design, and providing constructive feedback and direction.



## 9. References

### 9.1. Normative References

- [I-D.fuldseth-netvc-thor]  
Fuldseth, A., Bjontegaard, G., Midtskogen, S., Davies, T.,  
and M. Zanaty, "Thor Video Codec", draft-fuldseth-netvc-  
thor-03 (work in progress), October 2016.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate  
Requirement Levels", BCP 14, RFC 2119,  
DOI 10.17487/RFC2119, March 1997,  
<<http://www.rfc-editor.org/info/rfc2119>>.

### 9.2. Informative References

- [BDR] Bjontegaard, G., "Calculation of average PSNR differences  
between RD-curves", ITU-T SG16 Q6 VCEG-M33 , April 2001.

## Authors' Addresses

Steinar Midtskogen  
Cisco  
Lysaker  
Norway

Email: [stemidts@cisco.com](mailto:stemidts@cisco.com)

Arild Fuldseth  
Cisco  
Lysaker  
Norway

Email: [arilfuld@cisco.com](mailto:arilfuld@cisco.com)

Mo Zanaty  
Cisco  
RTP, NC  
USA

Email: [mzanaty@cisco.com](mailto:mzanaty@cisco.com)

Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: September 10, 2015

T. Terriberry  
Mozilla Corporation  
March 9, 2015

Coding Tools for a Next Generation Video Codec  
draft-terriberry-codingtools-02

Abstract

This document proposes a number of coding tools that could be incorporated into a next-generation video codec.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 10, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Entropy Coding . . . . .	2
2.1. Non-binary Arithmetic Coding . . . . .	4
2.2. Non-binary Context Modeling . . . . .	4
2.3. Simple Experiment . . . . .	8
3. Reversible Integer Transforms . . . . .	8
3.1. Lifting Steps . . . . .	9
3.2. 4-Point Transform . . . . .	11
3.3. Larger Transforms . . . . .	14
3.4. Walsh-Hadamard Transforms . . . . .	15
4. Development Repository . . . . .	17
5. IANA Considerations . . . . .	17
6. Acknowledgments . . . . .	17
7. References . . . . .	17
7.1. Informative References . . . . .	17
7.2. URIs . . . . .	18
Author's Address . . . . .	18

## 1. Introduction

One of the biggest contributing factors to the success of the Internet is that the underlying protocols are implementable on a royalty-free basis. This allows them to be implemented widely and easily distributed by application developers, service operators, and end users, without asking for permission. In order to produce a next-generation video codec that is competitive with the best patent-encumbered standards, yet avoids patents which are not available on an open-source compatible, royalty-free basis, we must use old coding tools in new ways and develop new coding tools. This draft documents some of the tools we have been working on for inclusion in such a codec. This is early work, and the performance of some of these tools (especially in relation to other approaches) is not yet fully known. Nevertheless, it still serves to outline some possibilities an eventual working group, if formed, could consider.

## 2. Entropy Coding

The basic theory of entropy coding was well-established by the late 1970's [Pas76]. Modern video codecs have focused on Huffman codes (or "Variable-Length Codes"/VLCs) and binary arithmetic coding. Huffman codes are limited in the amount of compression they can provide and the design flexibility they allow, but as each code word consists of an integer number of bits, their implementation complexity is very low, so they were provided at least as an option in every video codec up through H.264. Arithmetic coding, on the other hand, uses code words that can take up fractional parts of a

bit, and are more complex to implement. However, the prevalence of cheap, H.264 High Profile hardware, which requires support for arithmetic coding, shows that it is no longer so expensive that a fallback VLC-based approach is required. Having a single entropy-coding method simplifies both up-front design costs and interoperability.

However, the primary limitation of arithmetic coding is that it is an inherently serial operation. A given symbol cannot be decoded until the previous symbol is decoded, because the bits (if any) that are output depend on the exact state of the decoder at the time it is decoded. This means that a hardware implementation must run at a sufficiently high clock rate to be able to decode all of the symbols in a frame. Higher clock rates lead to increased power consumption, and in some cases the entropy coding is actually becoming the limiting factor in these designs.

As fabrication processes improve, implementers are very willing to trade increased gate count for lower clock speeds. So far, most approaches to allowing parallel entropy coding have focused on splitting the encoded symbols into multiple streams that can be decoded independently. This "independence" requirement has a non-negligible impact on compression, parallelizability, or both. For example, H.264 can split frames into "slices" which might cover only a small subset of the blocks in the frame. In order to allow decoding these slices independently, they cannot use context information from blocks in other slices (harming compression). Those contexts must adapt rapidly to account for the generally small number of symbols available for learning probabilities (also harming compression). In some cases the number of contexts must be reduced to ensure enough symbols are coded in each context to usefully learn probabilities at all (once more, harming compression). Furthermore, an encoder must specially format the stream to use multiple slices per frame to allow any parallel entropy decoding at all. Encoders rarely have enough information to evaluate this "compression efficiency" vs. "parallelizability" trade-off, since they don't generally know the limitations of the decoders for which they are encoding. That means there will be many files or streams which could have been decoded if they were encoded with different options, but which a given decoder cannot decode because of bad choices made by the encoder (at least from the perspective of that decoder). The same set of drawbacks apply to the DCT token partitions in VP8 [RFC6386].

### 2.1. Non-binary Arithmetic Coding

Instead, we propose a very different approach: use non-binary arithmetic coding. In binary arithmetic coding, each decoded symbol has one of two possible values: 0 or 1. The original arithmetic coding algorithms allow a symbol to take on any number of possible values, and allow the size of that alphabet to change with each symbol coded. Reasonable values of  $N$  (for example,  $N \leq 16$ ) offer the potential for a decent throughput increase for a reasonable increase in gate count for hardware implementations.

Binary coding allows a number of computational simplifications. For example, for each coded symbol, the set of valid code points is partitioned in two, and the decoded value is determined by finding the partition in which the actual code point that was received lies. This can be determined by computing a single partition value (in both the encoder and decoder) and (in the decoder) doing a single comparison. A non-binary arithmetic coder partitions the set of valid code points into multiple pieces (one for each possible value of the coded symbol). This requires the encoder to compute two partition values, in general (for both the upper and lower bound of the symbol to encode). The decoder, on the other hand, must search the partitions for the one that contains the received code point. This requires computing at least  $O(\log N)$  partition values.

However, coding a parameter with  $N$  possible values with a binary arithmetic coder requires  $O(\log N)$  symbols in the worst case (the only case that matters for hardware design). Hence, this does not represent any actual savings (indeed, it represents an increase in the number of partition values computed by the encoder). In addition, there are a number of overheads that are per-symbol, rather than per-value. For example, renormalization (which enlarges the set of valid code points after partitioning has reduced it too much), carry propagation (to deal with the case where the high and low ends of a partition straddle a bit boundary), etc., are all performed on a symbol-by-symbol basis. Since a non-binary arithmetic coder codes a given set of values with fewer symbols than a binary one, it incurs these per-symbol overheads less often. This suggests that a non-binary arithmetic coder can actually be more efficient than a binary one.

### 2.2. Non-binary Context Modeling

The other aspect that binary coding simplifies is probability modeling. In arithmetic coding, the size of the sets the code points are partitioned into are (roughly) proportional to the probability of each possible symbol value. Estimating these probabilities is part of the coding process, though it can be cleanly separated from the

task of actually producing the coded bits. In a binary arithmetic coder, this requires estimating the probability of only one of the two possible values (since the total probability is 1.0). This is often done with a simple table lookup that maps the old probability and the most recently decoded symbol to a new probability to use for the next symbol in the current context. The trade-off, of course, is that non-binary symbols must be "binarized" into a series of bits, and a context (with an associated probability) chosen for each one.

In a non-binary arithmetic coder, the decoder must compute at least  $O(\log N)$  cumulative probabilities (one for each partition value it needs). Because these probabilities are usually not estimated directly in "cumulative" form, this can require computing  $(N - 1)$  non-cumulative probability values. Unless  $N$  is very small, these cannot be updated with a single table lookup. The normal approach is to use "frequency counts". Define the frequency of value  $k$  to be

$$f[k] = A * \langle \text{the number of times } k \text{ has been observed} \rangle + B$$

where  $A$  and  $B$  are parameters (usually  $A=2$  and  $B=1$  for a traditional Krichevsky-Trofimov estimator). The resulting probability,  $p[k]$ , is given by

$$ft = \sum_{k=0}^{N-1} f[k]$$

$$p[k] = \frac{f[k]}{ft}$$

When  $ft$  grows too large, the frequencies are rescaled (e.g., halved, rounding up to prevent reduction of a probability to 0).

When  $ft$  is not a power of two, partitioning the code points requires actual divisions (see [RFC6716] Section 4.1 for one detailed example of exactly how this is done). These divisions are acceptable in an audio codec like Opus [RFC6716], which only has to code a few hundreds of these symbols per second. But video requires hundreds of thousands of symbols per second, at a minimum, and divisions are still very expensive to implement in hardware.

There are two possible approaches to this. One is to come up with a replacement for frequency counts that produces probabilities that sum to a power of two. Some possibilities, which can be applied individually or in combination:

1. Use probabilities that are fixed for the duration of a frame. This is the approach taken by VP8, for example, even though it uses a binary arithmetic coder. In fact, it is possible to convert many of VP8's existing binary-alphabet probabilities into probabilities for non-binary alphabets, an approach that is used in the experiment presented at the end of this section.
2. Use parametric distributions. For example, DCT coefficient magnitudes usually have an approximately exponential distribution. This distribution can be characterized by a single parameter, e.g., the expected value. The expected value is trivial to update after decoding a coefficient. For example

$$E[x[n+1]] = E[x[n]] + \text{floor}(C*(x[n] - E[x[n]]))$$

produces an exponential moving average with a decay factor of  $(1 - C)$ . For a choice of  $C$  that is a negative power of two (e.g.,  $1/16$  or  $1/32$  or similar), this can be implemented with two adds and a shift. Given this expected value, the actual distribution to use can be obtained from a small set of pre-computed distributions via a lookup table. Linear interpolation between these pre-computed values can improve accuracy, at the cost of  $O(N)$  computations, but if  $N$  is kept small this is trivially parallelizable, in SIMD or otherwise.

3. Change the frequency count update mechanism so that  $ft$  is constant. For example, let

$$fl[k] = \frac{\sum_{i=0}^{k-1} f[i]}{ft}$$

be the cumulative frequency of all symbol values less than  $k$  and

$$e[i][k] = \begin{cases} 0, & k \leq i \\ 1, & k > i \end{cases}$$

be the elementary change in the cumulative frequency count  $fl[k]$  caused by adding 1 to  $f[i]$ . Then one possible update formula after decoding the value  $i$  is

$$fl[k]' = fl[k] - \text{floor}(D*fl[k]) + k + F*e[i][k]$$

where  $D$  is a negative power of two chosen such that  $\text{floor}(D*ft) == (N + F)$ . This ensures that  $ft == fl[N] == fl[N]'$

is a constant. This requires  $O(N)$  operations, but the arithmetic is very simple (given the freedom to choose  $D$  and  $F$ , and to some extent  $N$ ), and trivially parallelizable, in SIMD or otherwise. The downside is the addition of the value  $k$  at each step. This is necessary to ensure that the probability of an individual symbol ( $fl[k+1] - fl[k]$ ) is never reduced to zero. However it is equivalent to mixing in a uniform distribution with counts that are otherwise an exponential moving average. That means that  $ft$  and  $F$  must be sufficiently large, or there will be an adverse impact on coding efficiency. The upside is that  $F * e[i]$  may be replaced by any monotonically non-decreasing vector whose  $N$ th element is  $F$ . That is, instead of just incrementing the probability of symbol  $i$ , it can increase the probability of values that are highly correlated with  $i$ . E.g., this allows decoding value  $i$  to apply a small probability increase to the neighboring values  $(i - 1)$  and  $(i + 1)$ , in addition to a large probability increase to the value  $i$ . This may help, for example, in motion vector coding, and is much more sensible than the approach taken with binary context modeling, which often does things like "increase the probability of all even values when decoding a 6" because the same context is always used to code the least significant bit.

The other approach is to change the function used to partition the set of valid code points so that it does not need a division, even when  $ft$  is not a power of two. Let the range of valid code points in the current arithmetic coder state be  $[L, L + R)$ , where  $L$  is the lower bound of the range and  $R$  is the number of valid code points. Assume that  $ft \leq R < 2 * ft$  (this is easy to enforce with the normal rescaling operations used with frequency counts). Then one possible partition function is

$$r[k] = fl[k] + \min(fl[k], R - ft)$$

so that the new range after coding symbol  $k$  is  $[L + r[k], L + r[k+1])$ .

This is a variation of the partition function proposed by [SM98]. The size of the new partition ( $r[k+1] - r[k]$ ) is no longer truly proportional to  $R * p[k]$ . It can be off by up to a factor of 2, implying a peak error as large as one bit per symbol. However, if the probabilities are accurate and the symbols being coded are independent, the average inefficiency introduced will be as low as  $\log_2(\log_2(e) * 2/e) \approx 0.0861$  bits per symbol. This error can, of course, be reduced by coding fewer symbols with larger alphabets. In practice the overhead is roughly equal to the overhead introduced by other approximate arithmetic coders like H.264's CABAC.



### 2.3. Simple Experiment

As a simple experiment to validate the non-binary approach, we compared a non-binary arithmetic coder to the VP8 (binary) entropy coder. This was done by instrumenting `vp8_treed_read()` in `libvpx` to dump out the symbol decoded and the associated probabilities used to decode it. This data only includes macroblock mode and motion vector information, as the DCT token data is decoded with custom inline functions, and not `vp8_treed_read()`. This data is available at [1]. It includes 1,019,670 values encode using 2,125,995 binary symbols (or 2.08 symbols per value). We expect that with a conscious effort to group symbols during the codec design, this average could easily be increased.

We then implemented both the regular VP8 entropy decoder (in plain C, using all of the optimizations available in `libvpx` at the time) and a multisymbol entropy decoder (also in plain C, using similar optimizations), which encodes each value with a single symbol. For the decoder partition search in the non-binary decoder, we used a simple for loop ( $O(N)$  worst-case), even though this could be made constant-time and branchless with a few SIMD instructions such as (on x86) `PCMPGTW`, `PACKUSWB`, and `PMOVMASKB` followed by `BSR`. The source code for both implementations is available at [2] (compile with `-DEC_BINARY` for the binary version and `-DEC_MULTISYM` for the non-binary version).

The test simply loads the tokens, and then loops 1024 times encoding them using the probabilities provided, and then decoding them. The loop was added to reduce the impact of the overhead of loading the data, which is implemented very inefficiently. The total runtime on a Core i7 from 2010 is 53.735 seconds for the binary version, and 27.937 seconds for the non-binary version, or a 1.92x improvement. This is very nearly equal to the number of symbols per value in the binary coder, suggesting that the per-symbol overheads account for the vast majority of the computation time in this implementation.

### 3. Reversible Integer Transforms

Integer transforms in image and video coding date back to at least 1969 [PKA69]. Although standards such as MPEG2 and MPEG4 Part 2 allow some flexibility in the transform implementation, implementations were subject to drift and error accumulation, and encoders had to impose special macroblock refresh requirements to avoid these problems, not always successfully. As transforms in modern codecs only account for on the order of 10% of the total decoder complexity, and, with the use of weighted prediction with gains greater than unity and intra prediction, are far more

susceptible to drift and error accumulation, it no longer makes sense to allow a non-exact transform specification.

However, it is also possible to make such transforms "reversible", in the sense that applying the inverse transform to the result of the forward transform gives back the original input values, exactly. This gives a lossy codec, which normally quantizes the coefficients before feeding them into the inverse transform, the ability to scale all the way to lossless compression without requiring any new coding tools. This approach has been used successfully by JPEG XR, for example [TSSRM08].

Such reversible transforms can be constructed using "lifting steps", a series of shear operations that can represent any set of plane rotations, and thus any orthogonal transform. This approach dates back to at least 1992 [BE92], which used it to implement a four-point 1-D Discrete Cosine Transform (DCT). Their implementation requires 6 multiplications, 10 additions, 2 shifts, and 2 negations, and produces output that is a factor of  $\sqrt{2}$  larger than the orthonormal version of the transform. The expansion of the dynamic range directly translates into more bits to code for lossless compression. Because the least significant bits are usually very nearly random noise, this scaling increases the coding cost by approximately half a bit per sample.

### 3.1. Lifting Steps

To demonstrate the idea of lifting steps, consider the two-point transform

$$\begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}$$

This can be implemented up to scale via

$$y_0 = x_0 + x_1$$

$$y_1 = 2 \cdot x_1 - y_0$$

and reversed via

$$x_1 = (y_0 + y_1) \gg 1$$

$$x_0 = y_0 - x_1$$

Both  $y_0$  and  $y_1$  are too large by a factor of  $\sqrt{2}$ , however.

It is also possible to implement any rotation by an angle  $t$ , including the orthonormal scale factor, by decomposing it into three steps:

$$u0 = x0 + \frac{\cos(t) - 1}{\sin(t)} * x1$$

$$y1 = x1 + \sin(t) * u0$$

$$y0 = u0 + \frac{\cos(t) - 1}{\sin(t)} * y1$$

By letting  $t=-\pi/4$ , we get an implementation of the first transform that includes the scaling factor. To get an integer approximation of this transform, we need only replace the transcendental constants by fixed-point approximations:

$$u0 = x0 + ((27 * x1 + 32) >> 6)$$

$$y1 = x1 - ((45 * u0 + 32) >> 6)$$

$$y0 = u0 + ((27 * y1 + 32) >> 6)$$

This approximation is still perfectly reversible:

$$u0 = y0 - ((27 * y1 + 32) >> 6)$$

$$x1 = y1 + ((45 * u0 + 32) >> 6)$$

$$x0 = u0 - ((27 * x1 + 32) >> 6)$$

Each of the three steps can be implemented using just two ARM instructions, with constants that have up to 14 bits of precision (though using fewer bits allows more efficient hardware implementations, at a small cost in coding gain). However, it is still much more complex than the first approach.

We can get a compromise with a slight modification:

$$y0 = x0 + x1$$

$$y1 = x1 - (y0 >> 1)$$

This still only implements the original orthonormal transform up to scale. The  $y0$  coefficient is too large by a factor of  $\sqrt{2}$  as before, but  $y1$  is now too small by a factor of  $\sqrt{2}$ . If our goal

is simply to (optionally quantize) and code the result, this is good enough. The different scale factors can be incorporated into the quantization matrix in the lossy case, and the total expansion is roughly equivalent to that of the orthonormal transform in the lossless case. Plus, we can perform each step with just one ARM instruction.

However, if instead we want to apply additional transformations to the data, or use the result to predict other data, it becomes much more convenient to have uniformly scaled outputs. For a two-point transform, there is little we can do to improve on the three-multiplications approach above. However, for a four-point transform, we can use the last approach and arrange multiple transform stages such that the "too large" and "too small" scaling factors cancel out, producing a result that has the true, uniform, orthonormal scaling. To do this, we need one more tool, which implements the following transform:

$$\begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} \cos(t) & -\sin(t) \\ \sin(t) & \cos(t) \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}$$

This takes unevenly scaled inputs, rescales them, and then rotates them. Like an ordinary rotation, it can be reduced to three lifting steps:

$$\begin{aligned} u_0 &= x_0 + \frac{2\cos(t) - \sqrt{2}}{\sin(t)} * x_1 \\ y_1 &= x_1 + \frac{1}{\sqrt{2}} * \sin(t) * u_0 \\ y_0 &= u_0 + \frac{\cos(t) - \sqrt{2}}{\sin(t)} * y_1 \end{aligned}$$

As before, the transcendental constants may be replaced by fixed-point approximations without harming the reversibility property.

### 3.2. 4-Point Transform

Using the tools from the previous section, we can design a reversible integer four-point DCT approximation with uniform, orthonormal scaling. This requires 3 multiplies, 9 additions, and 2 shifts (not

counting the shift and rounding offset used in the fixed-point multiplies, as these are built into the multiplier). This is significantly cheaper than the [BE92] approach, and the output scaling is smaller by a factor of  $\sqrt{2}$ , saving half a bit per sample in the lossless case. By comparison, the four-point forward DCT approximation used in VP9, which is not reversible, uses 6 multiplies, 6 additions, and 2 shifts (counting shifts and rounding offsets which cannot be merged into a single multiply instruction on ARM). Four of its multipliers also require 28-bit accumulators, whereas this proposal can use much smaller multipliers without giving up the reversibility property. The total dynamic range expansion is 1 bit: inputs in the range  $[-256, 255)$  produce transformed values in the range  $[-512, 510)$ . This is the smallest dynamic range expansion possible for any reversible transform constructed from mostly-linear operations. It is possible to make reversible orthogonal transforms with no dynamic range expansion by using "piecewise-linear" rotations [SLD04], but each step requires a large number of operations in a software implementation.

Pseudo-code for the forward transform follows:

```

Input:  x0, x1, x2, x3
Output: y0, y1, y2, y3
/* Rotate (x3, x0) by -pi/4, asymmetrically scaled output. */
t3  = x0 - x3
t0  = x0 - (t3 >> 1)
/* Rotate (x1, x2) by pi/4, asymmetrically scaled output. */
t2  = x1 + x2
t2h = t2 >> 1
t1  = t2h - x2
/* Rotate (t2, t0) by -pi/4, asymmetrically scaled input. */
y0  = t0 + t2h
y2  = y0 - t2
/* Rotate (t3, t1) by 3*pi/8, asymmetrically scaled input. */
t3  = t3 - (45*t1 + 32 >> 6)
y1  = t1 + (21*t3 + 16 >> 5)
y3  = t3 - (71*y1 + 32 >> 6)

```

Even though there are three asymmetrically scaled rotations by  $\pi/4$ , by careful arrangement we can share one of the shift operations (to help software implementations: shifts by a constant are basically free in hardware). This technique can be used to even greater effect in larger transforms.

The inverse transform is constructed by simply undoing each step in turn:

```

Input:  y0, y1, y2, y3
Output: x0, x1, x2, x3
/* Rotate (y3, y1) by -3*pi/8, asymmetrically scaled output. */
t3  = y3 + (71*y1 + 32 >> 6)
t1  = y1 - (21*t3 + 16 >> 5)
t3  = t3 + (45*t1 + 32 >> 6)
/* Rotate (y2, y0) by pi/4, asymmetrically scaled output. */
t2  = y0 - y2
t2h = t2 >> 1
t0  = y0 - t2h
/* Rotate (t1, t2) by -pi/4, asymmetrically scaled input. */
x2  = t2h - t1
x1  = t2 - x2
/* Rotate (x3, x0) by pi/4, asymmetrically scaled input. */
x0  = t0 - (t3 >> 1)
x3  = x0 - t3

```

Although the right shifts make this transform non-linear, we can compute "basis functions" for it by sending a vector through it with a single value set to a large constant (256 was used here), and the rest of the values set to zero. The true basis functions for a four-point DCT (up to five digits) are

```

[ y0 ]   [ 0.50000  0.50000  0.50000  0.50000 ] [ x0 ]
[ y1 ] = [ 0.65625  0.26953 -0.26953 -0.65625 ] [ x1 ]
[ y2 ]   [ 0.50000 -0.50000 -0.50000  0.50000 ] [ x2 ]
[ y3 ]   [ 0.27344 -0.65234  0.65234 -0.27344 ] [ x3 ]

```

The corresponding basis functions for our reversible, integer DCT, computed using the approximation described above, are

```

[ y0 ]   [ 0.50000  0.50000  0.50000  0.50000 ] [ x0 ]
[ y1 ] = [ 0.65328  0.27060 -0.27060 -0.65328 ] [ x1 ]
[ y2 ]   [ 0.50000 -0.50000 -0.50000  0.50000 ] [ x2 ]
[ y3 ]   [ 0.27060 -0.65328  0.65328 -0.27060 ] [ x3 ]

```

The mean squared error (MSE) of the output, compared to a true DCT, can be computed with some assumptions about the input signal. Let  $G$  be the true DCT basis and  $G'$  be the basis for our integer approximation (computed as described above). Then the error in the transformed results is

$$e = G.x - G'.x = (G - G').x = D.x$$

where  $D = (G - G')$ . The MSE is then [Que98]

$$\begin{aligned}
\frac{1}{N} * E[e^T \cdot e] &= \frac{1}{N} * E[x^T \cdot D^T \cdot D \cdot x] \\
&= \frac{1}{N} * E[\text{tr}(D \cdot x \cdot x^T \cdot D^T)] \\
&= \frac{1}{N} * E[\text{tr}(D \cdot R_{xx} \cdot D^T)]
\end{aligned}$$

where  $R_{xx}$  is the autocorrelation matrix of the input signal. Assuming the input is a zero-mean, first-order autoregressive (AR(1)) process gives an autocorrelation matrix of

$$R_{xx}[i, j] = \rho^{|i - j|}$$

for some correlation coefficient  $\rho$ . A value of  $\rho = 0.95$  is typical for image compression applications. Smaller values are more normal for motion-compensated frame differences, but this makes surprisingly little difference in transform design. Using the above procedure, the theoretical MSE of this approximation is  $1.230E-6$ , which is below the level of the truncation error introduced by the right shift operations. This suggests the dynamic range of the input would have to be more than 20 bits before it became worthwhile to increase the precision of the constants used in the multiplications to improve accuracy, though it may be worth using more precision to reduce bias.

### 3.3. Larger Transforms

The same techniques can be applied to construct a reversible eight-point DCT approximation with uniform, orthonormal scaling using 15 multiplies, 31 additions, and 5 shifts. It is possible to reduce this to 11 multiplies and 29 additions, which is the minimum number of multiplies possible for an eight-point DCT with uniform scaling [LLM89], by introducing a scaling factor of  $\sqrt{2}$ , but this harms lossless performance. The dynamic range expansion is 1.5 bits (again the smallest possible), and the MSE is  $1.592E-06$ . By comparison, the eight-point transform in VP9 uses 12 multiplications, 32 additions, and 6 shifts.

Similarly, we have constructed a reversible sixteen-point DCT approximation with uniform, orthonormal scaling using 33 multiplies, 83 additions, and 16 shifts. This is just 2 multiplies and

2 additions more than the (non-reversible, non-integer, but uniformly scaled) factorization in [LLM89]. By comparison, the sixteen-point transform in VP9 uses 44 multiplies, 88 additions, and 18 shifts. The dynamic range expansion is only 2 bits (again the smallest possible), and the MSE is 1.495E-5.

We also have a reversible 32-point DCT approximation with uniform, orthonormal scaling using 87 multiplies, 215 additions, and 38 shifts. By comparison, the 32-point transform in VP9 uses 116 multiplies, 194 additions, and 66 shifts. Our dynamic range expansion is still the minimal 2.5 bits, and the MSE is 8.006E-05

Code for all of these transforms is available in the development repository listed in Section 4.

### 3.4. Walsh-Hadamard Transforms

These techniques can also be applied to constructing Walsh-Hadamard Transforms, another useful transform family that is cheaper to implement than the DCT (since it requires no multiplications at all). The WHT has many applications as a cheap way to approximately change the time and frequency resolution of a set of data (either individual bands, as in the Opus audio codec, or whole blocks). VP9 uses it as a reversible transform with uniform, orthonormal scaling for lossless coding in place of its DCT, which does not have these properties.

Applying a 2x2 WHT to a block of 2x2 inputs involves running a 2-point WHT on the rows, and then another 2-point WHT on the columns. The basis functions for the 2-point WHT are, up to scaling, [1, 1] and [1, -1]. The four variations of a two-step lifer given in Section 3.1 are exactly the lifting steps needed to implement a 2x2 WHT: two stages that produce asymmetrically scaled outputs followed by two stages that consume asymmetrically scaled inputs.

```
Input:  x00, x01, x10, x11
Output: y00, y01, y10, y11
/* Transform rows */
t1 = x00 - x01
t0 = x00 - (t1 >> 1) /* == (x00 + x01)/2 */
t2 = x10 + x11
t3 = (t2 >> 1) - x11 /* == (x10 - x11)/2 */
/* Transform columns */
y00 = t0 + (t2 >> 1) /* == (x00 + x01 + x10 + x11)/2 */
y10 = y00 - t2       /* == (x00 + x01 - x10 - x11)/2 */
y11 = (t1 >> 1) - t3 /* == (x00 - x01 - x10 + x11)/2 */
y01 = t1 - y11       /* == (x00 - x01 + x10 - x11)/2 */
```



By simply re-ordering the operations, we can see that there are two shifts that may be shared between the two stages:

```

Input:  x00, x01, x10, x11
Output: y00, y01, y10, y11
t1 = x00 - x01
t2 = x10 + x11
t0 = x00 - (t1 >> 1) /* == (x00 + x01)/2 */
y00 = t0 + (t2 >> 1) /* == (x00 + x01 + x10 + x11)/2 */
t3 = (t2 >> 1) - x11 /* == (x10 - x11)/2 */
y11 = (t1 >> 1) - t3 /* == (x00 - x01 - x10 + x11)/2 */
y10 = y00 - t2      /* == (x00 + x01 - x10 - x11)/2 */
y01 = t1 - y11      /* == (x00 - x01 + x10 - x11)/2 */

```

By eliminating the double-negation of x11 and re-ordering the additions to it, we can see even more operations in common:

```

Input:  x00, x01, x10, x11
Output: y00, y01, y10, y11
t1 = x00 - x01
t2 = x10 + x11
t0 = x00 - (t1 >> 1) /* == (x00 + x01)/2 */
y00 = t0 + (t2 >> 1) /* == (x00 + x01 + x10 + x11)/2 */
t3 = x11 + (t1 >> 1) /* == x11 + (x00 - x01)/2 */
y11 = t3 - (t2 >> 1) /* == (x00 - x01 - x10 + x11)/2 */
y10 = y00 - t2      /* == (x00 + x01 - x10 - x11)/2 */
y01 = t1 - y11      /* == (x00 - x01 + x10 - x11)/2 */

```

Simplifying further, the whole transform may be computed with just 7 additions and 1 shift:

```

Input:  x00, x01, x10, x11
Output: y00, y01, y10, y11
t1 = x00 - x01
t2 = x10 + x11
t4 = (t2 - t1) >> 1 /* == (-x00 + x01 + x10 + x11)/2 */
y00 = x00 + t4      /* == (x00 + x01 + x10 + x11)/2 */
y11 = x11 - t4      /* == (x00 - x01 - x10 + x11)/2 */
y10 = y00 - t2      /* == (x00 + x01 - x10 - x11)/2 */
y01 = t1 - y11      /* == (x00 - x01 + x10 - x11)/2 */

```

This is a significant savings over other approaches described in the literature, which require 8 additions, 2 shifts, and 1 negation [FOIK99] (37.5% more operations), or 10 additions, 1 shift, and 2 negations [TSSRM08] (62.5% more operations). The same operations can be applied to compute a 4-point WHT in one dimension. This implementation is used in this way in VP9's lossless mode. Since larger WHTs may be trivially factored into multiple smaller

WHTs, the same approach can implement a reversible, orthonormally scaled WHT of any size  $(2*N) \times (2*M)$ , so long as  $(N + M)$  is even.

#### 4. Development Repository

The tools presented here were developed as part of Xiph.Org's Daala project. They are available, along with many others in greater and lesser states of maturity, in the Daala git repository at [3]. See [4] for more information.

#### 5. IANA Considerations

This document has no actions for IANA.

#### 6. Acknowledgments

Thanks to Nathan Egge, Gregory Maxwell, and Jean-Marc Valin for their assistance in the implementation and experimentation, and in preparing this draft.

#### 7. References

##### 7.1. Informative References

- [RFC6386] Bankoski, J., Koleszar, J., Quillio, L., Salonen, J., Wilkins, P., and Y. Xu, "VP8 Data Format and Decoding Guide", RFC 6386, November 2011.
- [RFC6716] Valin, JM., Vos, K., and T. Terriberry, "Definition of the Opus Audio Codec", RFC 6716, September 2012.
- [BE92] Bruekers, F. and A. van den Enden, "New Networks for Perfect Inversion and Perfect Reconstruction", IEEE Journal on Selected Areas in Communication 10(1):129--137, January 1992.
- [FOIK99] Fukuma, S., Oyama, K., Iwahashi, M., and N. Kambayashi, "Lossless 8-point Fast Discrete Cosine Transform Using Lossless Hadamard Transform", Technical Report The Institute of Electronics, Information, and Communication Engineers of Japan, October 1999.
- [LLM89] Loeffler, C., Ligtenberg, A., and G. Moschytz, "Practical Fast 1-D DCT Algorithms with 11 Multiplications", Proc. Acoustics, Speech, and Signal Processing (ICASSP'89) vol. 2, pp. 988--991, May 1989.

- [Pas76] Pasco, R., "Source Coding Algorithms for Fast Data Compression", Ph.D. Thesis Dept. of Electrical Engineering, Stanford University, May 1976.
- [PKA69] Pratt, W., Kane, J., and H. Andrews, "Hadamard Transform Image Coding", Proc. IEEE 57(1):58--68, Jan 1969.
- [Que98] de Queiroz, R., "On Unitary Transform Approximations", IEEE Signal Processing Letters 5(2):46--47, Feb 1998.
- [SLD04] Senecal, J., Lindstrom, P., and M. Duchaineau, "An Improved N-Bit to N-Bit Reversible Haar-Like Transform", Proc. of the 12th Pacific Conference on Computer Graphics and Applications (PG'04) pp. 371--380, October 2004.
- [SM98] Stuiver, L. and A. Moffat, "Piecewise Integer Mapping for Arithmetic Coding", Proc. of the 17th IEEE Data Compression Conference (DCC'98) pp. 1--10, March/April 1998.
- [TSSRM08] Tu, C., Srinivasan, S., Sullivan, G., Regunathan, S., and H. Malvar, "Low-complexity Hierarchical Lapped Transform for Lossy-to-Lossless Image Coding in JPEG XR/HD Photo", Applications of Digital Image Processing XXXI vol 7073, August 2008.

## 7.2. URIs

- [1] [https://people.xiph.org/~tterribe/daala/ec\\_test0/ec\\_tokens.txt](https://people.xiph.org/~tterribe/daala/ec_test0/ec_tokens.txt)
- [2] [https://people.xiph.org/~tterribe/daala/ec\\_test0/ec\\_test.c](https://people.xiph.org/~tterribe/daala/ec_test0/ec_test.c)
- [3] <https://git.xiph.org/daala.git>
- [4] <https://xiph.org/daala/>

## Author's Address

Timothy B. Terriberry  
Mozilla Corporation  
331 E. Evelyn Avenue  
Mountain View, CA 94041  
USA

Phone: +1 650 903-0800  
Email: [tterribe@xiph.org](mailto:tterribe@xiph.org)

Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: January 7, 2016

T. Terriberry  
Mozilla Corporation  
July 6, 2015

Overlapped Block Motion Compensation for NETVC  
draft-terriberry-netvc-obmc-00

Abstract

This document proposes a scheme for overlapped block motion compensation that could be incorporated into a next-generation video codec. The scheme described is that currently used by Xiph's Daala project, which supports variable block sizes without introducing any discontinuities at block boundaries. This makes the scheme suitable for use with lapped transforms or other techniques where encoding such discontinuities is expensive.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 7, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Adaptive Subdivision OBMC . . . . .	3
3. Implementation and Motion Estimation . . . . .	7
3.1. Initial Estimates . . . . .	10
3.2. Adaptive Subdivision . . . . .	10
3.3. Iterative Refinement . . . . .	12
3.3.1. Rate and Distortion Changes . . . . .	12
3.3.2. Complexity Reduction . . . . .	13
3.3.3. Subpel Refinement . . . . .	14
4. References . . . . .	15
4.1. Informative References . . . . .	15
4.2. URIs . . . . .	16
Author's Address . . . . .	16

## 1. Introduction

Most current video codecs still use straightforward Block Matching Algorithms (BMA) to perform motion compensation, despite their simplicity. These algorithms simply copy a block of pixels from a reference frame, possibly after applying a sub-pixel (subpel) filter to allow for increased motion resolution. When the motion vectors (MVs) of two adjacent blocks differ, a discontinuity is likely to be created along the block edge. These discontinuities are expensive to correct with transform stages that do not themselves have discontinuities along block edges, such as lapped transforms [I-D.egge-videocodec-tdlt]. Even with a more traditional block-based DCT as the transform stage, the creation of these discontinuities requires that some residual be coded to correct them (and to activate loop filtering along these edges) and requires that the size of a transform block used to code that residual be no larger than the size of a prediction block (or they will suffer the same efficiency problem as lapped transforms in correcting them).

Overlapped Block Motion Compensation (OBMC) avoids discontinuities on the block edges by copying slightly larger blocks of pixels, and blending them together with those of neighboring blocks, in an overlapping fashion. Under the assumption that pixels in the reference frames are highly spatially correlated, this blending compensates for motion uncertainty at the pixels farthest from the estimated MVs. This improves the accuracy of the prediction near block edges, making the expected error more uniform across a block, and improving coding performance over a similar BMA scheme (with

fixed-size blocks) by 0.4 dB [K095] to 1.0 dB [KK97], depending on the search strategy used.

Non-overlapped BMA schemes that support varying the block size give much better compression than fixed-size schemes [Lee95]. Although previous formats such as Dirac use OBMC, it has always been with a (small) fixed blending window size. The size of a block might vary from, say, 4x4 to 16x16 pixels, with each block given a single MV, but the overlap with neighboring blocks remains fixed, limited by the smallest supported block size to, say, 2 pixels on either side of an edge (the exact sizes in Dirac are configurable, but do not vary within a frame). This is essentially equivalent to performing prediction for the entire frame at the smallest block size (4x4) with an efficient scheme for specifying that the same MV be used for many of the blocks.

We propose a subdivision scheme for OBMC that supports adaptive blending window sizes, allowing much larger blending windows in blocks that are not subdivided, which previous research has suggested should improve prediction performance compared to fixed-size windows [ZAS98]. Our scheme uses simple window functions that can be computed on the fly, rather than stored in large tables, allowing it to scale to very large block sizes. It admits a dynamic programming algorithm to optimize the subdivision levels with a reasonable complexity.

## 2. Adaptive Subdivision OBMC

Traditional BMA schemes and previous OBMC schemes have a one-to-one correspondence between blocks and MVs, with each block having a single MV. That MV is most reliable in the center of the block, where the prediction error is generally the smallest [ZSNKI02]. Instead, we use a grid of MVs at the corners of blocks. With a fixed-size grid, away from the edges of a frame, this difference is mostly academic: equivalent to shifting block boundaries by half the size of a block in each direction. However, with variable-sized blocks, the distinction becomes more relevant: there is no longer a one-to-one correspondence between blocks and MVs. Under the scheme where MVs correspond to the center of a block, splitting a block removes the old MV at the center of the old block and produces new MVs at the centers of the new blocks. Under the scheme where MVs belong to the corners, splitting a block retains the MVs at the existing corners (corresponding to the same motion as before), but may add new MVs at the new block corners.

We use square blocks with an origin in the upper-left corner and  $x$  and  $y$  coordinates that vary between 0 and 1 within a block. The

vertices and edges of a block are indexed in a clockwise manner, as illustrated in Figure 1.

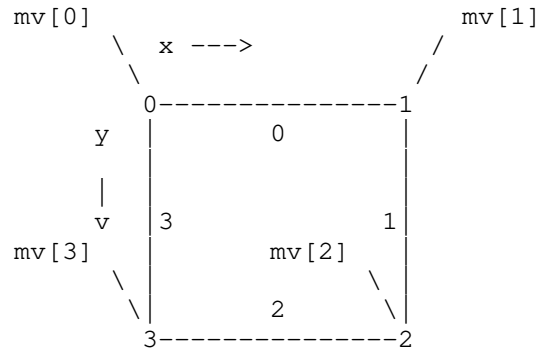


Figure 1: Vertex and Edge Indices for a Block

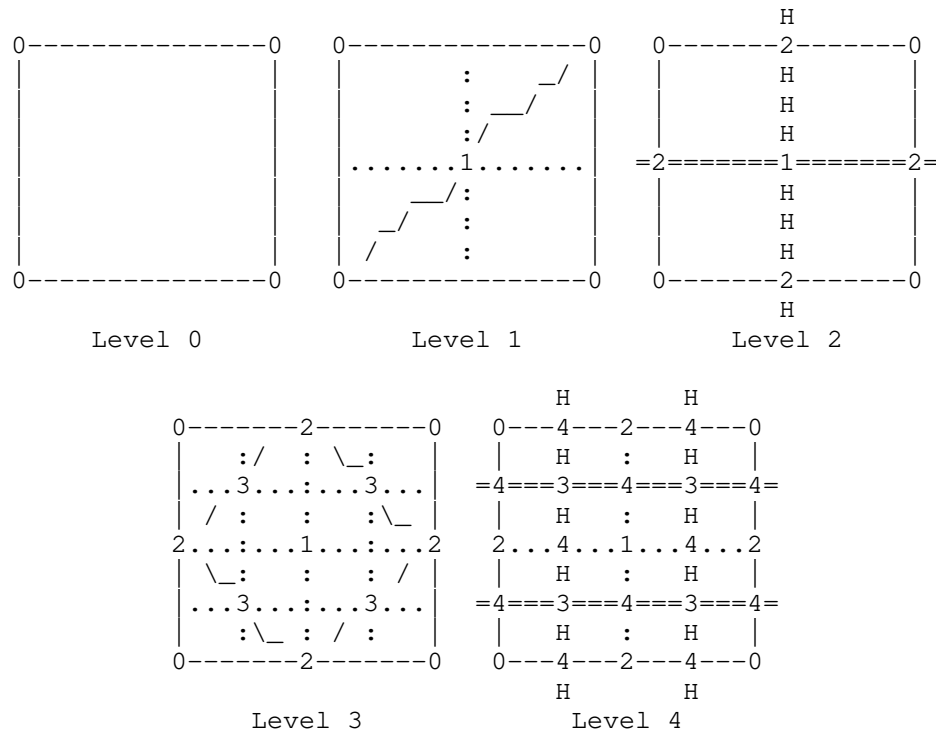
In a block with MVs at all four corners, we use normal bilinear weights to blend the predictions from each MV. The bilinear weights for each vertex,  $w[k]$ , at a pixel location  $(x, y)$  are defined as

$$\begin{aligned} w[0] &= (1 - x) * (1 - y) \\ w[1] &= x * (1 - y) \\ w[2] &= x * y \\ w[3] &= (1 - x) * y \end{aligned}$$

Let "I" be the reference image, and for simplicity denote the predictor  $I[x + mv[k].x, y + mv[k].y]$  for the pixel at location  $(x, y)$  with motion vector  $mv[k]$  as simply  $I(mv[k])$ . In a regular grid, with unique motion vectors defined at all four corners of a block, we predict the interior of the block using

$$I(mv[0]) * w[0] + I(mv[1]) * w[1] + I(mv[2]) * w[2] + I(mv[3]) * w[3]$$

In order to extend OBMC to handle variable block sizes while maintaining continuity along the edges, we start by imposing the restriction that the size of two adjacent blocks differ by at most a factor of two, which greatly simplifies the problem. To do this, we borrow a data structure from the related graphics problem of surface simplification, the semi-regular 4-8 mesh [VG00]. This is normally used to represent subdivisions in a triangle mesh, but we use it for variable-sized quadrilaterals.



The first four subdivision levels in a 4-8 mesh. Numbers indicate vertices with transmitted MVs. Diagonal lines (on odd levels) and double lines (on even levels) connect each new vertex to its two parents at the previous level (in some cases, this parent may lie in an adjacent block). Dotted lines indicate additional block boundaries.

Figure 2: Subdivision Levels in a 4-8 Mesh

Subdivision in a 4-8 mesh proceeds in two phases, as illustrated in Figure 2. In the first phase, a new vertex is added to the center of a quadrilateral. This subdivides the quadrilateral into four "quadrants", but does not add any additional vertices to the edges. Such edges are called "unsplit". In the second phase, each of the quadrilateral's edges may be split and connected to the center vertex, forming four new quadrilaterals. One useful property of this two-phase subdivision is that the number of vertices in the mesh merely doubles during each phase, instead of quadrupling as it would under normal quadtree subdivision. This provides more fine-grained control over the number of MVs transmitted.



To ensure that the size of two adjacent blocks differs by no more than a factor of two, we assign every vertex two parents in the mesh, which are the two adjacent vertices from the immediately preceding subdivision level. A vertex may not be added to the mesh until both of its parents are present. That is, a level 2 vertex may not be added to an edge until the blocks on either side have had a level 1 vertex added, and a level 3 vertex may not be added to the center of a block until both of the level 2 vertices have been added to its corners, and so on.

Therefore, we need only specify how to handle a block that has undergone phase one subdivision, but still has one or more unsplit edges, as illustrated in Figure 3. Such a block is divided into quadrants, and each is interpolated separately using a modified version of the previous bilinear weights.

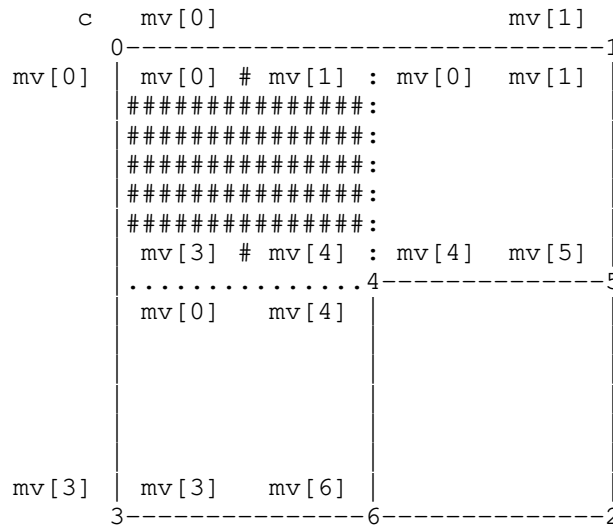


Figure 3: Interpolation Setup for Unsplit Edges

The same two MVs are used along the unsplit edge(s) as before, but we shift some of the weight used for blending from the middle of the edge to the exterior corner. More precisely, the weights  $w[k]$  are replaced by modified weights  $s[k]$ . For example, if  $c$  is the index of the vertex in the exterior corner,  $(+)$  denotes addition modulo four, and  $c (+) 1$  is the index of the corner bisecting the unsplit edge (the top edge in the figure), then

$$\begin{aligned} s[c] &= w[c] + 0.5*w[c (+) 1] \\ s[c (+) 1] &= 0.5*w[c (+) 1] \end{aligned}$$

The remaining weights are unchanged. A similar modification is used if it is  $c (+) 3$  that lies on the unsplit edge. The modifications are cumulative. That is, if both  $c (+) 1$  and  $c (+) 3$  lie on unsplit edges (as in the hashed region in Figure 3),

$$\begin{aligned}s[c] &= w[c] + 0.5*w[c (+) 1] + 0.5*w[c (+) 3] \\s[c (+) 1] &= 0.5*w[c (+) 1] \\s[c (+) 3] &= 0.5*w[c (+) 3] \\s[c (+) 2] &= w[c (+) 2]\end{aligned}$$

This definition of the blending weights clearly matches an adjacent block along an unsplit edge, regardless of whether or not that block has been split. Careful examination will verify that it also matches other quadrants along the interior edges. Each weight can be evaluated with finite differences at the cost of one add per pixel, plus setup overhead. The blending can be done with three multiplies per pixel by taking advantage of the fact that the weights sum to one, just as with regular bilinear interpolation.

The mesh itself may require more vertices than an unconstrained mesh to achieve a given level of subdivision in a local area, but requires fewer bits to encode the subdivision itself, simply because there are fewer admissible meshes. As long as a  $(0, 0)$  MV residual can be efficiently encoded, the worst-case rate of the 4-8 mesh should be close to that of a similar, unconstrained mesh.

This process can be extended to handle blocks that differ by more than one level of subdivision, so long as the edge between them remains entirely unsplit. For example, to handle block sizes that differ by a factor of four, instead of shifting half the blending weight from one vertex to the other, one simply needs to shift  $1/4$ ,  $1/2$ , or  $3/4$  of the weight, depending on the location of the block along the unsplit edge. However, the 4-8 mesh is no longer suitable for describing which vertices can appear in the mesh, and some modifications of the adaptive subdivision algorithm in Section 3.2 are required. We have not yet implemented these extensions.

### 3. Implementation and Motion Estimation

The algorithms in Section 2 have been implemented in the Daala video codec [Daala-website]. We use them to produce a complete "motion compensated reference frame", which is then lapped and transformed (in both the encoder and decoder) [I-D.egge-videocodec-tdlt] to make it available as a frequency-domain predictor for the transform stage [I-D.valin-netvc-pvq]. The full source code, including all of the OBMC work described in this draft is available in the project git repository at [1].

Luma blocks are square with sizes ranging from 32x32 to 4x4. The corners of the MV blocks are aligned with the corners of the transform blocks. An earlier design had the MV blocks offset from the transform blocks, so that MVs remained in the center of the transform blocks at the coarsest level, with an extra ring of implicit (0, 0) MVs around the frame (to keep the minimum number of transmitted MVs the same as with BMA). However, we found that there was essentially no performance difference between the two approaches (see commit 461310929fc5). Some things are simpler with the current approach (all of the special cases for the implicit (0, 0) MVs go away), and some things are more complicated, but most of the complications are confined to the computation of MV predictors.

The encoder performs rate-distortion (R-D) optimization during motion estimation to balance the prediction error (D) attainable against the number of bits required to achieve it (R), e.g., minimizing

$$J = D + \lambda R$$

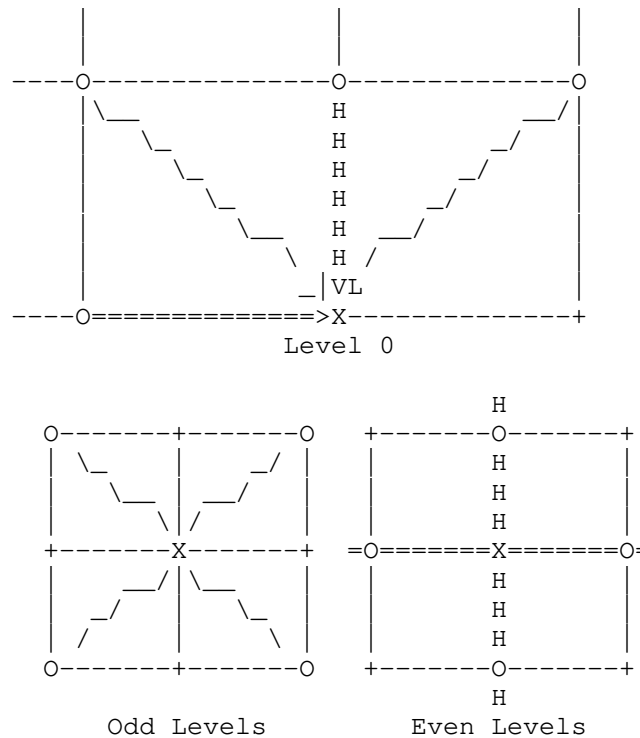
The value of  $\lambda$  is obtained directly from the target quantizer setting.

We use the Sum of Absolute Differences (SAD) in the luma plane as our distortion metric for the first three stages of the search, and use the Sum of Absolute Transformed Difference (SATD) during a final subpel refinement stage (with an appropriate adjustment to  $\lambda$ ).

We approximate the rate of small MV components (with a magnitude less than 3 after subtracting a predictor) using statistics from the previous frame, plus one sign bit for each non-zero component. Larger MV components have an additional non-adaptive rate cost added that increases logarithmically with the MV magnitude. The rate term for each vertex also includes a bit for each flag that indicates the presence of a child (2 per vertex on average). The real motion information is adaptively arithmetic encoded [I-D.terriberly-netvc-codingtools], but these approximations avoid having to update the rate term for every MV every time a single MV is changed.

We use a median-of-four predictor for almost every MV, as illustrated in Figure 4. The middle two values of each MV component are averaged, rounding towards even values. There are two exceptions. If an MV required for prediction lies outside the frame, a (0, 0) MV is substituted in its place. If an MV required for prediction lies inside a 32x32 block that comes after the current one in raster order, then that MV is ignored and we use the median of the three remaining MVs. This occurs when predicting MVs on even levels that lie on the right or bottom edges of a 32x32 block. MVs on the top

and left edges of the frame are considered to belong to the 32x32 block below or to the right, respectively (that is, the corresponding MV in that block is not ignored).



Key:

X - MV being predicted

O - MV used for prediction. Except at level 0, these are all ancestors of the MV being predicted, and thus are required to be present.

+ - MV grid point not used for prediction (might not be coded)

Figure 4: The Predictors Used for MVs at Each Level

The current bitstream encodes MVs level-by-level for the entire frame. It is expected at some point that this will be migrated to code all the MVs for a single 32x32 block at a time. This is the reason for excluding predictors outside of the current 32x32 block.

The number of combinations of subdivision levels and MVs available make finding a globally optimal set of parameters impractical. The problem of finding optimal subdivision levels alone is known to be NP-hard [AS94]. The estimation procedure outlined below attempts to

balance speed with compression performance, though it could certainly be improved with future research.

### 3.1. Initial Estimates

First we produce an initial MV estimate at each point in the fully-subdivided grid using BMA. We compute several MV candidates from spatial and temporal neighbors and assuming constant speed or acceleration. The candidates are grouped into sets by reliability and the search terminates early if the best candidate from a set has a SAD below a dynamically chosen threshold. Otherwise, a local gradient search is performed using a square pattern around the best candidate vector. The thresholds ensure extra computation time is spent only on blocks whose predictor can be reasonably expected to improve. Although we look solely at SAD to determine whether to continue the search, the candidates themselves are ranked using the full R-D cost metric,  $J$ .

Level 0 searches using (non-overlapped) 32x32 blocks centered on the corresponding grid points, while the next two levels use 16x16 blocks, the next two levels 8x8, and so on. MVs are estimated from the coarsest levels to the finest, to allow for the accurate computation of MV predictors used in the rate estimates. As the mesh is subdivided, existing grid points do not have their MVs re-estimated with smaller block sizes, even though the area that those MVs would influence in a grid subdivided to that level is reduced. All MVs are estimated only up to whole-pel accuracy at this stage.

### 3.2. Adaptive Subdivision

The second stage of motion estimation fixes the mesh subdivision. During this stage, the SAD for each block is computed using full OBMC instead of BMA. The MVs produced in the previous stage are held fixed in this one. Only the mesh subdivision level changes.

The extra subdivision required to add a vertex to the 4-8 mesh is similar to the implicit subdivision used by Zhang et al. in their variable block size OBMC scheme [ZAS98]. The difference is that we optimize over and encode such subdivision explicitly. We use a global R-D optimization strategy with general mesh decimations, as proposed by Balmeli [Bal01]. This is a greedy approach that starts with a full mesh and successively decimates vertices. Restricting decimation candidates to the leaves of the mesh can frequently produce sequences where decimating a MV (reducing rate) causes distortion to go down, clearly indicating that the previous rate allocation was not optimal. General mesh decimations, on the other hand, allow any MV to be removed at a given step, not just the leaves. If a non-leaf is decimated, all of its children are

decimated as well. This helps smooth out non-monotonicities in the distortion measure during the decimation process, especially at low rates

The following notation is used to describe the algorithm. The current mesh is denoted by  $M$ , and  $M_v$  is the "merging domain" of  $v$  in  $M$ : the set of vertices in  $M$  that must be removed to remove  $v$ . This includes  $v$  and all of its undecimated children. Additionally, the variation  $dU(M_v)$  contains the pairs  $(dD(M_v), dR(M_v))$ : the change in distortion (SAD) and rate (bits) caused by removing  $M_v$  from  $M$ . We also refer to the change in SAD in a single block  $b$  caused by removing a single vertex  $v$  as  $dD_b(v)$ . Finally,  $A_v$  is the set of ancestors of  $v$  in  $M$ . Some minor additions to Balmelli's original algorithm are made to handle the fact that distortion is measured over squares, not triangles. The steps of the algorithm are:

1. For all  $v$ , compute  $dU(M_v)$ .
2. Do
  - (a) Let  $v^*$  be the value of  $v$  in  $M$  for which  $-dD(M_v)/dR(M_v)$  is the smallest.
  - (b) If  $-dD(M_{v^*})/dR(M_{v^*}) > \lambda$ , stop.
  - (c) For all  $w$  in  $M_{v^*}$ , sorted by depth from deepest to shallowest:
    - i. For all  $a$  in  $A_w$ , subtract  $dU(M_w)$  from  $dU(M_a)$ .
    - ii. Remove  $w$  from the mesh.
    - iii. If  $w$  was on an even level, then for each adjacent block  $b$  with a  $w'$  in  $M$  such that  $w'$  lies on the same level as  $w$ :
      - A. Let  $d$  be change in  $dD_b(w')$  before and after decimating  $w$ .
      - B. For all  $w$  in  $\{w'\} \cup A_{w'} \setminus A_w$ , add  $d$  to  $dD(M_a)$ .

These steps ensure that  $dU(M_v)$  contains the up-to-date changes in the global rate and distortion after each merging domain is decimated. This update process properly accounts for overlapping merging domains due to an inclusion-exclusion principle. See Balmelli for details [Bal01]. Step 2(c)iii handles the case of decimating one corner of a block,  $w$ , when the opposite corner,  $w'$ , remains. This changes  $dD_b(w')$ , the cost of decimating the opposite

corner, and that change must be propagated to each merging domain to which  $w'$  belongs. No change needs to be made to the common ancestors of  $w$  and  $w'$  however: once  $dD(M_{w'})$  is updated, the normal update process that will be executed when  $w'$  is decimated is sufficient. The addition of these extra steps does not affect the computational complexity of the algorithm which is  $\Theta(n \log n)$ , where  $n$  is the size of the initial mesh.

The distortion measurements needed to initialize and update  $dU(M_v)$  can be computed once, in advance, by computing the SAD value of each block in all sizes and with all possible combinations of unsplit edges. All told, each pixel in the image is used in exactly 13 SAD computations (one for the largest block size, with no unsplit edges, and for each additional block size). Also, since the mesh only undergoes six levels of subdivision, there are only a small number of unique merging domains and ancestor sets. These can be computed offline and stored in tables to simplify the decimation process. To compute the set difference  $A_{w'} \setminus A_w$ , we note that  $w$  and  $w'$  share a single common parent,  $p$ . The common ancestors of  $w$  and  $w'$  are now formed by the set  $\{p\} \cup A_p$ , meaning one can add  $d$  to the nodes in  $A_{w'}$  and then subtract it from the nodes in  $\{p\} \cup A_p$  to effect the set difference in Step 2(c)iiiB. Alternatively, one could simply use a larger set of lookup tables.

### 3.3. Iterative Refinement

The next stage uses the iterated dynamic programming (DP) proposed by Chen and Willson to refine the MVs, accounting for their interdependencies [CW00]. In this scheme, a single row (resp. column) of MVs is optimized at a time using a Viterbi trellis [For73], while the rest remain fixed. If there is no direct block edge between two consecutive MVs in a row (resp. column) then the trellis stops, and a new one is started. This continues until the entire row (resp. column) has been examined. The process is then repeated until the total change in Lagrangian cost,  $J$ , falls below a given threshold.

#### 3.3.1. Rate and Distortion Changes

We use the change in rate and distortion to compute the cost of each path in the trellis. A single MV can influence the distortion of as many as 12 neighboring blocks. Only the ones to the left (resp. above) are added to the current cost of each path. When the following MV is chosen, an additional 2 to 8 blocks may be added. If necessary, the blocks to the right (resp. below) are added after the last MV in the trellis.

Unfortunately, the rate of a MV depends on the values of the MVs used to predict it. Chen and Willson assume MVs use 1-D differential coding, as in MPEG-1. With our prediction scheme, several (not necessarily consecutive) MVs on the DP path may be used to predict a given MV, and the corresponding change in rate is not known until a MV has been chosen for all of them.

If we were to consider all possible combinations of candidates for the predictors, the number of trellis edges would increase by several orders of magnitude. This seems excessively wasteful, since as long as the changes to the MVs are small, the median operation ensures only one or two of them are likely to have any influence on the predicted value in the first place. Instead, we immediately compute the rate change in each predicted vector---excluding those that themselves lie further along the DP path, since we do not yet know what MV will be encoded. We do this assuming all MVs not already considered by the DP remain fixed, and add the change to the cost of the current path. If changing a subsequent MV on the path causes the rate of one of these predicted MVs to change again, the new rate change is used from then on.

Because we essentially discard a large number of trellis states of limited utility, we might theoretically discard the path that does not change any MVs, even though its true cost is lower than the ones we keep. Thus, as a safety precaution, we check the final cost of the best path, and do not apply it if it is greater than zero. This does occur in practice, but very rarely.

Other, simpler alternatives to this approach are also possible. For example, we tried only considering rate changes for MVs on the actual DP path, which is much like Chen and Willson's approach. However, on frames with complex motion, we have seen dramatic improvements in visual quality and motion field smoothness by properly accounting for all rate changes. This is because a level 0 MV, for example, may be used to predict up to 24 other MVs, only 8 of which lie on a given DP path. In a dense mesh, the rate changes off the path may dominate the ones on it.

### 3.3.2. Complexity Reduction

Chen and Willson showed that using a logarithmic search instead of an exhaustive one for the DP resulted in an average PSNR loss of only 0.05 dB and an average MV bitrate increase of 55 bits per frame. We take an even more aggressive approach, and replace the logarithmic search with a diamond search. Because the complexity of a given DP chain increases quadratically in the number of MV candidates at each node, reducing the candidate count can give a substantial performance boost.



The logarithmic search uses a 9-candidate square pattern in each stage. The displacements used in the pattern shrink by a factor of two in each phase. Chen and Willson used a 3-phase search to achieve an effective search radius of  $\pm 7 \times 7$  pixels. In our framework, this requires examining  $3 \times 9 \times 2 = 243$  trellis edges per MV for one full iteration. However, the large displacement patterns only alter a small number of MVs that have usually been grossly mis-estimated. They are even likely to cause further mis-estimation in areas with repeated structure or lack of texture, which makes further refinement less effective.

One alternative is to simply discard them, and only perform the square pattern search with one-pixel displacements. The chance of being trapped in a local minimum is increased, but three times as many iterations can be performed in the same amount of time. On the other hand, a small diamond search pattern has only 5 candidates, making it even more attractive. This allows more than nine times as many iterations as the full logarithmic search in the same amount of time. We support using the diamond search pattern, the square search pattern, and the square search pattern with logarithmic step sizes with various complexity settings, but by default run with only the diamond pattern with a single step size. The logarithmic square pattern search saves between 0.6% and 1.2% rate on metrics, but adds 50% to the total encode time.

The computational complexity of this iterative refinement is still relatively high. In a single iteration, each of the four edges of a block are traversed exactly once by a DP path, during which its SAD is evaluated 25 times, for a total of 100 SAD calculations per block. This is nearly as many as full search BMA with a  $\pm 6 \times 6$  window, and computing our blended predictors already has higher complexity. Thus it is not suitable for a real-time implementation, but it can easily be disabled, or even lighter-weight versions designed.

### 3.3.3. Subpel Refinement

The same small diamond-pattern search can be used to refine the motion vectors to subpel precision. A square pattern is also supported at the highest complexity level, and saves an additional 0.5% to 0.7% on bitrate, but is half the speed of the default settings. Our implementation supports half-, quarter-, or eighth-pel resolution MVs. First, the DP process is iterated with the small diamond and half-pel displacements until the change in Lagrangian cost,  $J$ , for an iteration falls below a given threshold.

Finer resolutions are only used if they provide an overall R-D benefit, which is tested on a frame-by-frame basis. First, iteration is done with quarter-pel displacements, followed, if successful, by

eighth-pel. If the decrease in SAD from the finer resolution MVs cannot balance the (approximately) 2 bit per MV increase in bitrate, then the coarser vectors are used instead.

Subpel interpolation is performed using a separable 6-tap polyphase filter bank. Only eight filters are currently used, one for each subpel offset at eighth-pel resolution. If chroma is decimated (for 4:2:0 video) and eighth-pel MVs are used, then the MV is divided by two and rounded to the nearest even value to select an appropriate subpel filter.

#### 4. References

##### 4.1. Informative References

- [I-D.egge-videocodec-tdlt]  
Egge, N. and T. Terriberry, "Time Domain Lapped Transforms for Video Coding", draft-egge-videocodec-tdlt-01 (work in progress), March 2015.
- [I-D.terriberry-netvc-codingtools]  
Terriberry, T., "Coding Tools for a Next Generation Video Codec", draft-terriberry-netvc-codingtools-00 (work in progress), June 2015.
- [I-D.valin-netvc-pvq]  
Valin, J., "Pyramid Vector Quantization for Video Coding", draft-valin-netvc-pvq-00 (work in progress), June 2015.
- [AS94] Agarwal, P. and S. Suri, "Surface Approximation and Geometric Partitions", Proc. of the 5th ACM-SIAM Symposium on Discrete Algorithms (SODA'94) pp. 24--33, January 1994.
- [Bal01] Balmelli, L., "Rate-Distortion Optimal Mesh Simplification for Communications", PhD thesis, Ecole Polytechnique Federale de Lausanne, Switzerland, 2001.
- [CW00] Chen, M. and A. Willson, "Motion-Vector Optimization of Control Grid Interpolation and Overlapped Block Motion Compensation Using Iterated Dynamic Programming", IEEE Transactions on Image Processing 9(7):1145--1157, July 2000.
- [For73] Forney, G., "The Viterbi Algorithm", Proceedings of the IEEE 61(3):268--278, March 1973.

- [KO95] Katto, J. and M. Ohta, "An Analytical Framework for Overlapped Motion Compensation", Proc. of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP'95) vol. 4, pp. 2189--2192, May 1995.
- [KK97] Kuo, T. and C. Kuo, "A Hybrid BMC/OBMC Motion Compensation Scheme", Proc. of the International Conference on Image Processing (ICIP'97) vol. 2, pp. 795--798, October 1997.
- [Lee95] Lee, J., "Optimal Quadtree for Variable Block Size Motion Estimation", Proc. of the IEEE International Conference on Image Processing vol. 3, pp. 480--483, October 1995.
- [VG00] Velho, L. and J. Gomes, "Variable Resolution 4-k Meshes: Concepts and Applications", Computer Graphics Forum 19(4):195--212, December 2000.
- [ZAS98] Zhang, J., Ahamd, M., and M. Swamy, "New Windowing Techinques for Variable-Size Block Motion Compensation", IEE Proceedings---Vision, Image, and Signal Processing 145(6):399--407, December 1998.
- [ZSNKI02] Zhen, W., Shishikui, Y., Naemure, M., Kanatsugu, Y., and S. Itoh, "Analysis of Space-Dependent Characteristics of Motion-Compensated Frame Differences Based on a Statistical Motion Distribution Model", IEEE Transactions on Image Processing 11(4):377--386, April 2002.
- [Daala-website] "Daala website", Xiph.Org Foundation , <<https://xiph.org/daala/>>.

#### 4.2. URIs

- [1] <https://git.xiph.org/daala.git>

#### Author's Address

Timothy B. Terriberry  
Mozilla Corporation  
331 E. Evelyn Avenue  
Mountain View, CA 94041  
USA

Phone: +1 650 903-0800  
Email: [tterribe@xiph.org](mailto:tterribe@xiph.org)

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: December 11, 2015

JM. Valin  
Mozilla  
June 9, 2015

Pyramid Vector Quantization for Video Coding  
draft-valin-netvc-pvq-00

Abstract

This proposes applying pyramid vector quantization (PVQ) to video coding.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 11, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may not be modified, and derivative works of it may not be created, and it may not be published except as an Internet-Draft.

## Table of Contents

1. Introduction . . . . .	2
2. Gain-Shape Coding and Activity Masking . . . . .	2
3. Householder Reflection . . . . .	3
4. Angle-Based Encoding . . . . .	4
5. Bi-prediction . . . . .	5
6. Coefficient coding . . . . .	6
7. Development Repository . . . . .	6
8. IANA Considerations . . . . .	6
9. Security Considerations . . . . .	6
10. Acknowledgements . . . . .	7
11. Informative References . . . . .	7
Author's Address . . . . .	7

## 1. Introduction

This draft describes a proposal for adapting the Opus RFC 6716 [RFC6716] energy conservation principle to video coding based on a pyramid vector quantizer (PVQ) [Pyramid-VQ]. One potential advantage of conserving energy of the AC coefficients in video coding is preserving textures rather than low-passing them. Also, by introducing a fixed-resolution PVQ-type quantizer, we automatically gain a simple activity masking model.

The main challenge of adapting this scheme to video is that we have a good prediction (the reference frame), so we are essentially starting from a point that is already on the PVQ hyper-sphere, rather than at the origin like in CELT. Other challenges are the introduction of a quantization matrix and the fact that we want the reference (motion predicted) data to perfectly correspond to one of the entries in our codebook. This proposal is described in greater details in [Perceptual-VQ], as well as in demo [PVQ-demo].

## 2. Gain-Shape Coding and Activity Masking

The main idea behind the proposed video coding scheme is to code groups of DCT coefficient as a scalar gain and a unit-norm "shape" vector. A block's AC coefficients may all be part of the same group, or may be divided by frequency (e.g. by octave) and/or by directionality (horizontal vs vertical).

It is desirable for a single quality parameter to control the resolution of both the gain and the shape. Ideally, that quality parameter should also take into account activity masking, that is, the fact that the eye is less sensitive to regions of an image that have more details. According to Jason Garrett-Glaser, the perceptual analysis in the x264 encoder uses a resolution proportional to the

variance of the AC coefficients raised to the power  $a$ , with  $a=0.173$ . For gain-shape quantization, this is equivalent to using a resolution of  $g^{(2a)}$ , where  $g$  is the gain. We can derive a scalar quantizer that follows this resolution:

$$g = Q_g \gamma^{\frac{b}{2a}},$$

where  $\gamma$  is the gain quantization index,  $b=1/(1-2a)$  and  $Q_g$  is the gain resolution and main quality parameter.

An important aspect of the current proposal is the use of prediction. In the case of the gain, there is usually a significant correlation with the gain of neighboring blocks. One way to predict the gain of a block is to compute the gain of the coefficients obtained through intra or inter prediction. Another way is to use the encoded gain of the neighboring blocks to explicitly predict the gain of the current block.

### 3. Householder Reflection

Let vector  $x_d$  denote the (pre-normalization) DCT band to be coded in the current block and vector  $r_d$  denote the corresponding reference (based on intra prediction or motion compensation), the encoder computes and encodes the "band gain"  $g = \sqrt{x_d^T x_d}$ . The normalized band is computed as

$$x = \frac{x_d}{\|x_d\|},$$

with the normalized reference vector  $r$  similarly computed based on  $r_d$ . The encoder then finds the position and sign of the largest component in vector  $r$ :

$$\begin{aligned} m &= \operatorname{argmax}_i |r_i| \\ s &= \operatorname{sign}(r_m) \end{aligned}$$

and computes the Householder reflection that reflects  $r$  to  $-s e_m$ , where  $e_m$  is a unit vector that points in the direction of dimension  $m$ . The reflection vector is given by

$$v = r + s e_m.$$

The encoder reflects the normalized band to find the unit-norm vector

$$z = x - 2 \frac{v^T x}{v^T v} v .$$

The closer the current band is from the reference band, the closer  $z$  is from  $-s e_m$ . This can be represented either as an angle, or as a coordinate on a projected pyramid.

#### 4. Angle-Based Encoding

Assuming no quantization, the similarity can be represented by the angle

$$\theta = \arccos(-s z_m) .$$

If  $\theta$  is quantized and transmitted to the decoder, then  $z$  can be reconstructed as

$$z = -s \cos(\theta) e_m + \sin(\theta) z_r ,$$

where  $z_r$  is a unit vector based on  $z$  that excludes dimension  $m$ .

The vector  $z_r$  can be quantized using PVQ. Let  $y$  be a vector of integers that satisfies

$$\sum_i (|y[i]|) = K ,$$

with  $K$  determined in advance, then the PVQ search finds the vector  $y$  that maximizes  $y^T z_r / (y^T y)$ . The quantized version of  $z_r$  is

$$z_{rq} = \frac{y}{\|y\|} .$$

If we assume that MSE is a good criterion for optimizing the resolution, then the angle quantization resolution should be (roughly)

$$Q_{\theta} = \frac{dg}{d(\gamma)} \frac{1}{g} = \frac{b}{\gamma} .$$

To derive the optimal  $K$  we need to consider the normalized distortion for a Laplace-distributed variable found experimentally to be approximately

$$D_p = \frac{(N-1)^2 + C(N-1)}{24K^2},$$

with  $C \approx 4.2$ . The distortion due to the gain is

$$D_g = \frac{b^2 Q_g^2 \gamma^{(2b-2)}}{12}.$$

Since PVQ codes  $N-2$  degrees of freedom, its distortion should also be  $(N-2)$  times the gain distortion, which eventually leads us to the optimal number of pulses

$$K = \frac{\gamma \sin(\theta)}{b} \sqrt{\frac{N + C - 2}{2}}.$$

The value of  $K$  does not need to be coded because all the variables it depends on are known to the decoder. However, because  $Q_\theta$  depends on the gain, this can lead to unacceptable loss propagation behavior in the case where inter prediction is used for the gain. This problem can be worked around by making the approximation  $\sin(\theta) \approx \theta$ . With this approximation, then  $K$  depends only on the  $\theta$  quantization index, with no dependency on the gain. Alternatively, instead of quantizing  $\theta$ , we can quantize  $\sin(\theta)$  which also removes the dependency on the gain. In the general case, we quantize  $f(\theta)$  and then assume that  $\sin(\theta) \approx f(\theta)$ . A possible choice of  $f(\theta)$  is a quadratic function of the form:

$$f(\theta) = a_1 \theta - a_2 \theta^2.$$

where  $a_1$  and  $a_2$  are two constants satisfying the constraint that  $f(\pi/2) = \pi/2$ . The value of  $f(\theta)$  can also be predicted, but in case where we care about error propagation, it should only be predicted from information coded in the current frame.

## 5. Bi-prediction

We can use this scheme for bi-prediction by introducing a second  $\theta$  parameter. For the case of two (normalized) reference frames  $r_1$  and  $r_2$ , we introduce  $s_1 = (r_1 + r_2)/2$  and  $s_2 = (r_1 - r_2)/2$ . We start by using  $s_1$  as a reference, apply the Householder reflection to both  $x$  and  $s_2$ , and evaluate  $\theta_1$ . From there, we derive a second Householder reflection from the reflected version of  $s_2$  and apply it to  $z$ . The result is that the  $\theta_2$  parameter controls how the



current image compares to the two reference images. It should even be possible to use this in the case of fades, using two references that are before the frame being encoded.

## 6. Coefficient coding

Encoding coefficients quantized with PVQ differs from encoding scalar-quantized coefficients from the fact that the sum of the coefficients magnitude is known (equal to K). It is possible to take advantage of the known K value either through modeling the distribution of coefficient magnitude or by modeling the zero runs. In the case of magnitude modeling, the expectation of the magnitude of coefficient n is modeled as

$$E(|y_n|) = \alpha * \frac{K_n}{N - n} ,$$

where  $K_n$  is the number of pulses left after encoding coefficients from 0 to n-1 and alpha depends on the distribution of the coefficients. For run-length modeling, the expectation of the position of the next non-zero coefficient is given by

$$E(|run|) = \beta * \frac{N - n}{K_n} ,$$

where beta also models the coefficient distribution.

## 7. Development Repository

The algorithms in this proposal are being developed as part of Xiph.Org's Daala project. The code is available in the Daala git repository at <<https://git.xiph.org/daala.git>>. See <<https://xiph.org/daala/>> for more information.

## 8. IANA Considerations

This document makes no request of IANA.

## 9. Security Considerations

This draft has no security considerations.

## 10. Acknowledgements

Thanks to Jason Garrett-Glaser, Timothy Terriberry, Greg Maxwell, and Nathan Egge for their contribution to this document.

## 11. Informative References

## [Perceptual-VQ]

Valin, JM. and TB. Terriberry, "Perceptual Vector Quantization for Video Coding", Proceedings of SPIE Visual Information Processing and Communication , February 2015, <[http://jmvalin.ca/papers/spie\\_pvq.pdf](http://jmvalin.ca/papers/spie_pvq.pdf)>.

## [PVQ-demo]

Valin, JM., "Daala: Perceptual Vector Quantization (PVQ)", November 2014, <[https://people.xiph.org/~jm/daala/pvq\\_demo/](https://people.xiph.org/~jm/daala/pvq_demo/)>.

## [Pyramid-VQ]

Fischer, T., "A Pyramid Vector Quantizer", IEEE Trans. on Information Theory, Vol. 32 pp. 568-583, July 1986.

[RFC6716] Valin, JM., Vos, K., and T. Terriberry, "Definition of the Opus Audio Codec", RFC 6716, September 2012.

## Author's Address

Jean-Marc Valin  
Mozilla  
331 E. Evelyn Avenue  
Mountain View, CA 94041  
USA

Email: [jmvalin@jmvalin.ca](mailto:jmvalin@jmvalin.ca)