Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 17, 2016

T. Davies
Cisco

March 16, 2016

Quantisation matrices for Thor video coding
draft-davies-netvc-qmtx-00

Abstract

This draft describes a family of default quantisation matrices that may be used to improve perceptual quality when encoding with Thor. Similar quantisation matrix designs may be used in most block-based video and image codecs.

Table of Contents

1. Introduction 2

2. Definitions 2
2.1. Terminology 2

3. Quantisation matrix design 3
3.1. The function of quantisation matrices 3
3.2. Quantisation matrices in AVC and HEVC 4
3.3. Quantisation matrices in Thor 4
3.4. Implementation 5

4. Compression performance 6

5. Informative References 7

Author's Address 7

1. Introduction

This document describes a family of default quantisation matrices that may be used to improve perceptual quality when encoding with Thor. The quantisation matrices are designed to be near-flat at high quantisation levels and more strongly profiled at low quantisation levels, to avoid ringing artefacts and better shape quantisation error across a whole sequence with varying quantisation levels.

2. Definitions

2.1. Terminology

This document uses the following terms.

QP: quantisation parameter

QM: quantisation matrix

CSF: contrast sensitivity function

BDR: Bjontegaard Delta-Rate

3. Quantisation matrix design

3.1. The function of quantisation matrices

Quantisation matrices work by shaping the residual error after quantisation in the spatial frequency domain, usually the DCT domain. This is done by varying the quantisation factor applied across spatial frequencies in the transform block. Typically a high quantisation factor is applied at high spatial frequencies and a low one at low spatial frequencies.

The aim is roughly to match a Contrast Sensitivity Function for the human visual system. This provides a curve of sensitivity to detail (and therefore coding errors) with spatial frequency. Given known resolutions and assumed viewing distances, a weighting function can be simply defined for all the coefficients in a transform block.

This simple approach is complicated, however, by a number of factors. The first is the CSF is in reality not a simple function of spatial frequency, but depends on factors such as brightness which are imperfectly corrected for by television gammas. There is little that can be done about that in the quantisation matrices themselves, but adjusting QP itself may help.

The second factor is that CSFs are determined experimentally based on models of Just Noticeable Difference (JND) and do not reflect so well the impact of distortions well above this level. Adjustments at high levels of quantisation are needed to reflect this.

Finally, applying quantisation matrices to video is affected by the fact that most frames are predicted and the QM is applied to the residual after prediction. This means that the quantisation error for a block consists of the quantisation error in the reference block, plus any additional error introduced in the current block. These errors will add if they are uncorrelated, but they may well be correlated at high QP.

Despite these difficulties, QMs are widely used and known to work well, and are available in video coding standards such as H264/AVC and H265/HEVC [AVC,HEVC].

3.2 Quantisation matrix design in AVC and HEVC

Quantisation matrices are available in a number of different codecs. The design in AVC and HEVC is to provide default matrices together with the ability to signal bespoke matrices [AVC,HEVC]. These matrices must cover all the different transform block sizes, components (Y, Cb, Cr) and intra and inter frame or block types, with fall-backs defined if bespoke matrices are not provided. Default inter block matrices are flatter than intra matrices, no doubt because of the noise-addition effect described in section 3.1: if they had the same profile as for intra then the overall profile of the combined prediction + residual could be over-shaped.

3.3 Design of quantisation matrices in Thor

Thor provides a set of matrices for each component of 420-sampled video, for each block size and each quantisation parameter. The principles behind the design are as follows:

1) QP dependence. Matrices become flatter as quantisation levels increase

2) Energy preservation for intra. The inverse quantisation matrices for intra blocks are normalised to approximately preserve energy of the residual

3) DC preservation for inter. The inverse quantisation matrices for inter blocks are normalised to preserve the DC level

4) Matrices are also flatter for inter blocks than for intra blocks.

5) Quantisation matrix strength is globally adjustable

The QP dependence takes account of a number of factors. Firstly it reflects that inter blocks typically have higher QPs than the blocks used to predict them. This means that flattening the matrices at higher QP naturally prevents over-shaping the quantisation error.

Secondly, the high-QP flattening process also reflects the fact that errors at this level are very visible even at high spatial frequencies. Strong error-shaping at these QP levels leads to very visible additional ringiness.

SSIM-based metrics [SSIM,MSSSIM,FASTSSIM] indicate that preserving image variances and therefore residual energies is perceptually important. This is feasible for intra where residuals are substantial but in the case of inter it is also important to preserve DC levels since getting these wrong can produce very visible artefacts.

Intra frames tend to have lower QP than inter frames, and this means that QP dependence absorbs most of the requirement for inter matrices to be flatter than intra matrices. However inter matrices are still a little flatter, to take account of the different characteristics of intra and inter blocks within the same frame.

In determining the quantisation matrix, there are 12 possible sets available giving a new set of matrix for each change of approximately 4 in quantisation value. Thor also supports a global adjustment or strength parameter, which offsets the LUT mapping quantisation parameter to quantisation matrix set. This is a value from -32 to 31. A value of -32 will reduce the qp used by 32, increasing the strength of quantisation matrix dramatically. Likewise a value of 31 will eliminate quantisation matrices for all but the smallest QPs.

The effect of the ability to signal strength, and the provision of a range of QP-dependent matrices are intended to remove the need to signal bespoke matrices at all.

3.4 Implementation

Quantisation matrices are applied as multiplicative factors in forward or inverse quantisation processes. In Thor the basic unweighted dequantisation process for a coefficient c with quantisation parameter q is based on two values: scale[q], which depends only on q%6, and shift[q] which depends only on q/6, the block size and the signal dynamic range. scale[q] takes care of quantisation step sizes which fall between powers of 2 and shift[q] takes care of the basic power of 2 part of the quantisation step.

The formula for unweighted dequantisation is then:

```
c -> (c*scale[q] + (1<<(shift[q]-1))) >> shift[q]                    (1)
```

for positive shift[q], otherwise

```
c -> (c*scale[q])<<(-shift[q])                                       (2)
```

To apply a matrix M to a coefficient c[i,j] at position (i,j) within a block, the formulae (1), (2) change to:

```
c[i,j]->(c[i,j]*M[i,j]*scale[q]+(1<<(shift[q]+5)))>>(shift[q]+6)     (3)
```

if shift[q]+6 > 0, otherwise

```
c[i,j]->(c[i,j]*M[i,j]*scale[q])<<(-shift[q]-6)                      (4)
```

otherwise.

Exactly complementary formulae can be derived for the forward quantisation process.

4. Compression performance

Although largely a visual tool, the effectiveness of QMs can be inferred by changes to PSNRHVS [PSNRHVS] and FASTSSIM metrics. FASTSSIM tends to over-estimate gains a little, as it has a bias towards low-pass filtering. Overall BDR results for the Low-Delay B (LDB) and High-Delay B GOP 16 configuration (HDB16) are as follows (QPs 22, 27, 32, 37):

Config	PSNR	PSNRHVS	FASTSSIM
LDB	+1.1%	-3.3%	-9.0%
HDB	+2.2%	-2.6%	-11.6%

These were computed on the same test sequences as in IRFVC.

FASTSSIM and PSNRHVS gains are typically larger, and PSNR losses smaller, for higher resolution material.

5. Informative References

[AVC] ITU-T Recommendation H.264, "Advanced video coding for generic audiovisual services", March 2010.

[HEVC] ITU-T Recommendation H.265, "High efficiency video coding", April 2013.

[FASTSSIM] Chen, M. and A. Bovik, "Fast structural similarity index algorithm", 2010, <http://live.ece.utexas.edu/publications/2011/chen_rtip_2011.pdf>.

[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>.

[IRFVC] Davies, T. "Interpolated reference frames for video coding", IETF draft https://www.ietf.org/id/draft-davies-netvc-irfvc-00.txt

Author's Address:

Thomas Davies
Cisco
Feltham
UK

Email: thdavies@cisco.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

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.

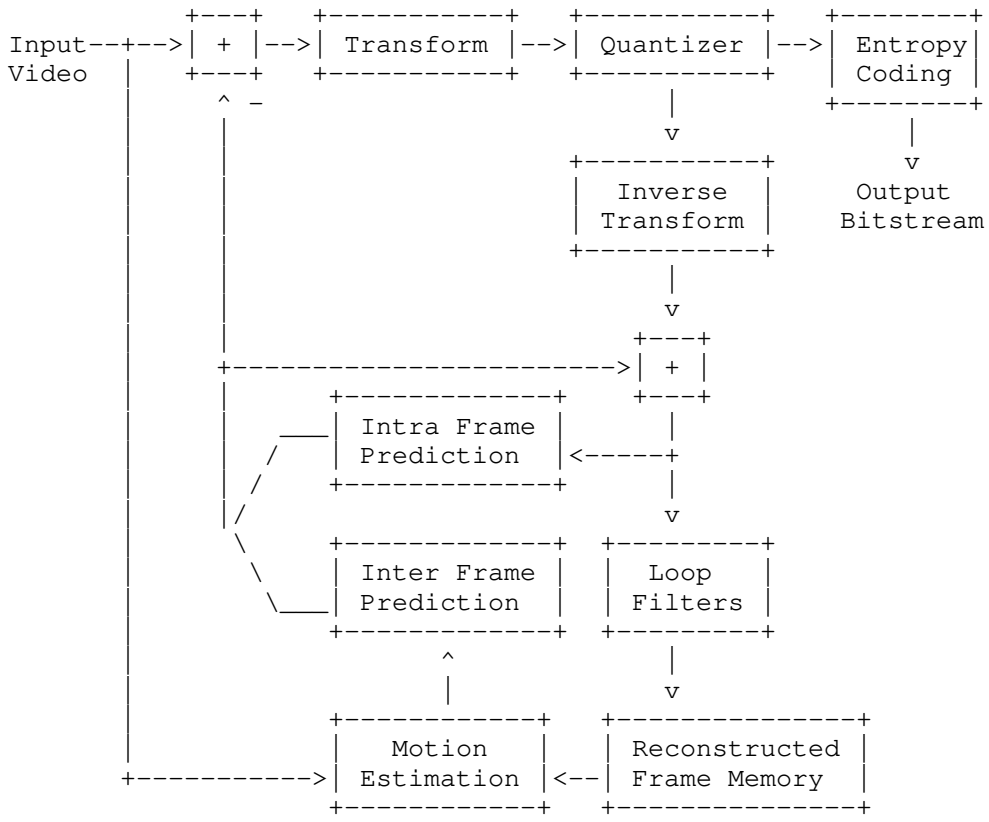


Figure 1: Encoder Structure

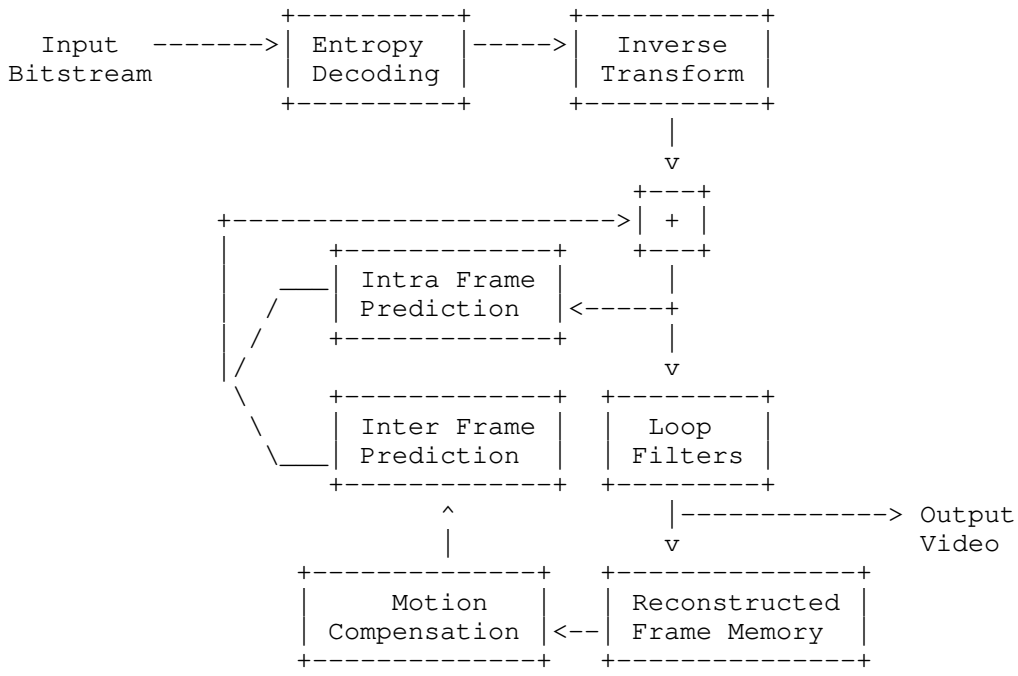


Figure 2: Decoder Structure

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.

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 if 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:

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

- A rectangular inter0 block and
- Further split.

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

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

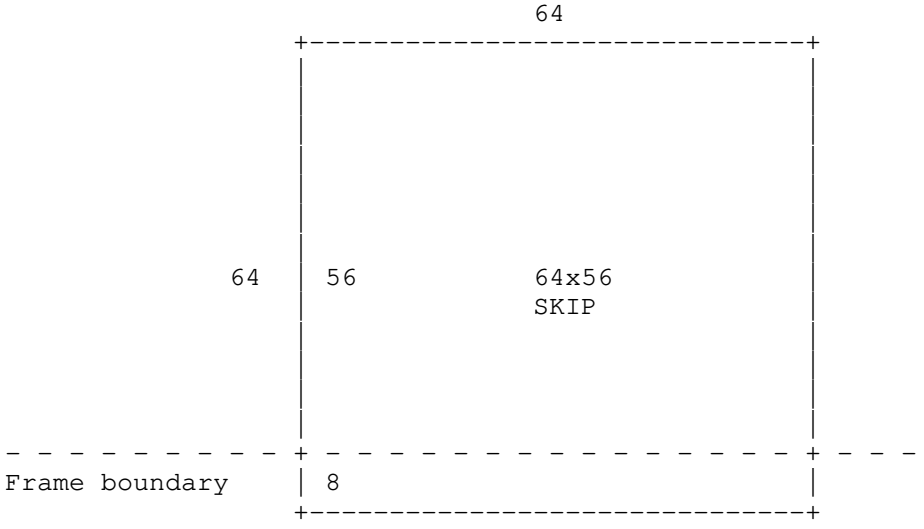


Figure 3: Super block at frame boundary

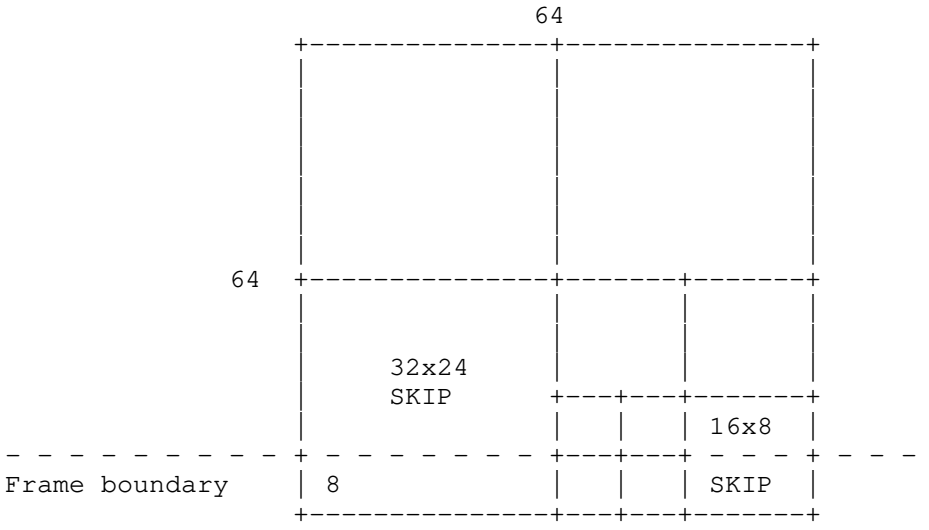


Figure 4: Coding block at frame boundary

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.

- 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.
- In the frame header, signal which of these frames shall be active for the current frame.
- 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. i0,0 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 $j = -2,...,3$

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

The minimum sub-block size is 8x8.

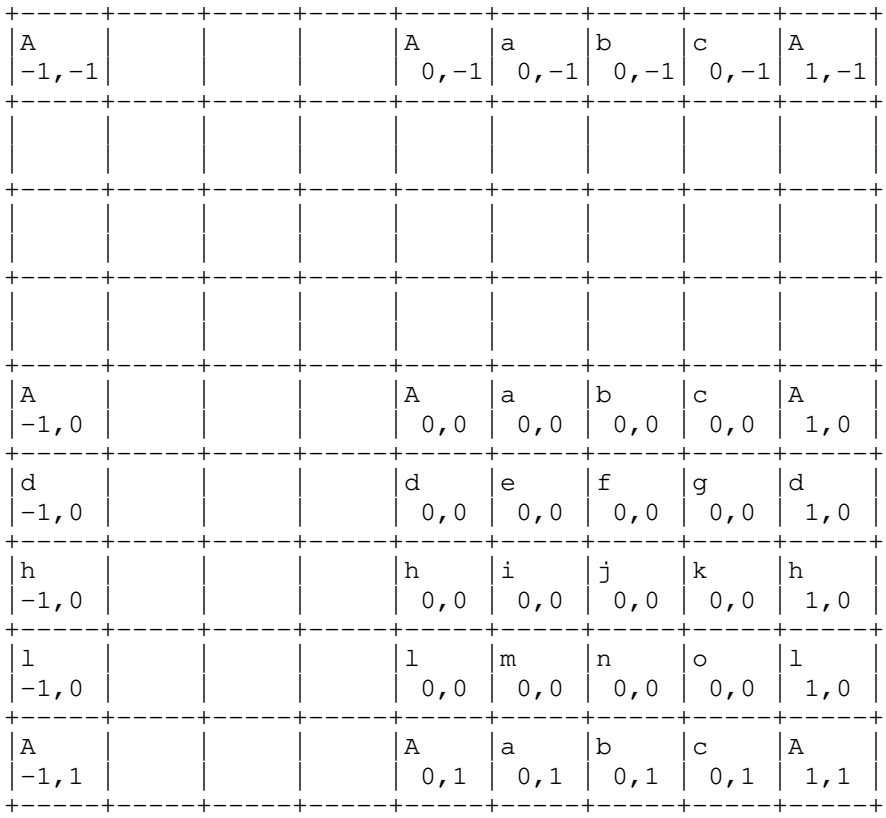


Figure 5: Sub-pixel positions

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*A-1,-1 + 1*A0,-1 + 1*A1,-1 + 0*A2,-1 +

1*A-1,0 + 2*A0,0 + 2*A1,0 + 1*A2,0 +

1*A-1,1 + 2*A0,1 + 2*A1,1 + 1*A2,1 +

0*A-1,2 + 1*A0,2 + 1*A1,2 + 0*A2,2 + 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.

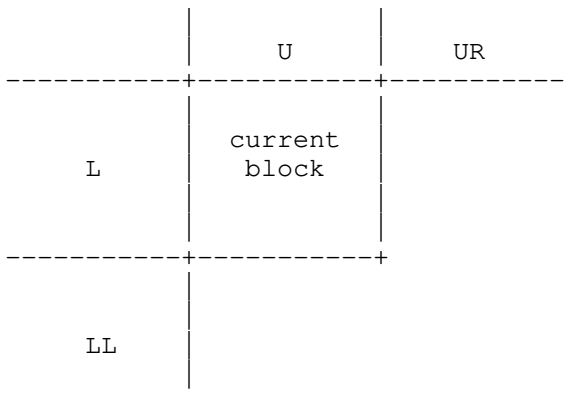


Figure 6: Availability of neighbor blocks

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.

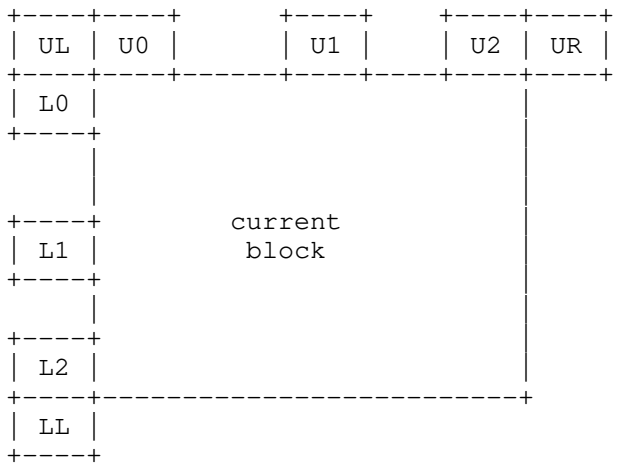


Figure 7: Motion vector candidates

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 = abs(a-b) + 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 < beta(QP)

   * The edge is also a transform block edge

   * abs(mvx(left)) > 2, or abs(mvx(right)) > 2, or

     abs(mvy(left)) > 2, or abs(mvy(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:

   delta = clip((18*(c-b) - 6*(d-a) + 16)/32,tc,-tc),

   where tc is a QP-dependent value.

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

a' = a + delta/2

b' = b + delta

c' = c + delta

d' = d + 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.

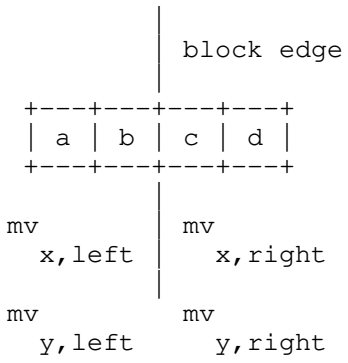


Figure 10: Deblocking filter pixel positions

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:

   delta = clip((4*(c-b) + (d-a) + 4)/8,tc,-tc),

   where tc is a QP-dependent value.

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

   b' = b + delta

   c' = c + 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:

- Sequence header

The following information is signaled at the frame level:

- Frame header

The following information is signaled at the CB level:

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

The following information is signaled at the TB level:

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

The following information is signaled at the PB level:

- Motion vector differences

9.2. Low Level Syntax

9.2.1. CB Level

```
    super-mode          (inter0/split/inter1/inter2-ref0/intra/inter2-ref1/inter
2-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_spl
it=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 NxN (64>=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:

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

+ The absolute value, level=abs(coeff), using a VLC code and

+ If level > 0, the sign bit (sign=0 or sign=1 for coeff>0 and 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 level=abs(coeff) is greater than 1.

3. End of block (EOB) indicating that there are no more non-zero coefficients.

* Additionally, if level = 1, code the sign bit.

* Additionally, if level > 1 define code = 2*(level-2)+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.

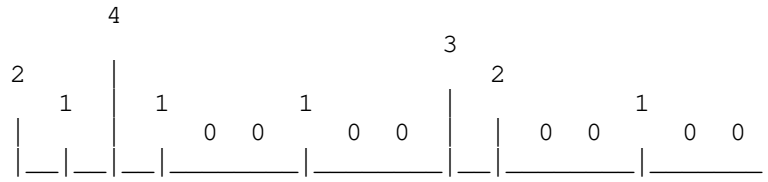


Figure 11: Coefficients to encode

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

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

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

10.2. Frame Header

- Frame type - 1 bit
- QP - 8 bits
- Identification of active reference frames - num_ref*4 bits
- Number of intra modes - 4 bits
- Number of active reference frames - 2 bits
- Active reference frames - number of active reference frames * 6 bits
- Frame number - 16 bits
- 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)
- 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

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: Informational

A. Filippov
Huawei Technologies
A. Norkin
Netflix
J.R. Alvarez
Huawei Technologies

Expires: August 4, 2019

February 4, 2019

<Video Codec Requirements and Evaluation Methodology>
draft-ietf-netvc-requirements-09.txt

Status of this Memo

Copyright Notice

Abstract

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

Table of Contents

1. Introduction....3
2. Applications....3
2.1. Internet Video Streaming....4
2.2. Internet Protocol Television (IPTV)....6
2.3. Video conferencing....7
2.4. Video sharing....8
2.5. Screencasting....9
2.6. Game streaming....10
2.7. Video monitoring / surveillance....11
3. Requirements....12
3.1. General requirements....12
3.2. Basic requirements....14
3.2.1. Input source formats....14
3.2.2. Coding delay....14
3.2.3. Complexity....15
3.2.4. Scalability....15
3.2.5. Error resilience....15
3.3. Optional requirements....16
3.3.1. Input source formats....16
3.3.2. Scalability....16
3.3.3. Complexity....16
3.3.4. Coding efficiency....17
4. Evaluation methodology....17
4.1. Compression performance evaluation....17
4.2. Reference software....20
5. Security Considerations....20
6. Conclusions....20
7. IANA Considerations....20
8. References....20
8.1. Normative References....20
8.2. Informative References....21
9. Acknowledgments....22

Appendix A. Abbreviations used in the text of this document.....23
Appendix B. Used terms.....24

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 Internet video streaming, IPTV, peer-to-peer video conferencing, video sharing, screencasting, game streaming 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 networks with packet losses. 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:

- If an amount of delay is not crucial for an application, then reliable transport protocols such as TCP that retransmits undelivered packets can be used to guarantee correct decoding of transmitted data.

- 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 YCbCr 4:2:0 is assumed for the application scenarios listed below.

2.1. Internet Video Streaming

Typical content for this application is movies, TV-series and shows, and animation. Internet video streaming uses a variety of client devices and has to operate under changing network conditions. For this reason, an adaptive streaming model has been widely adopted. Video material is encoded at different quality levels and different resolutions, which are then chosen by a client depending on its capabilities and current network bandwidth. An example combination of resolutions and bitrates is shown in Table 1 below.

Resolution *	Frame-rate, fps	PAM
4K, 3840x2160	24/1.001, 24, 25,	RA
2K (1080p), 1920x1080	30/1.001, 30, 50,	RA
1080i, 1920x1080*	60/1.001, 60, 100,	RA
720p, 1280x720	120/1.001, 120	RA
576p (EDTV), 720x576	The set of frame-rates	RA
576i (SDTV), 720x576*	presented in this table	RA
480p (EDTV), 720x480	is taken from Table 2	RA
480i (SDTV), 720x480*	in [1]	RA
512x384		RA
QVGA, 320x240		RA

Table 1. Internet Video Streaming: 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 the progressive format.

A video encoding pipeline in on-demand Internet video streaming typically operates as follows:

- Video is encoded in the cloud by software encoders.
- Source video is split into chunks, each of which is encoded separately, in parallel.
- Closed-GOP encoding with 2-5 second intra-picture intervals (or more) is used.
- Encoding is perceptually optimized. Perceptual quality is important and should be considered during the codec development.

Characteristics and requirements of this application scenario are as follows:

- High encoder complexity (up to 10x and more) can be tolerated since encoding happens once and in parallel for different segments.
- Decoding complexity should be kept at reasonable levels to enable efficient decoder implementation.
- Support and efficient encoding of a wide range of content types and formats is required:
  - High Dynamic Range (HDR), Wide Color Gamut (WCG), high resolution (currently, up to 4K), high frame-rate content are important use cases, the codec should be able to encode such content efficiently.
  - Coding efficiency improvement at both lower and higher resolutions is important since low resolutions are used when streaming in low bandwidth conditions.
  - Improvement on both "easy" and "difficult" content in terms of compression efficiency at the same quality level contributes to the overall bitrate/storage savings.
  - Film grain (and sometimes other types of noise) is often present in the streaming movie-type content and is usually a part of the creative intent.

- Significant improvements in compression efficiency between generations of video standards are desirable since this scenario typically assumes long-term support of legacy video codecs.

- Random access points are inserted frequently (one per 2-5 seconds) to enable switching between resolutions and fast-forward playback.

- Elementary stream should have a model that allows easy parsing and identification of the sample components.

- Middle QP values are normally used in streaming, this is also the range where compression efficiency is important for this scenario.

- Scalability or other forms of supporting multiple quality representations are beneficial if they do not incur significant bitrate overhead and if mandated in the first version.

2.2. Internet Protocol Television (IPTV)

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:

- unicast (e.g., for video on demand), where delay is not crucial;

- multicast/broadcast (e.g., for transmitting news) where zapping, i.e. stream changing, delay is important.

In the IPTV scenario, traffic is transmitted over managed (QoS-based) networks. 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, 1-5 seconds). Optional requirements are as follows:

- Temporal (frame-rate) scalability;

- Resolution and quality (SNR) scalability.

For this application, typical values of resolutions, frame-rates, and RAPs are presented in Table 2.

Resolution *	Frame-rate, fps	PAM
2160p (4K),3840x2160	24/1.001, 24, 25,	RA
1080p, 1920x1080	30/1.001, 30, 50,	RA
1080i, 1920x1080*	60/1.001, 60, 100,	RA
720p, 1280x720	120/1.001, 120	RA
576p (EDTV), 720x576	The set of frame-rates	RA
576i (SDTV), 720x576*	presented in this table	RA
480p (EDTV), 720x480	is taken from Table 2	RA
480i (SDTV), 720x480*	in [1]	RA

Table 2. 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.

2.3. 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:

- Delay should be kept as low as possible (the preferable and maximum end-to-end delay values should be less than 100 ms [7] and 320 ms [2], respectively);
- Temporal (frame-rate) scalability;
- 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
1080p, 1920x1080	15, 30	FIZD
720p, 1280x720	30, 60	FIZD
4CIF, 704x576	30, 60	FIZD
4SIF, 704x480	30, 60	FIZD
VGA, 640x480	30, 60	FIZD
360p, 640x360	30, 60	FIZD

Table 3. Video conferencing: typical values of resolutions, frame-rates, and RAPs

2.4. 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:

- Random access to pictures for downloaded video data;
- Temporal (frame-rate) scalability;
- 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 4.

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 4. Video sharing: typical values of resolutions, frame-rates [8], and RAPs

2.5. 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 (numerically) lossless [9]. 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 [9]), 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 low-delay configurations with zero structural delay, a wide range of video formats (e.g., RGB) in addition to YCbCr 4:2:0 and YCbCr 4:4:4 [9]. For this application, typical values of resolutions, frame-rates, and RAPs are presented in Table 5.

Resolution	Frame-rate, fps	PAM
Input color format: RGB 4:4:4		
5k, 5120x2880	15, 30, 60	AI, RA, FIZD
4k, 3840x2160	15, 30, 60	AI, RA, FIZD
WQXGA, 2560x1600	15, 30, 60	AI, RA, FIZD
WUXGA, 1920x1200	15, 30, 60	AI, RA, FIZD

WSXGA+, 1680x1050	15, 30, 60	AI, RA, FIZD
WXGA, 1280x800	15, 30, 60	AI, RA, FIZD
XGA, 1024x768	15, 30, 60	AI, RA, FIZD
SVGA, 800x600	15, 30, 60	AI, RA, FIZD
VGA, 640x480	15, 30, 60	AI, RA, FIZD
Input color format: YCbCr 4:4:4		
5k, 5120x2880	15, 30, 60	AI, RA, FIZD
4k, 3840x2160	15, 30, 60	AI, RA, FIZD
1440p (2K), 2560x1440	15, 30, 60	AI, RA, FIZD
1080p, 1920x1080	15, 30, 60	AI, RA, FIZD
720p, 1280x720	15, 30, 60	AI, RA, FIZD

Table 5. Screencasting for RGB and YCbCr 4:4:4 format: typical values of resolutions, frame-rates, and RAPs

2.6. 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 [10]. 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 [10]. An example of this application is Nvidia Grid [10]. Another category application is broadcast of video games played by people over the Internet in real time or for later viewing [10]. There are many companies such as Twitch, YY in China enable game broadcasting [10]. Games typically contain a lot of sharp edges and large motion [10]. The main requirements are as follows:

- Random access to pictures for game broadcasting;
- 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 5.

2.7. 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 and a monochrome mode (e.g., for infrared cameras) as well as resolution and quality (SNR) scalability is an essential requirement for video surveillance. In some use-cases, high video signal fidelity is required even after lossy compression. Typical values of resolutions, frame-rates, and RAPs for video monitoring / surveillance applications are presented in Table 6.

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

Table 6. 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.

3.1. General requirements

3.1.1. The most basic requirement is coding efficiency, i.e. compression performance on both "easy" and "difficult" content for applications and use cases in Section 2. The codec should provide higher coding efficiency over state-of-the-art video codecs such as HEVC/H.265 and VP9, at least by 25% in accordance with the methodology described in Section 4.1 of this document. For higher resolutions, the coding efficiency improvements are expected to be higher than for lower resolutions.

3.1.2. Good quality specification and well-defined profiles and levels are required to enable device interoperability and facilitate decoder implementations. A profile consists of a subset of entire bitstream syntax elements and consequently it also defines the necessary tools for decoding a conforming bitstream of that profile. A level imposes a set of numerical limits to the values of some syntax elements. An example of codec levels to be supported is presented in Table 7. An actual level definition should include constraints on features that impact the decoder complexity. For example, these features might be as follows: maximum bit-rate, line buffer size, memory usage, etc.

Level	Example picture resolution at highest frame rate
1	128x96(12,288*)@30.0 176x144(25,344*)@15.0
2	352x288(101,376*)@30.0
3	352x288(101,376*)@60.0 640x360(230,400*)@30.0
4	640x360(230,400*)@60.0 960x540(518,400*)@30.0
5	720x576(414,720*)@75.0 960x540(518,400*)@60.0 1280x720(921,600*)@30.0

	1,280x720(921,600*)@68.0
6	2,048x1,080(2,211,840*)@30.0
	1,280x720(921,600*)@120.0
7	2,048x1,080(2,211,840*)@60.0
	1,920x1,080(2,073,600*)@120.0
8	3,840x2,160(8,294,400*)@30.0
	4,096x2,160(8,847,360*)@30.0
	1,920x1,080(2,073,600*)@250.0
9	4,096x2,160(8,847,360*)@60.0
	1,920x1,080(2,073,600*)@300.0
10	4,096x2,160(8,847,360*)@120.0
	3,840x2,160(8,294,400*)@120.0
11	8,192x4,320(35,389,440*)@30.0
	3,840x2,160(8,294,400*)@250.0
12	8,192x4,320(35,389,440*)@60.0
	3,840x2,160(8,294,400*)@300.0
13	8,192x4,320(35,389,440*)@120.0

Table 7. Codec levels

NB *: The quantities of pixels are presented for such applications where a picture can have an arbitrary size (e.g., screencasting)

3.1.3. Bitstream syntax should allow extensibility and backward compatibility. New features can be supported easily by using metadata (e.g., such as SEI messages, VUI, headers) without affecting the bitstream compatibility with legacy decoders. A newer version of the decoder shall be able to play bitstreams of an older version of the same or lower profile and level.

3.1.4. A bitstream should have a model that allows easy parsing and identification of the sample components (such as ISO/IEC14496-10, Annex B or ISO/IEC 14496-15). In particular, information needed for packet handling (e.g., frame type) should not require parsing anything below the header level.

3.1.5. Perceptual quality tools (such as adaptive QP and quantization matrices) should be supported by the codec bit-stream.

3.1.6. The codec specification shall define a buffer model such as hypothetical reference decoder (HRD).

3.1.7. Specifications providing integration with system and delivery layers should be developed.

3.2. Basic requirements

3.2.1. Input source formats:

- Bit depth: 8- and 10-bits (up to 12-bits for a high profile) per color component;
- Color sampling formats:
  - YCbCr 4:2:0;
  - YCbCr 4:4:4, YCbCr 4:2:2 and YCbCr 4:0:0 (preferably in different profile(s)).
- For profiles with bit depth of 10 bits per sample or higher, support of high dynamic range and wide color gamut.
- Support of arbitrary resolution according to the level constraints for such applications where a picture can have an arbitrary size (e.g., in screencasting).
- Exemplary input source formats for codec profiles are shown in Table 8.

Profile	Bit-depths per color component	Color sampling formats
1	8 and 10	4:0:0 and 4:2:0
2	8 and 10	4:0:0, 4:2:0 and 4:4:4
3	8, 10 and 12	4:0:0, 4:2:0, 4:2:2 and 4:4:4

Table 8. Exemplary input source formats for codec profiles

3.2.2. Coding delay:

- Support of configurations with zero structural delay also referred to as "low-delay" configurations.

. Note 1: end-to-end delay should be up to 320 ms [2] but its preferable value should be less than 100 ms [7]

- Support of efficient random access point encoding (such as intra coding and resetting of context variables) as well as efficient switching between multiple quality representations.

- Support of configurations with non-zero structural delay (such as out-of-order or multi-pass encoding) for applications without low-delay requirements if such configurations provide additional compression efficiency improvements.

3.2.3. Complexity:

- Feasible real-time implementation of both an encoder and a decoder supporting a chosen subset of tools for hardware and software implementation on a wide range of state-of-the-art platforms. The real-time encoder tools subset should provide meaningful improvement in compression efficiency at reasonable complexity of hardware and software encoder implementations as compared to current real-time implementations of state-of-the-art video compression technologies such as HEVC/H.265 and VP9.

- High-complexity software encoder implementations used by offline encoding applications can have 10x or more complexity increase compared to state-of-the-art video compression technologies such as HEVC/H.265 and VP9.

3.2.4. Scalability:

- Temporal (frame-rate) scalability should be supported.

3.2.5. Error resilience:

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

- The codec should support mechanisms that facilitate packetization of a bitstream for common network protocols.

- Packetization mechanisms should enable frame-level error recovery by means of retransmission or error concealment.

- The codec should support effective mechanisms for allowing decoding and reconstruction of significant parts of pictures in the event that parts of the picture data are lost in transmission.

- The bitstream specification shall support independently decodable sub-frame units similar to slices or independent tiles. It shall be possible for the encoder to restrict the bit-stream to allow parsing of the bit-stream after a packet-loss and to communicate it to the decoder.

3.3. Optional requirements

3.3.1. Input source formats

- Bit depth: up to 16-bits per color component.

- Color sampling formats: RGB 4:4:4.

- Auxiliary channel (e.g., alpha channel) support.

3.3.2. Scalability:

- Resolution and quality (SNR) scalability that provide low compression efficiency penalty (up to 5% of BD-rate [12] increase per layer with reasonable increase of both computational and hardware complexity) can be supported in the main profile of the codec being developed by the NETVC WG. Otherwise, a separate profile is needed to support these types of scalability.

- Computational complexity scalability(i.e. computational complexity is decreasing along with degrading picture quality) is desirable.

3.3.3. 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.

- 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

- Low-level instruction set parallelism: favor algorithms that are SIMD/GPU friendly over inherently serial algorithms

3.3.4. Coding efficiency

Compression efficiency on noisy content, content with film grain, computer generated content, and low resolution materials is desirable.

4. Evaluation methodology

4.1. Compression performance evaluation

As shown in Fig.1, compression performance testing is performed in 3 overlapped ranges that encompass 10 different bitrate values:

- Low bitrate range (LBR) is the range that contains the 4 lowest bitrates of the 10 specified bitrates (1 of the 4 bitrate values is shared with the neighboring range);

- Medium bitrate range (MBR) is the range that contains the 4 medium bitrates of the 10 specified bitrates (2 of the 4 bitrate values are shared with the neighboring ranges);

- High bitrate range (HBR) is the range that contains the 4 highest bitrates of the 10 specified bitrates (1 of the 4 bitrate values is shared with the neighboring range).

Initially, for the codec selected as a reference one (e.g., HEVC or VP9), a set of 10 QP (quantization parameter) values should be specified (in a separate document on Internet video codec testing) and corresponding quality values should be calculated. In Fig.1, QP and quality values are denoted as QP0, QP1, QP2,..., QP8, QP9 and Q0, Q1, Q2,..., Q8, Q9, respectively. To guarantee the overlaps of quality levels between the bitrate ranges of the reference and tested codecs, a quality alignment procedure should be performed for each range's outermost (left- and rightmost) quality levels Qk of the reference codec (i.e. for Q0, Q3, Q6, and Q9) and the quality levels Q'k (i.e. Q'0, Q'3, Q'6, and Q'9) of the tested codec. Thus, these quality levels Q'k and, hence, the corresponding QP value QP'k (i.e. QP'0, QP'3, QP'6, and QP'9) of the tested codec should be selected using the following formulas:

$$Q'k = \min_{i \text{ in } R} \{ \text{abs}(Q'i - Qk) \},$$

$$QP'k = \underset{i \in R}{\operatorname{argmin}} \{ \operatorname{abs}(Q'i(QP'i) - Qk(QPk)) \},$$

where R is the range of the QP indexes of the tested codec, i.e. the candidate Internet video codec. The inner quality levels (i.e. Q'1, Q'2, Q'4, Q'5, Q'7, and Q'8) as well as their corresponding QP values of each range (i.e. QP'1, QP'2, QP'4, QP'5, QP'7, and QP'8) should be as equidistantly spaced as possible between the left- and rightmost quality levels without explicitly mapping their values using the above described procedure.

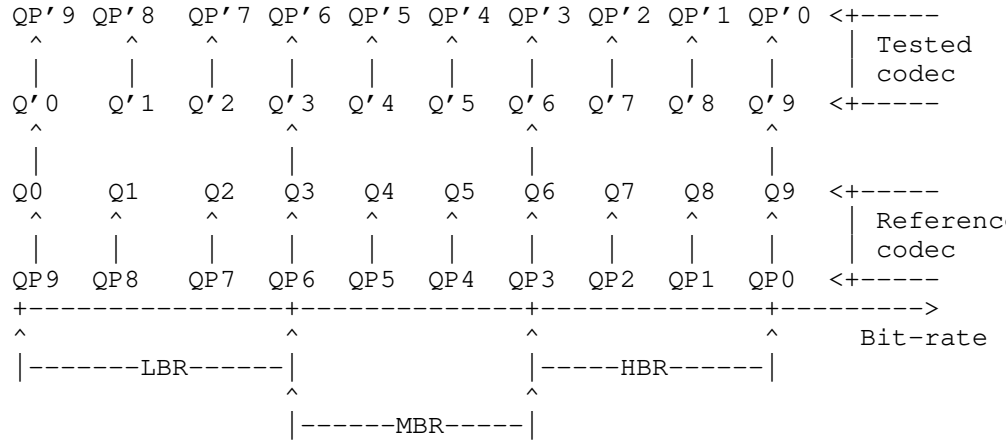


Figure 1 Quality/QP alignment for compression performance evaluation

Since the QP mapping results may vary for different sequences, eventually, this quality alignment procedure needs to be separately performed for each quality assessment index and each sequence used for codec performance evaluation to fulfill the above described requirements.

To assess the quality of output (decoded) sequences, two indexes, PSNR [3] and MS-SSIM [3,11] are separately computed. In the case of the YCbCr color format, PSNR should be calculated for each color plane whereas MS-SSIM is calculated for luma channel only. In the case of the RGB color format, both metrics are computed for R, G and B channels. Thus, for each sequence, 30 RD-points for PSNR (i.e. three RD-curves, one for each channel) and 10 RD-points for MS-SSIM (i.e. one RD-curve, for luma channel only) should be calculated in the case of YCbCr. If content is encoded as RGB, 60 RD-points (30 for PSNR and 30 for MS-SSIM) should be calculated, i.e. three RD-curves (one for each channel) are computed for PSNR as well as three RD-curves (one for each channel) for MS-SSIM.

Finally, to obtain an integral estimation, BD-rate savings [12] should be computed for each range and each quality index. In addition, average values over all the 3 ranges should be provided for both PSNR and MS-SSIM. A list of video sequences that should be used for testing as well as the 10 QP values for the reference codec 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, state-of-the-art video codecs such as HEVC/H.265 [4,5] or VP9 must be used. The reference source code of the HEVC/H.265 codec can be found at [6]. The HEVC/H.265 codec must be configured according to [13] and Table 9.

Intra-period, second	HEVC/H.265 encoding mode according to [13]
AI	Intra Main or Intra Main10
RA	Random access Main or Random access Main10
FIZD	Low delay Main or Low delay Main10

Table 9. Intra-periods for different HEVC/H.265 encoding modes according to [13]

According to the coding efficiency requirement described in Section 3.1.1, BD-rate savings calculated for each color plane and averaged for all the video sequences used to test the NETVC codec should be, at least,

- 25% if calculated over the whole bitrate range;

- 15% if calculated for each bitrate subrange (LBR, MBR, HBR).

Since values of the two objective metrics (PSNR and MS-SSIM) are available for some color planes, each value should meet these coding efficiency requirements, i.e. the final BD-rate saving denoted as S is calculated for a given color plane as follows:

S = min { S_psnr, S_ms-ssim },

where S_psnr and S_ms-ssim are BD-rate savings calculated for the given color plane using PSNR and MS-SSIM metrics, respectively.

In addition to the objective quality measures defined above, subjective evaluation must also be performed for the final NETVC codec adoption. For subjective tests, the MOS-based evaluation procedure must be used as described in section 2.1 of [3]. 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.

4.2. Reference software

Reference software provided to the NETVC WG for candidate codecs should comprise a fully operational encoder supporting necessary rate controls, subjective quality optimization features and some degree of speed optimization and a "real-time" decoder.

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. IANA Considerations

This document has no IANA actions.

8. References

8.1. Normative References

[1] Recommendation ITU-R BT.2020-2: Parameter values for ultra-high definition television systems for production and international programme exchange, 2015.

[2] Recommendation ITU-T G.1091: Quality of Experience requirements for telepresence services, 2014.

[3] ISO/IEC PDTR 29170-1: Information technology -- Advanced image coding and evaluation methodologies -- Part 1: Guidelines for codec evaluation.

[4] ISO/IEC 23008-2:2015. Information technology -- High efficiency coding and media delivery in heterogeneous environments -- Part 2: High efficiency video coding

[5] Recommendation ITU-T H.265: High efficiency video coding, 2013.

[6] https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/

8.2. Informative References

[7] 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)"

[8] "Recommended upload encoding settings (Advanced)" https://support.google.com/youtube/answer/1722171?hl=en

[9] 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

[10] Manindra Parhy, "Game streaming requirement for Future Video Coding," MPEG Contribution m36771, June 2015, Warsaw, Poland.

[11] 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.

[12] 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.

[13] F. Bossen, "Common test conditions and software reference configurations," JCTVC-L1100, Geneva, Switzerland, Jan. 2013.

[14] http://www.digitizationguidelines.gov/term.php?term=compressionvisuallylossless)

9. Acknowledgments

The authors would like to thank Dr. Jianle Chen, Mr. Jiantong Zhou, Mr. Paul Coverdale, Mr. Vasily Rufitskiy, and Dr. Haitao Yang for many useful discussions on this document and their help while preparing it as well as Mr. Mo Zanaty, Dr. Minhua Zhou, Dr. Ali Begen, Mr. Thomas Daede, Dr. Thomas Davies, Mr. Jonathan Lennox, Dr. Timothy Terriberry, Mr. Peter Thatcher, Dr. Jean-Marc Valin, Mr. Jack Moffitt, Mr. Greg Coppa and Mr. Andrew Krupiczka for their valuable comments on different 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
FIZD	just the First picture is Intra-coded, Zero structural Delay
GOP	Group of Picture
HBR	High Bitrate Range
HDR	High Dynamic Range
HRD	Hypothetical Reference Decoder
IPTV	Internet Protocol Television
LBR	Low Bitrate Range
MBR	Medium Bitrate Range
MOS	Mean Opinion Score
MS-SSIM	Multi-Scale Structural Similarity quality index
PAM	Picture Access Mode
PSNR	Peak Signal-to-Noise Ratio
QoS	Quality of Service
QP	Quantization Parameter
RA	Random Access
RAP	Random Access Period
RD	Rate-Distortion
SEI	Supplemental Enhancement Information
UGC	User-Generated Content
VDI	Virtual Desktop Infrastructure
VUI	Video Usability Information
WCG	Wide Color Gamut

Appendix B. Used terms

Term	Meaning
High dynamic range imaging	is a set of techniques that allow a greater dynamic range of exposures or values (i.e., a wide range of values between light and dark areas) than normal digital imaging techniques. The intention is to accurately represent the wide range of intensity levels found in such examples as exterior scenes that include light-colored items struck by direct sunlight and areas of deep shadow [14].
Random access period	is the period of time between two closest independently decodable frames (pictures).
RD-point	A point in a 2 dimensional rate-distortion space where the values of bitrate and quality metric are used as x- and y-coordinates, respectively
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 [14].
Wide color gamut	is a certain complete color subset (e.g., considered in ITU-R BT.2020) that supports a wider range of colors (i.e., an extended range of colors that can be generated by a specific input or output device such as a video camera, monitor or printer and can be interpreted by a color model) than conventional color gamuts (e.g., considered in ITU-R BT.601 or BT.709).

Authors' Addresses

Alexey Filippov
Huawei Technologies

Email: alexey.filippov@huawei.com

Andrey Norkin
Netflix

Email: anorkin@netflix.com

Jose Roberto Alvarez
Huawei Technologies

Email: j.alvarez@ieee.org

Network Working Group
Internet-Draft
Intended status: Informational
Expires: July 29, 2019

T. Daede
Mozilla
A. Norkin
Netflix
I. Brailovskiy
Amazon Lab126

January 25, 2019

Video Codec Testing and Quality Measurement
draft-ietf-netvc-testing-08

Abstract

This document describes guidelines and procedures for evaluating a video codec. 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 July 29, 2019.

Copyright Notice

Table of Contents

1. Introduction 2
2. Subjective quality tests 3
2.1. Still Image Pair Comparison 3
2.2. Video Pair Comparison 4
2.3. Mean Opinion Score 4
3. Objective Metrics 5
3.1. Overall PSNR 5
3.2. Frame-averaged PSNR 5
3.3. PSNR-HVS-M 5
3.4. SSIM 6
3.5. Multi-Scale SSIM 6
3.6. CIEDE2000 6
3.7. VMAF 6
4. Comparing and Interpreting Results 7
4.1. Graphing 7
4.2. BD-Rate 7
4.3. Ranges 8
5. Test Sequences 8
5.1. Sources 8
5.2. Test Sets 8
5.2.1. regression-1 8
5.2.2. objective-2-slow 9
5.2.3. objective-2-fast 12
5.2.4. objective-1.1 14
5.2.5. objective-1-fast 17
5.3. Operating Points 19
5.3.1. Common settings 19
5.3.2. High Latency CQP 19
5.3.3. Low Latency CQP 19
5.3.4. Unconstrained High Latency 20
5.3.5. Unconstrained Low Latency 20
6. Automation 20
6.1. Regression tests 21
6.2. Objective performance tests 21
6.3. Periodic tests 22
7. Informative References 22
Authors' Addresses 23

1. Introduction

When developing a 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 when evaluating changes under consideration. Authors of features or changes should provide the results of the appropriate test when proposing codec modifications.

2. Subjective quality tests

Subjective testing is the preferable method of testing video codecs.

Subjective testing results take priority over objective testing results, when available. Subjective testing is recommended especially when taking advantage of psychovisual effects that may not be well represented by objective metrics, or when different objective metrics disagree.

Selection of a testing methodology depends on the feature being tested and the resources available. Test methodologies are presented in order of increasing accuracy and cost.

Testing relies on the resources of 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.

Subjective tests should use the same operating points as the objective tests.

2.1. Still Image Pair Comparison

A simple way to determine superiority of one compressed image is to visually compare two compressed images, and have the viewer judge which one has a higher quality. For example, this test may be suitable for an intra de-ringing filter, but not for a new inter prediction mode. 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.

Once testing is complete, a p-value can be computed using the binomial test. A significant result should have a resulting p-value less than or equal to 0.5. For example:

```
p_value = binom_test(a,a+b)
```

where a is the number of votes for one video, b is the number of votes for the second video, and binom_test(x,y) returns the binomial PMF with x observed tests, y total tests, and expected probability 0.5.

If ties are allowed to be reported, then the equation is modified:

p_value = binom_test(a+floor(t/2),a+b+t)

where t is the number of tie votes.

Still image pair comparison is used for rapid comparisons during development - the viewer may be either a developer or user, for example. As the results are only relative, it is effective even with an inconsistent viewing environment. Because this test only uses still images (keyframes), this is only suitable for changes with similar or no effect on inter frames.

2.2. Video Pair Comparison

The still image pair comparison method can be modified to also compare vidoes. This is necessary when making changes with temporal effects, such as changes to inter-frame prediction. Video pair comparisons follow the same procedure as still images. Videos used for testing should be limited to 10 seconds in length, and can be rewatched an unlimited number of times.

2.3. Mean Opinion Score

A Mean Opinion Score (MOS) viewing test is the preferred method of evaluating the quality. The subjective test should be performed as either consecutively showing the video sequences on one screen or on two screens located side-by-side. The testing procedure should normally follow rules described in [BT500] and be performed with non-expert test subjects. The result of the test will be (depending on the test procedure) mean opinion scores (MOS) or differential mean opinion scores (DMOS). Confidence intervals are also calculated to judge whether the difference between two encodings is statistically significant. In certain cases, a viewing test with expert test subjects can be performed, for example if a test should evaluate technologies with similar performance with respect to a particular artifact (e.g. loop filters or motion prediction). Unlike pair comparisions, a MOS test requires a consistent testing environment. This means that for large scale or distributed tests, pair comparisons are preferred.

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]. Implementations of metrics must directly support the input's resolution, bit depth, and sampling format.

Unless otherwise specified, all of the metrics described below only apply to the luma plane, individually by frame. When applied to the video, the scores of each frame are averaged to create the final score.

Codecs must output the same resolution, bit depth, and sampling format as the input.

3.1. Overall PSNR

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

20 * log10 (MAX / RMSE)

or, equivalently:

10 * log10 (MAX^2 / MSE)

where the error is computed over all the pixels in the video, which is the method used in the dump_psnr.c reference implementation.

This metric may be applied to both the luma and chroma planes, with all planes reported separately.

3.2. Frame-averaged PSNR

PSNR can also be calculated per-frame, and then the values averaged together. This is reported in the same way as overall PSNR.

3.3. 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.

3.4. 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.

To linearize the metric for BD-Rate computation, the score is converted into a nonlinear decibel scale:

-10 * log10 (1 - SSIM)

3.5. Multi-Scale SSIM

Multi-Scale SSIM is SSIM extended to multiple window sizes [MSSSIM]. The metric score is converted to decibels in the same way as SSIM.

3.6. CIEDE2000

CIEDE2000 is a metric based on CIEDE color distances [CIEDE2000]. It generates a single score taking into account all three chroma planes. It does not take into consideration any structural similarity or other psychovisual effects.

3.7. VMAF

Video Multi-method Assessment Fusion (VMAF) is a full-reference perceptual video quality metric that aims to approximate human perception of video quality [VMAF]. This metric is focused on quality degradation due compression and rescaling. VMAF estimates the perceived quality score by computing scores from multiple quality assessment algorithms, and fusing them using a support vector machine (SVM). Currently, three image fidelity metrics and one temporal signal have been chosen as features to the SVM, namely Anti-noise SNR (ANSNR), Detail Loss Measure (DLM), Visual Information Fidelity (VIF), and the mean co-located pixel difference of a frame with respect to the previous frame.

The quality score from VMAF is used directly to calculate BD-Rate, without any conversions.

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 publication, the X axis should be linear. The Y axis metric should be plotted in decibels. If the quality metric does not natively report quality in decibels, it should be converted as described in the previous section.

4.2. BD-Rate

The Bjontegaard rate difference, also known as BD-rate, allows the measurement of the bitrate reduction offered by a codec or codec feature, while maintaining the same quality as measured by objective metrics. The rate change is computed as the average percent difference in rate over a range of qualities. Metric score ranges are not static - they are calculated either from a range of bitrates of the reference codec, or from quantizers of a third, anchor codec. Given a reference codec and test codec, BD-rate values are calculated as follows:

- Rate/distortion points are calculated for the reference and test codec.
  - At least four points must be computed. These points should be the same quantizers when comparing two versions of the same codec.
  - Additional points outside of the range should be discarded.
- The rates are converted into log-rates.
- A piecewise cubic hermite interpolating polynomial is fit to the points for each codec to produce functions of log-rate in terms of distortion.
- Metric score ranges are computed:
  - If comparing two versions of the same codec, the overlap is the intersection of the two curves, bound by the chosen quantizer points.
  - If comparing dissimilar codecs, a third anchor codec's metric scores at fixed quantizers are used directly as the bounds.

- The log-rate is numerically integrated over the metric range for each curve, using at least 1000 samples and trapezoidal integration.

- The resulting integrated log-rates are converted back into linear rate, and then the percent difference is calculated from the reference to the test codec.

4.3. Ranges

For individual feature changes in libaom or libvpx, the overlap BD-Rate method with quantizers 20, 32, 43, and 55 must be used.

For the final evaluation described in [I-D.ietf-netvc-requirements], the quantizers used are 20, 24, 28, 32, 36, 39, 43, 47, 51, and 55.

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.

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 for reproducibility of results. All current test sets are available for download [TESTSEQUENCES].

Test sequences should be downloaded in whole. They should not be recreated from the original sources.

5.2.1. regression-1

This test set is used for basic regression testing. It contains a very small number of clips.

- kirlandvga (640x360, 8bit, 4:2:0, 300 frames)
- FourPeople (1280x720, 8bit, 4:2:0, 60 frames)
- Narrarator (4096x2160, 10bit, 4:2:0, 15 frames)
- CSGO (1920x1080, 8bit, 4:4:4 60 frames)

5.2.2. objective-2-slow

This test set is a comprehensive test set, grouped by resolution. These test clips were created from originals at [TESTSEQUENCES]. They have been scaled and cropped to match the resolution of their category. This test set requires compiling with high bit depth support.

4096x2160, 4:2:0, 60 frames:

- Netflix_BarScene_4096x2160_60fps_10bit_420_60f
- Netflix_BoxingPractice_4096x2160_60fps_10bit_420_60f
- Netflix_Dancers_4096x2160_60fps_10bit_420_60f
- Netflix_Narrator_4096x2160_60fps_10bit_420_60f
- Netflix_RitualDance_4096x2160_60fps_10bit_420_60f
- Netflix_ToddlerFountain_4096x2160_60fps_10bit_420_60f
- Netflix_WindAndNature_4096x2160_60fps_10bit_420_60f
- street_hdr_amazon_2160p

1920x1080, 4:2:0, 60 frames:

- aspen_1080p_60f
- crowd_run_1080p50_60f
- ducks_take_off_1080p50_60f
- guitar_hdr_amazon_1080p
- life_1080p30_60f
- Netflix_Aerial_1920x1080_60fps_8bit_420_60f

- Netflix_Boat_1920x1080_60fps_8bit_420_60f
- Netflix_Crosswalk_1920x1080_60fps_8bit_420_60f
- Netflix_FoodMarket_1920x1080_60fps_8bit_420_60f
- Netflix_PierSeaside_1920x1080_60fps_8bit_420_60f
- Netflix_SquareAndTimelapse_1920x1080_60fps_8bit_420_60f
- Netflix_TunnelFlag_1920x1080_60fps_8bit_420_60f
- old_town_cross_1080p50_60f
- pan_hdr_amazon_1080p
- park_joy_1080p50_60f
- pedestrian_area_1080p25_60f
- rush_field_cuts_1080p_60f
- rush_hour_1080p25_60f
- seaplane_hdr_amazon_1080p
- station2_1080p25_60f
- touchdown_pass_1080p_60f

1280x720, 4:2:0, 120 frames:

- boat_hdr_amazon_720p
- dark720p_120f
- FourPeople_1280x720_60_120f
- gipsrestat720p_120f
- Johnny_1280x720_60_120f
- KristenAndSara_1280x720_60_120f
- Netflix_DinnerScene_1280x720_60fps_8bit_420_120f
- Netflix_DrivingPOV_1280x720_60fps_8bit_420_120f

- Netflix_FoodMarket2_1280x720_60fps_8bit_420_120f
- Netflix_RollerCoaster_1280x720_60fps_8bit_420_120f
- Netflix_Tango_1280x720_60fps_8bit_420_120f
- rain_hdr_amazon_720p
- vidyo1_720p_60fps_120f
- vidyo3_720p_60fps_120f
- vidyo4_720p_60fps_120f

640x360, 4:2:0, 120 frames:

- blue_sky_360p_120f
- controlled_burn_640x360_120f
- desktop2360p_120f
- kirland360p_120f
- mmstationary360p_120f
- niklas360p_120f
- rain2_hdr_amazon_360p
- red_kayak_360p_120f
- riverbed_360p25_120f
- shields2_640x360_120f
- snow_mnt_640x360_120f
- speed_bag_640x360_120f
- stockholm_640x360_120f
- tacomanarrows360p_120f
- thaloundeskmtg360p_120f
- water_hdr_amazon_360p

426x240, 4:2:0, 120 frames:

- bqfree_240p_120f
- bqhighway_240p_120f
- bqzoom_240p_120f
- chairlift_240p_120f
- dirtbike_240p_120f
- mozzoom_240p_120f

1920x1080, 4:4:4 or 4:2:0, 60 frames:

- CSGO_60f.y4m
- DOTA2_60f_420.y4m
- MINECRAFT_60f_420.y4m
- STARCRAFT_60f_420.y4m
- EuroTruckSimulator2_60f.y4m
- Hearthstone_60f.y4m
- wikipedia_420.y4m
- pvq_slideshow.y4m

5.2.3. objective-2-fast

This test set is a strict subset of objective-2-slow. It is designed for faster runtime. This test set requires compiling with high bit depth support.

1920x1080, 4:2:0, 60 frames:

- aspen_1080p_60f
- ducks_take_off_1080p50_60f
- life_1080p30_60f
- Netflix_Aerial_1920x1080_60fps_8bit_420_60f

- Netflix_Boat_1920x1080_60fps_8bit_420_60f
- Netflix_FoodMarket_1920x1080_60fps_8bit_420_60f
- Netflix_PierSeaside_1920x1080_60fps_8bit_420_60f
- Netflix_SquareAndTimelapse_1920x1080_60fps_8bit_420_60f
- Netflix_TunnelFlag_1920x1080_60fps_8bit_420_60f
- rush_hour_1080p25_60f
- seaplane_hdr_amazon_1080p
- touchdown_pass_1080p_60f

1280x720, 4:2:0, 120 frames:

- boat_hdr_amazon_720p
- dark720p_120f
- gipsrestat720p_120f
- KristenAndSara_1280x720_60_120f
- Netflix_DrivingPOV_1280x720_60fps_8bit_420_60f
- Netflix_RollerCoaster_1280x720_60fps_8bit_420_60f
- vidyo1_720p_60fps_120f
- vidyo4_720p_60fps_120f

640x360, 4:2:0, 120 frames:

- blue_sky_360p_120f
- controlled_burn_640x360_120f
- kirland360p_120f
- niklas360p_120f
- rain2_hdr_amazon_360p
- red_kayak_360p_120f

- riverbed_360p25_120f
- shields2_640x360_120f
- speed_bag_640x360_120f
- thaloundeskmtg360p_120f

426x240, 4:2:0, 120 frames:

- bqfree_240p_120f
- bqzoom_240p_120f
- dirtbike_240p_120f

1290x1080, 4:2:0, 60 frames:

- DOTA2_60f_420.y4m
- MINECRAFT_60f_420.y4m
- STARCRAFT_60f_420.y4m
- wikipedia_420.y4m

5.2.4. objective-1.1

This test set is an old version of objective-2-slow.

4096x2160, 10bit, 4:2:0, 60 frames:

- Aerial (start frame 600)
- BarScene (start frame 120)
- Boat (start frame 0)
- BoxingPractice (start frame 0)
- Crosswalk (start frame 0)
- Dancers (start frame 120)
- FoodMarket
- Narrator

- PierSeaside
- RitualDance
- SquareAndTimelapse
- ToddlerFountain (start frame 120)
- TunnelFlag
- WindAndNature (start frame 120)

1920x1080, 8bit, 4:4:4, 60 frames:

- CSGO
- DOTA2
- EuroTruckSimulator2
- Hearthstone
- MINECRAFT
- STARCRAFT
- wikipedia
- pvq_slideshow

1920x1080, 8bit, 4:2:0, 60 frames:

- ducks_take_off
- life
- aspen
- crowd_run
- old_town_cross
- park_joy
- pedestrian_area
- rush_field_cuts

- rush_hour
- station2
- touchdown_pass

1280x720, 8bit, 4:2:0, 60 frames:

- Netflix_FoodMarket2
- Netflix_Tango
- DrivingPOV (start frame 120)
- DinnerScene (start frame 120)
- RollerCoaster (start frame 600)
- FourPeople
- Johnny
- KristenAndSara
- vidyo1
- vidyo3
- vidyo4
- dark720p
- gipsrecmotion720p
- gipsrestat720p
- controlled_burn
- stockholm
- speed_bag
- snow_mnt
- shields

640x360, 8bit, 4:2:0, 60 frames:

- red_kayak
- blue_sky
- riverbed
- thaloundeskmtgvga
- kirlandvga
- tacomanarrowsvga
- tacomascmvvga
- desktop2360p
- mmmovingvga
- mmstationaryvga
- niklasvga

5.2.5. objective-1-fast

This is an old version of objective-2-fast.

1920x1080, 8bit, 4:2:0, 60 frames:

- Aerial (start frame 600)
- Boat (start frame 0)
- Crosswalk (start frame 0)
- FoodMarket
- PierSeaside
- SquareAndTimelapse
- TunnelFlag

1920x1080, 8bit, 4:2:0, 60 frames:

- CSGO
- EuroTruckSimulator2

- MINECRAFT
- wikipedia

1920x1080, 8bit, 4:2:0, 60 frames:

- ducks_take_off
- aspen
- old_town_cross
- pedestrian_area
- rush_hour
- touchdown_pass

1280x720, 8bit, 4:2:0, 60 frames:

- Netflix_FoodMarket2
- DrivingPOV (start frame 120)
- RollerCoaster (start frame 600)
- Johnny
- vidyo1
- vidyo4
- gipsrecmotion720p
- speed_bag
- shields

640x360, 8bit, 4:2:0, 60 frames:

- red_kayak
- riverbed
- kirlandvga
- tacomascmvvga

- mmmovingvga
- niklasvga

5.3. Operating Points

Four 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. Both of these modes come in CQP and unconstrained variants. When testing still image sets, such as subset1, high latency CQP mode should be used.

5.3.1. Common settings

Encoders should be configured to their best settings when being compared against each other:

- av1: -codec=av1 -ivf -frame-parallel=0 -tile-columns=0 -cpu-used=0 -threads=1

5.3.2. High Latency CQP

High Latency CQP is used for evaluating incremental changes to a codec. This method is well suited to compare codecs with similar coding tools. It allows codec features with intrinsic frame delay.

- daala: -v=x -b 2
- vp9: -end-usage=q -cq-level=x -lag-in-frames=25 -auto-alt-ref=2
- av1: -end-usage=q -cq-level=x -auto-alt-ref=2

5.3.3. Low Latency CQP

Low Latency CQP is used for evaluating incremental changes to a codec. This method is well suited to compare codecs with similar coding tools. It requires the codec to be set for zero intrinsic frame delay.

- daala: -v=x
- av1: -end-usage=q -cq-level=x -lag-in-frames=0

5.3.4. Unconstrained 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:

- x264: --crf=x
- x265: --crf=x
- daala: -v=x -b 2
- av1: --end-usage=q --cq-level=x --lag-in-frames=25 --auto-alt-ref=2

5.3.5. 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:

- x264: --crf-x --tune zerolatency
- x265: --crf=x --tune zerolatency
- daala: -v=x
- av1: --end-usage=q --cq-level=x --lag-in-frames=0

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 computers for fast results, with rd_tool [RD_TOOL]. This tool can be run via a web interface called AreWeCompressedYet [AWCY], or locally.

Because of computational constraints, several levels of testing are specified.

6.1. Regression tests

Regression tests run on a small number of short sequences - regression-test-1. The regression tests should include a number of various test conditions. The purpose of regression tests is to ensure bug fixes (and similar patches) do not negatively affect the performance. The anchor in regression tests is the previous revision of the codec in source control. Regression tests are run on both high and low latency CQP modes

6.2. Objective performance tests

Changes that are expected to affect the quality of encode or bitstream should run an objective performance test. The performance tests should be run on a wider number of sequences. The following data should be reported:

- Identifying information for the encoder used, such as the git commit hash.
- Command line options to the encoder, configure script, and anything else necessary to replicate the experiment.
- The name of the test set run (objective-1-fast)
- For both high and low latency CQP modes, and for each objective metric:
  - The BD-Rate score, in percent, for each clip.
  - The average of all BD-Rate scores, equally weighted, for each resolution category in the test set.
  - The average of all BD-Rate scores for all videos in all categories.

Normally, the encoder should always be run at the slowest, highest quality speed setting (cpu-used=0 in the case of AV1 and VP9). However, in the case of computation time, both the reference and changed encoder can be built with some options disabled. For AV1, -disable-ext_partition and -disable-ext_partition_types can be passed to the configure script to substantially speed up encoding, but the usage of these options must be reported in the test results.

6.3. Periodic tests

Periodic tests are run on a wide range of bitrates in order to gauge progress over time, as well as detect potential regressions missed by other tests.

7. Informative References

[AWCY] Xiph.Org, "Are We Compressed Yet?", 2016, <https://arewecompressedyet.com/>.

[BT500] ITU-R, "Recommendation ITU-R BT.500-13", 2012, <https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.500-13-201201-I!!PDF-E.pdf>.

[CIEDE2000] Yang, Y., Ming, J., and N. Yu, "Color Image Quality Assessment Based on CIEDE2000", 2012, <http://dx.doi.org/10.1155/2012/273723>.

[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>.

[I-D.ietf-netvc-requirements] Filippov, A. and A. Norkin, "<Video Codec Requirements and Evaluation Methodology>", draft-ietf-netvc-requirements-08 (work in progress), May 2018.

[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.

[RD_TOOL] Xiph.Org, "rd_tool", 2016, <https://github.com/tdaede/rd_tool>.

[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>.

[TESTSEQUENCES]
Daede, T., "Test Sets", n.d.,
<https://people.xiph.org/~tdaede/sets/>.

[VMAF] Aaron, A., Li, Z., Manohara, M., Lin, J., Wu, E., and C. Kuo, "VMAF - Video Multi-Method Assessment Fusion", 2015, <https://github.com/Netflix/vmaf>.

Authors' Addresses

Thomas Daede
Mozilla

Email: tdaede@mozilla.com

Andrey Norkin
Netflix

Email: anorkin@netflix.com

Ilya Brailovskiy
Amazon Lab126

Email: brailovs@lab126.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

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.

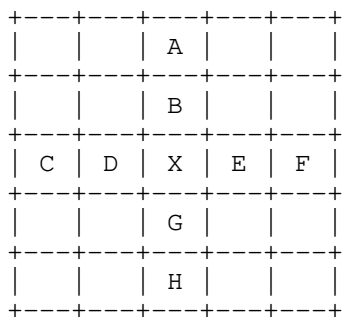


Figure 1: Filter pixel positions

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*constrain(A-X) + b*constrain(B-X) +
         c*constrain(C-X) + d*constrain(D-X) +
         e*constrain(E-X) + f*constrain(F-X) +
         g*constrain(G-X) + h*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 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 abs(x) approaches 2^d. d depends on the frame quality (qp) and is computed as

```
d = bitdepth - 4 + qp/16
```

Figure 4: Equation 4

for the luma plane and

```
d = bitdepth - 5 + qp/16
```

Figure 5: Equation 5

for the chroma planes.

The constrain function can be visualised as follows

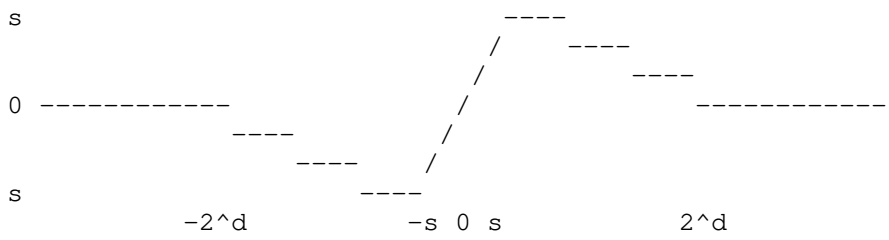


Figure 6: Graph 1

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*constrain(A-X, s, d) + 3*constrain(B-X, s, d) +
         (1*constrain(C-X, s, d) + 3*constrain(D-X, s, d) +
         (3*constrain(E-X, s, d) + 1*constrain(F-X, s, d) +
         (3*constrain(G-X, s, d) + 1*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.

	MEDIUM COMPLEXITY			HIGH COMPLEXITY		
Sequence	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

Arild Fuldseth
Cisco
Lysaker
Norway

Email: arilfuld@cisco.com

Mo Zanaty
Cisco
RTP,NC
USA

Email: 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

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 <= 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*<the number of times k has been observed> + 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]] + 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

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

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

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

   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] - floor(D*fl[k]) + k + F*e[i][k]

   where D is a negative power of two chosen such that 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 Nth 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 <= 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 log2(log2(e)*2/e) ~= 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

$$\begin{bmatrix} y0 \\ y1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x0 \\ x1 \end{bmatrix}$$

This can be implemented up to scale via

```
y0 = x0 + x1

y1 = 2*x1 - y0
```

and reversed via

```
x1 = (y0 + y1) >> 1

x0 = y0 - x1
```

Both y0 and y1 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:

$$\begin{bmatrix} y0 \\ y1 \end{bmatrix} = \frac{\sqrt{1}}{\sqrt{2}} \begin{bmatrix} \cos(t) & -\sin(t) \\ \sin(t) & \cos(t) \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x0 \\ x1 \end{bmatrix}$$

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

$$u0 = x0 + \frac{2*\cos(t) - \sqrt{2}}{\sin(t)} * x1$$

$$y1 = x1 + \frac{\sqrt{1}}{\sqrt{2}} * \sin(t)*u0$$

$$y0 = u0 + \frac{\cos(t) - \sqrt{2}}{\sin(t)} * y1$$

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]

$$\frac{1}{N} * E[e^T.e] = \frac{1}{N} * E[x^T.D^T.D.x]$$

$$= \frac{1}{N} * E[tr(D.x.x^T.D^T)]$$

$$= \frac{1}{N} * E[tr(D.Rxx.D^T)]$$

where Rxx 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

$$Rxx[i,j] = rho^{|i-j|}$$

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)x(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

[2] 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

netvc
Internet-Draft
Intended status: Informational
Expires: October 26, 2017

T. Terriberry
N. Egge
Mozilla Corporation

April 24, 2017

Coding Tools for a Next Generation Video Codec
draft-terriberry-netvc-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 October 26, 2017.

Table of Contents

1. Introduction 2
2. Entropy Coding 2
2.1. Non-binary Arithmetic Coding 4
2.2. Non-binary Context Modeling 5
2.3. Dyadic Adaptation 6
2.4. Simplified Partition Function 9
2.5. Context Adaptation 11
2.5.1. Implicit Adaptation 11
2.5.2. Explicit Adaptation 12
2.5.3. Early Adaptation 12
2.6. Simple Experiment 13
3. Reversible Integer Transforms 14
3.1. Lifting Steps 14
3.2. 4-Point Transform 17
3.3. Larger Transforms 20
3.4. Walsh-Hadamard Transforms 20
4. Development Repository 22
5. IANA Considerations 22
6. Acknowledgments 22
7. References 22
7.1. Informative References 22
7.2. URIs 23
Authors' Addresses 24

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 that NETVC 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 <= 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*<the number of times k has been observed> + 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]] + 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. This approach is described in the next section.

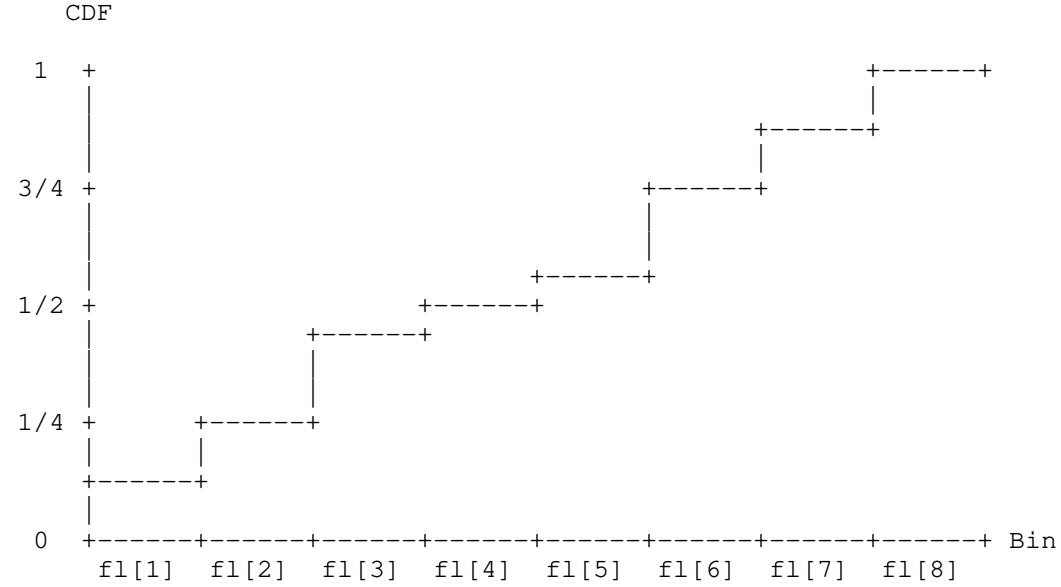
2.3. Dyadic Adaptation

The goal with context adaptation using dyadic probabilities is to maintain the invariant that the probabilities all sum to a power of two before and after adaptation. This can be achieved with a special update function that blends the cumulative probabilities of the current context with a cumulative distribution function where the coded symbol has probability 1.

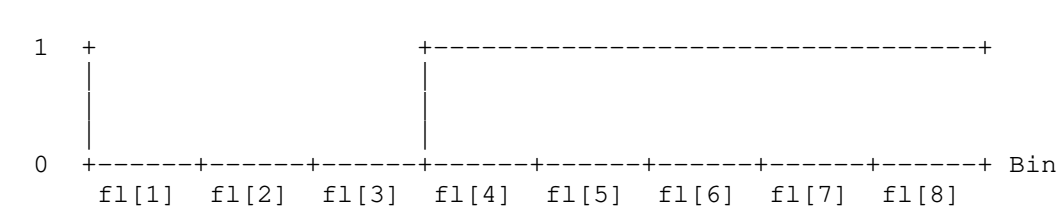
Suppose we have model for a given context that codes 8 symbols with the following probabilities:

p[0]	p[1]	p[2]	p[3]	p[4]	p[5]	p[6]	p[7]
1/8	1/8	3/16	1/16	1/16	3/16	1/8	1/8

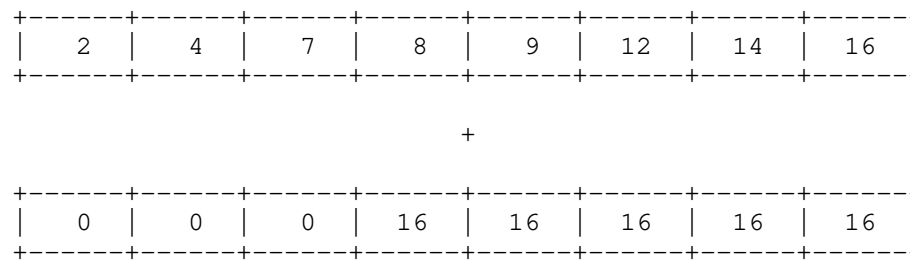
Then the cumulative distribution function is:



Suppose we code symbol 3 and wish to update the context model so that this symbol is now more likely. This can be done by blending the CDF for the current context with a CDF that has symbol 3 with likelihood 1.



Given an adaptation rate g between 0 and 1, and assuming ft = 2^4 = 16, what we are computing is:

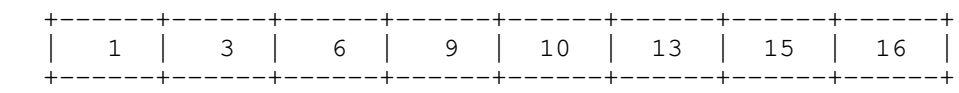


In order to prevent the probability of any one symbol from going to zero, the blending functions above and below the coded symbol are adjusted so that no adjacent cumulative probabilities are the same.

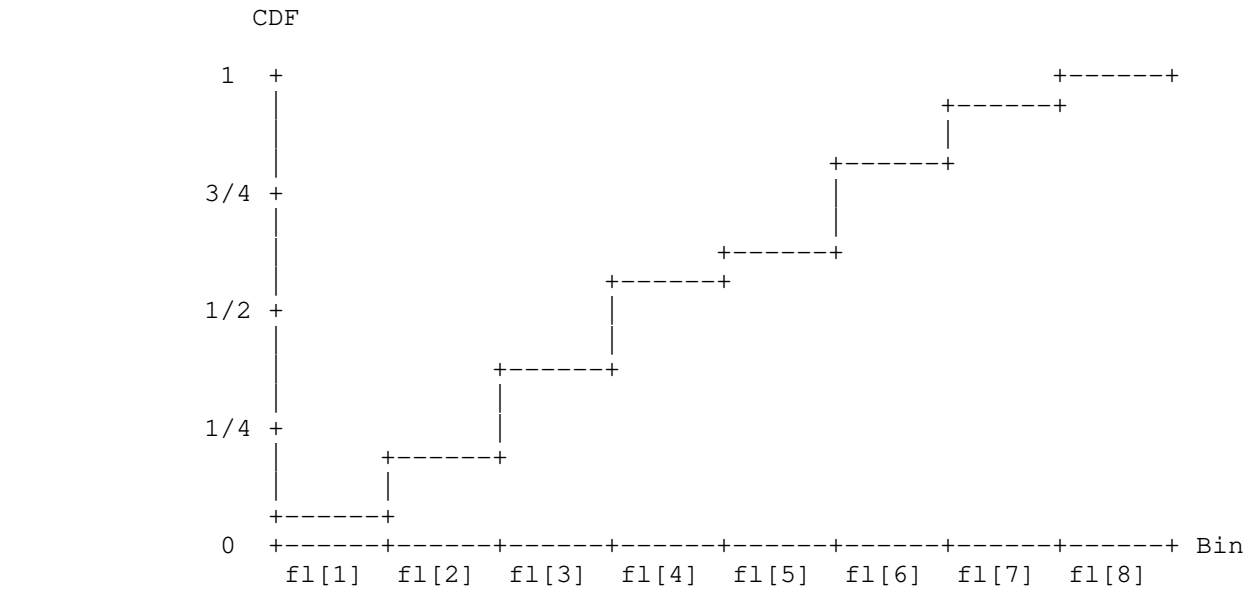
Let M be the alphabet size and 1/2^r be the adaptation rate:

```
        ( fl[i] - floor((fl[i] + 2^r - i - 1)/2^r), i <= coded symbol
fl[i] = <
        ( fl[i] - floor((fl[i] + M - i - ft)/2^r),  i > coded symbol
```

Applying these formulas to the example CDF where M = 8 with adaptation rate 1/2^16 gives the updated CDF:



Looking at the graph of the CDF we see that the likelihood for symbol 3 has gone up from 1/16 to 3/16, dropping the likelihood of all other symbols to make room.



2.4. Simplified Partition Function

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. The goal of the arithmetic coder is to partition this interval proportional to the probability of each symbol. When using dyadic probabilities, the partition point in the range corresponding to a given CDF value can be determined via

$$u[k] = \text{floor}\left(\frac{fl[k]*R}{ft}\right)$$

Since ft is a power of two, this may be implemented using a right shift by T bits in place of the division:

$$u[k] = (fl[k]*R) >> T$$

The latency of the multiply still dominates the hardware timing. However, we can reduce this latency by using a smaller multiply, at the cost of some accuracy in the partition. We cannot, in general, reduce the size of fl[k], since this might send a probability to zero (i.e., cause u[k] to have the same value as u[k+1]). On the other hand, we know that the top bit of R is always 1, since it gets renormalized with every symbol that is encoded. Suppose R contains 16 bits and that T is at least 8. Then we can greatly reduce the size of the multiply by using the formula

```
                ( (fl[k]*(R >> 8)) >> (T - 8), 0 <= k < M
        u[k] = <
                ( R,                           k == M
```

The special case for k == M is required because, with the general formula, u[M] no longer exactly equals R. Without the special case we would waste some amount of code space and require the decoder to check for invalid streams. This special case slightly inflates the probability of the last symbol. Unfortunately, in codecs the usual convention is that the last symbol is the least probable, while the first symbol (e.g., 0) is the most probable. That maximizes the coding overhead introduced by this approximation error. To minimize it, we instead add all of the accumulated error to the first symbol by using a variation of the above update formula:

```
                ( 0,                                                 k == 0
        u[k] = <
                ( R - (((ft - fl[k])*(R >> 8)) >> (T - 8)), 0 < k <= M
```

This also aids the software decoder search, since it can prime the search loop with the special case, instead of needing to check for it on every iteration of the loop. It is easier to incorporate into a SIMD search as well. It does, however, add two subtractions. Since the encoder always operates on the difference between two partition points, the first subtraction (involving R) can be eliminated. Similar optimizations can eliminate this subtraction in the decoder by flipping its internal state (measuring the distance of the encoder output from the top of the range instead of the bottom). To avoid the other subtraction, we can simply use "inverse CDFs" that natively store ifl[k] = (ft - fl[k]) instead of fl[k]. This produces the following partition function:

```
                    ( R,                               k == 0
        R - u[k] = <
                    ( (ifl[k]*(R >> 8)) >> (T - 8), 0 < k <= M
```

The reduction in hardware latency can be as much as 20%, and the impact on area is even larger. The overall software complexity overhead is minimal, and the coding efficiency overhead due to the approximation is about 0.02%. We could have achieved the same efficiency by leaving the special case on the last symbol and reversing the alphabet instead of inverting the probabilities. However, reversing the alphabet at runtime would have required an extra subtraction (or more general re-ordering requires a table lookup). That may be avoidable in some cases, but only by propagating the reordering alphabet outside of the entropy coding machinery, requiring changes to every coding tool and potentially leading to confusion. CDFs, on the other hand, are already a

somewhat abstract representation of the underlying probabilities used for computational efficiency reasons. Generalizing these to "inverse CDFs" is a straightforward change that only affects probability initialization and adaptation, without impacting the design of other coding tools.

2.5. Context Adaptation

The dyadic adaptation scheme described in Section 2.3 implements a low-complexity IIR filter for the steady-state case where we only want to adapt the context CDF as fast as the 1/2^r adaptation rate. In many cases, for example when coding symbols at the start of a video frame, only a limited number of symbols have been seen per context. Using this steady-state adaptation scheme risks adapting too slowly and spending too many bits to code symbols with incorrect probability estimates. In other video codecs, this problem is reduced by either implicitly or explicitly allowing for mechanisms to set the initial probability models for a given context.

2.5.1. Implicit Adaptation

One implicit way to use default probabilities is to simply require as a normative part of the decoder that some specific CDFs are used to initialize each context. A representative set of inputs is run through the encoder and a frequency based probability model is computed and reloaded at the start of every frame. This has the advantage of having zero bitstream overhead and is optimal for certain stationary symbols. However for other non-stationary symbols, or highly content dependent contexts where the sample input is not representative, this can be worse than starting with a flat distribution as it now takes even longer to adapt to the steady-state. Moreover the amount of hardware area required to store initial probability tables for each context goes up with the number of contexts in the codec.

Another implicit way to deal with poor initial probabilities is through backward adaptation based on the probability estimates from the previous frame. After decoding a frame, the adapted CDFs for each context are simply kept as-is and not reset to their defaults. This has the advantage of having no bitstream overhead, and tracking to certain content types closely as we expect frames with similar content at similar rates, to have well correlated CDFs. However, this only works when we know there will be no bitstream errors due to the transport layer, e.g., TCP or HTTP. In low delay use cases (video on demand, live streaming, video conferencing), implicit backwards adaptation is avoided as it risks desynchronizing the entropy decoder state and permanently losing the video stream.

2.5.2. Explicit Adaptation

For codecs that include the ability to update the probability models in the bitstream, it is possible to explicitly signal a starting CDF. The previously described implicit backwards adaptation is now possible by simply explicitly coding a probability update for each frame. However, the cost of signaling the updated CDF must be overcome by the savings from coding with the updated CDF. Blindly updating all contexts per frame may work at high rates where the size of the CDFs is small relative to the coded symbol data. However at low rates, the benefit of using more accurate CDFs is quickly overcome by the cost of coding them, which increases with the number of contexts.

More sophisticated encoders can compute the cost of coding a probability update for a given context, and compare it to the size reduction achieved by coding symbols with this context. Here all symbols for a given frame (or tile) are buffered and not serialized by the entropy coder until the end of the frame (or tile) is reached. Once the end of the entropy segment has been reached, the cost in bits for coding symbols with both the default probabilities and the proposed updated probabilities can be measured and compared. However, note that with the symbols already buffered, rather than consider the context probabilities from the previous frame, a simple frequency based probability model can be computed and measured. Because this probability model is computed based on the symbols we are about to code this technique is called forward adaptation. If the cost in bits to signal and code with this new probability model is less than that of using the default then it is used. This has the advantage of only ever coding a probability update if it is an improvement and producing a bitstream that is robust to errors, but requires an entire entropy segments worth of symbols be cached.

2.5.3. Early Adaptation

We would like to take advantage of the low-cost multi-symbol CDF adaptation described in Section 2.3 without in the broadest set of use cases. This means the initial probability adaptation scheme should support low-delay, error-resilient streams that efficiently implemented in both hardware and software. We propose an early adaptation scheme that supports this goal.

At the beginning of a frame (or tile), all CDFs are initialized to a flat distribution. For a given multi-symbol context with M potential symbols, assume that the initial dyadic CDF is initialized so that each symbol has probability 1/M. For the first M coded symbols, the CDF is updated as follows:

```
a[c,M] = ft/(M + c)

        ( fl[i] - floor((fl[i] - i)*a/ft),          i <= coded symbol
fl[i] = <
        ( fl[i] - floor((fl[i] + M - i - ft)*a/ft), i > coded symbol
```

where c goes from 0 to M-1 and is the running count of the number of symbols coded with this CDF. Note that for a fixed CDF precision (ft is always a power of two) and a maximum number of possible symbols M, the values of a[c,M] can be stored in a M*(M+1)/2 element table, which is 136 entries when M = 16.

2.6. 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

$$y0 = x0 + x1$$

$$y1 = 2*x1 - y0$$

and reversed via

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

$$x0 = y0 - x1$$

Both y0 and y1 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:

```
                 ___
[ y0 ]     / 1   [ cos(t) -sin(t) ] [ 1  0 ] [ x0 ]
[    ] =  / --- [                ] [     ] [    ]
[ y1 ]   v   2   [ sin(t)  cos(t) ] [ 0  2 ] [ x1 ]
```

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

$$u0 = x0 + \frac{2*\cos(t) - \overline{v2}}{\sin(t)} * x1$$

$$y1 = x1 + \sqrt{\frac{1}{v}} * \frac{1}{2} * \sin(t)*u0$$

$$y0 = u0 + \frac{\cos(t) - \overline{v2}}{\sin(t)} * y1$$

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]

$$\frac{1}{N} * E[e^T.e] = \frac{1}{N} * E[x^T.D^T.D.x]$$

$$= \frac{1}{N} * E[tr(D.x.x^T.D^T)]$$

$$= \frac{1}{N} * E[tr(D.Rxx.D^T)]$$

where Rxx 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

$$Rxx[i,j] = rho^{|i - j|}$$

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)x(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.

[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

[2] https://people.xiph.org/~tterribe/daala/ec_test0/ec_test.c

[3] https://git.xiph.org/daala.git

[4] https://xiph.org/daala/

Authors' Addresses

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

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

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

Phone: +1 650 903-0800
Email: negge@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

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 [KO95] 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.

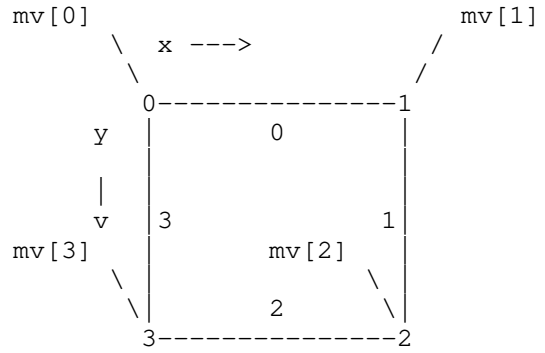


Figure 1: Vertex and Edge Indices for a Block

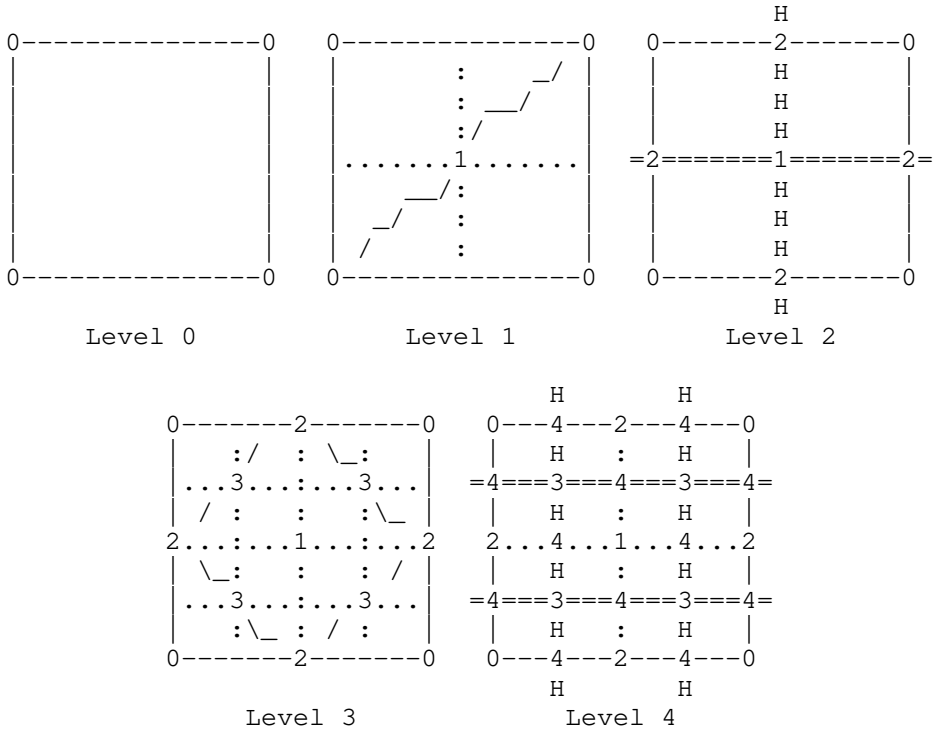
In a block with MVs at all four corners, we use normal bilinear weights to blend the predictions from each MV. The bilinear weights for each vertex, w[k], at a pixel location (x, y) are defined as

```
w[0] = (1 - x)*(1 - y)
w[1] = x*(1 - y)
w[2] = x*y
w[3] = (1 - x)*y
```

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 aditional 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.

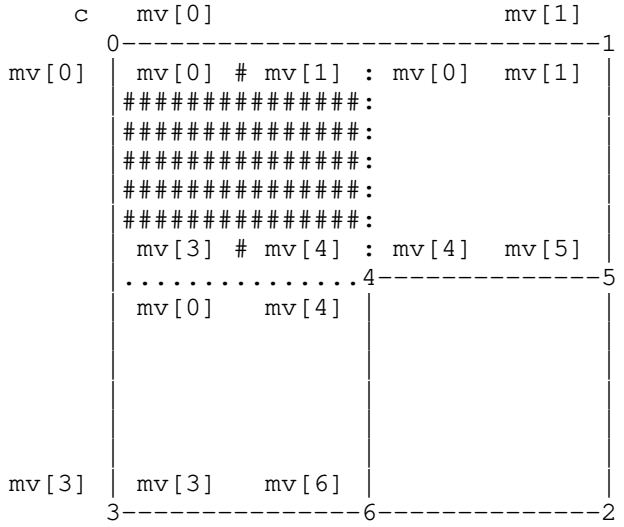


Figure 3: Interpolation Setup for Unsplit Edges

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

$$s[c] = w[c] + 0.5*w[c \ (+) \ 1]$$
$$s[c \ (+) \ 1] = 0.5*w[c \ (+) \ 1]$$

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),

```
          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]
```

This defintiion 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 admissable 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 ot 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

$$J = D + \lambda R$$

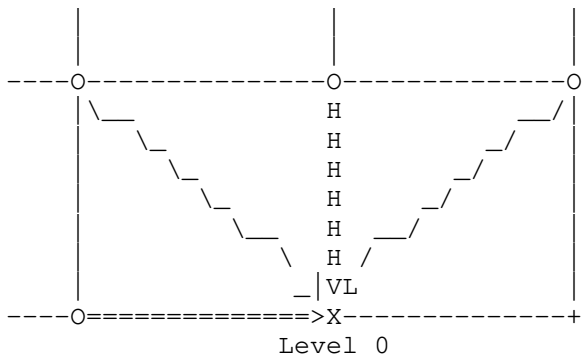
The value of lambda is obtained directly from the target quantizer setting.

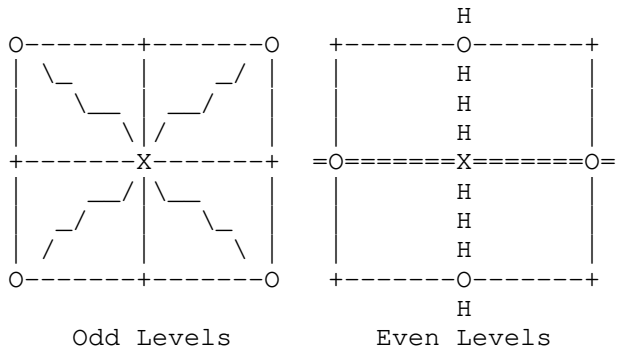
We use the Sum of Absolute Differences (SAD) in the luma plane as our distortion metric for the first three stage of the search, and use the Sum of Absolute Transformed Difference (SATD) during a final subpel refinement stage (with an appropriate adjustment to lambda).

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.terriberry-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). Finall, 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'} U A_w' \ 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 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' \ 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} U A_p, meaning one can add d to the nodes in A_w' and then subtract it from the nodes in {p} U 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 +/- 7x7 pixels. In our framework, this requires examining 3x9**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 evaluted 25 times, for a total of 100 SAD calculations per block. This is nearly as many as full search BMA with a +/- 6x6 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 samll 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 eigth-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

eigth-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 eigth-pel resolution. If chroma is decimated (for 4:2:0 video) and eigth-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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 22, 2016

JM. Valin
Mozilla

March 21, 2016

Directional Deringing Filter
draft-valin-netvc-deringing-01

Abstract

This document describes a deringing filter that takes into account the direction of edges and patterns being filtered. The filter works by identifying the direction of each block and then adaptively filtering along the identified direction. In a second pass, the blocks are also filtered in a different direction, with more conservative thresholds to avoid blurring edges. The proposed deringing filter is shown to improve the quality of both Daala and the Alliance for Open Media (AOM) 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 22, 2016.

Table of Contents

1. Introduction 2
2. Direction Search 2
3. Directional Filter 3
4. Second Stage Filter 4
5. Setting Thresholds 5
6. Superblock Filtering 6
7. Results 6
8. Conclusion 7
9. IANA Considerations 7
10. Security Considerations 7
11. Informative References 7
Author's Address 8

1. Introduction

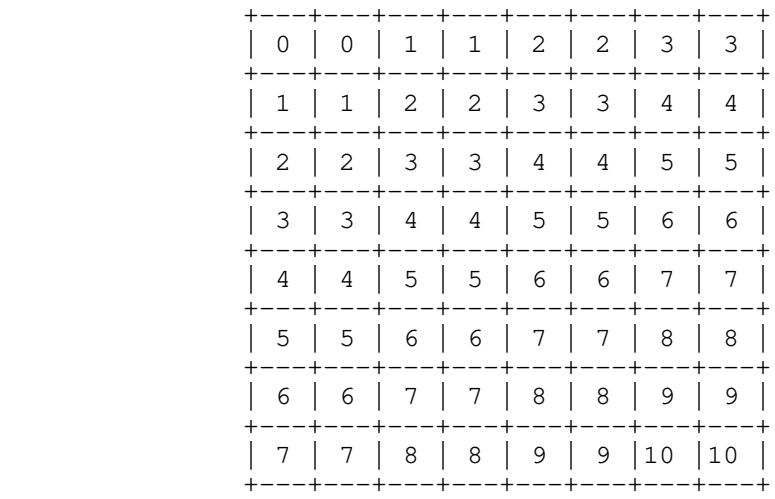
This document describes a deringing filter that takes into account the direction of edges and patterns being filtered. The filter works by identifying the direction of each block and then adaptively filtering along the identified direction. In a second pass, the blocks are also filtered in a different direction, with more conservative thresholds to avoid blurring edges. The deringing filter is implemented for both Daala and the Alliance for Open Media (AOM) codec.

2. Direction Search

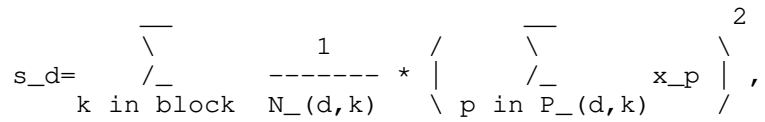
The first step is to divide the image into blocks of fixed or variable size. Variable-size blocks make it possible to use large blocks on long, continuous edges and small blocks where edges intersect or change direction. A fixed block size is easier to implement and does not require signaling the sizes on a block-by-block basis. For this work, we consider a fixed block size of 8x8.

Once the image is divided into blocks, we determine which direction best matches the pattern in each block. One way to determine the direction is to minimize mean squared difference (MSD) between the input block and a perfectly directional block. A perfectly directional block is a block for which each line along a certain

direction has a constant value. For each direction, we assign a line number to each pixel, as shown below.



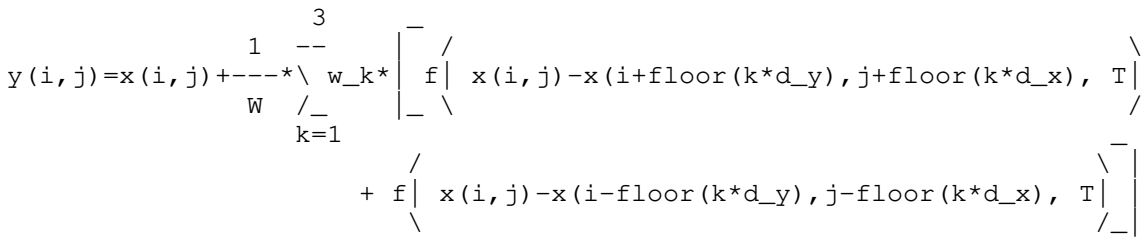
For each direction d, we compute the value s_d, which is equal to a direction-independent offset minus the MSD (see [Deringing-Note] for detauls) as:



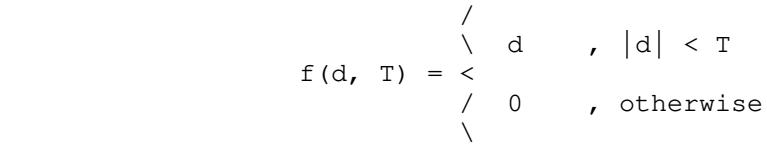
where x_p is the value of pixel p, P_(d,k) is the set of pixels in like k along direction d, and N_(d,k) is the cardinality of P_(d,k). From there, the direction is computed as the value of d that maximizes s_d.

3. Directional Filter

The directional filter for pixel (i,j) is defined as the 7-tap non-linear filter



where d_x and d_y define the direction, W is a constant normalizing factor, T is the filtering threshold for the block, and f(d,T) is defined as



The direction parameters are shown in the table below. The weights w_k can be chosen so that W is a power of two. For example, Daala currently uses w=[3 2 2] with W=16. Since the direction is constant over 8x8 blocks, all operations in this filter are directly vectorizable over the blocks.

Direction	d_x	d_y
0	1	-1
1	1	-1/2
2	1	0
3	1	1/2
4	1	1
5	1/2	1
6	0	1
7	-1/2	1

Table 1

4. Second Stage Filter

The 7-tap directional filter is sometimes not enough to eliminate all ringing, so we use an additional filtering step that operates across the direction lines used in the first filter. Considering that the input of the second filter has considerably less ringing than the input of the second filter, and the fact that the second filter risks

blurring edges, the position-dependent threshold T_2(i,j) for the second filter is set lower than that of the first filter T. The filter structure is the same as the one used for the directional filter. The direction parameters for the second stage filter are shown in the table below and the filter weights are w=[1 1] with W=16/3.

Direction	d_x	d_y
0	1	0
1	0	1
2	0	1
3	0	1
4	1	0
5	1	0
6	1	0
7	1	0

Table 2

5. Setting Thresholds

The thresholds T and T_2 must be set high enough to smooth out ringing artefacts, but low enough to avoid blurring important details in the image. Although the ringing is roughly proportional to the quantization step size Q, as the quantizer increases the error grows slightly less than linearly because the unquantized coefficients become very small compared to Q. As a starting point for determining the thresholds, Daala uses a power model of the form T_0=level*alpha_1*Q^beta, with beta=0.842, and where alpha_1 depends on the input scaling. The level is a threshold adjustment coded for each superblock (64x64). In the AOM codec, a global threshold is selected by the encoder instead of using a function of the quantizer, so T_0=level*global_level.

Another factor that affects the optimal filtering threshold is the presence of strong directional edges/patterns. These can be estimated from the s_d parameters computed in the direction search as delta=s_(d_opt)-s_(d_ortho), where d_ortho=d_opt+4 (mod 8). We compute the direction filtering threshold for each block as

$$T = T\_0 * \max\left(\frac{1}{2}, \min\left(3, \text{alpha\_2} * (\text{delta})^{1/6}\right)\right),$$

where alpha_2 also depends on the input scaling. For the second filter, we use a more conservative threshold that depends on the amount of change caused by the directional filter.

$$T\_2(i,j) = \min\left(T, \frac{T}{3} + |y(i,j)-x(i,j)|\right).$$

As a special case, when the pixels corresponding to the 8x8 block being filtered are all skipped, then T=T_2=0, so no deringing is performed.

6. Superblock Filtering

The filtering is applied one superblock at a time, in a way that depends on the level. In Daala, the level can take one of 6 values: 0, 0.5, 0.7, 1.0, 1.4, 2.0, where a level of zero disables the deringing filter for the current superblock. The level is the only information coded in the bitstream by the deringing filter. On keyframes, it is entropy-coded based on the neighbor values. On inter-predicted frames, the level is only coded for superblocks that are not skipped and is entropy-coded based on a single adapted probability distribution (no context from the neighbors). Superblocks where no level is coded have deringing disabled. Similarly, any skipped block within a superblock has deringing disabled, even if it is signaled enabled for the superblock.

The level of the deringing filter in AOM is handled similarly, except that only four levels are currently available and there is no entropy coding yet.

The deringing process sometimes reads pixels that lie outside of the superblock being processed. When these pixels belong to another superblock, the filtering always uses the unfiltered pixel values -- even for the second stage filter -- so that no dependency is added between the superblocks. This makes it possible -- in theory -- to filter all superblocks in parallel. When the pixels used for a filter lie outside of the viewable image, we set f(d,T)=0.

7. Results

The deringing filter described here has been implemented for the Daala [Daala-website] codec. It is available from the Daala Git repository [Daala-Git]. We tested the deringing filter on the Are We Compressed Yet [AWCY] ntt-short1 set over the 0.025 bit/pixel to 0.1 bit/pixel range, corresponding to a 1080p30 bitrate of 1.5 Mbit/s to 6 Mbit/s. The Bjontegaard-delta [I-D.daede-netvc-testing] rate reduction over that range was 6.5% for PSNR, 4.7% for PSNR-HVS, 5.6%

for SSIM and -6.0% (regression) for FAST-SSIM. Visual inspection confirmed that the quality is indeed improved, despite the regression in the FAST-SSIM result.

In AOM for the ntt-short1 set, the medium bitrate (0.02 to 0.06 bit/pixel) Bjontegaard-delta improvement is 2.5% for PSNR, 1.5% for PSNR-HVS, 1.5% for SSIM, and -3.8% (regression) on FAST-SSIM. The high bitrate (0.06 to 0.2 bit/pixel) Bjontegaard-delta improvement is 2.0% for PSNR, 0.8% for PSNR-HVS, 1.3% for SSIM, and -3.1% (regression) on FAST-SSIM.

The smaller improvement for AOM compared to Daala may be due to the newly integrated code not being mature, but also to the fact that some features in Daala tend to cause more ringing. These features include lapped transforms, quantization matrices, perceptual vector quantization, overlapped block motion compensation (OBMC), and activity masking.

8. Conclusion

We have demonstrated an effective algorithm to remove ringing artefacts from coded images and videos. The proposed filter takes into account the directionality of the patterns it is filtering to reduce the risk of blurring.

9. IANA Considerations

This document makes no request of IANA.

10. Security Considerations

This draft has no security considerations.

11. Informative References

[AWCY] "Are We Compressed Yet?", Xiph.Org Foundation , <https://arewecompressedyet.com/>.

[Daala-Git] "Daala Git repository", Xiph.Org Foundation , <http://git.xiph.org/?p=daala.git;a=summary>.

[Daala-website] "Daala website", Xiph.Org Foundation , <https://xiph.org/daala/>.

[Deringing-Note]
Valin, JM., "The Daala Directional Deringing Filter", arXiv:1602.05975 [cs.MM] , <http://arxiv.org/abs/1602.05975>.

[I-D.daede-netvc-testing]
Daede, T. and J. Jack, "Video Codec Testing and Quality Measurement", draft-daede-netvc-testing-02 (work in progress), October 2015.

Author's Address

Jean-Marc Valin
Mozilla
331 E. Evelyn Avenue
Mountain View, CA 94041
USA

Email: jmvalin@jmvalin.ca

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.

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:

$$g = Q_g \, \mathrm{gamma}^{b} ,$$

where gamma is the gain quantization index, b=1/(1-2*a) 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:

$$m = \mathrm{argmax}_i \, | r_i |$$
$$s = \mathrm{sign}(r_m)$$

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

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

with C ~= 4.2. The distortion due to the gain is

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

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)~=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)~=f(theta). A possible choice of f(theta) is a quadratic function of the form:

$$f(theta) = a1\ theta - a2\ theta^2.$$

where a1 and a2 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 r1 and r2, we introduce s1=(r1+r2)/2 and s2=(r1-r2)/2. We start by using s1 as a reference, apply the Householder reflection to both x and s2, and evaluate theta1. From there, we derive a second Householder reflection from the reflected version of s2 and apply it to z. The result is that the theta2 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

$$E(|y\_n|) = \text{alpha} * \frac{K\_n}{N - n},$$

where K_n is the number of pulses left after encoding coeffients 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

$$E(|\text{run}|) = \text{beta} * \frac{N - n}{K\_n},$$

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>.

[PVQ-demo] Valin, JM., "Daala: Perceptual Vector Quantization (PVQ)", November 2014, <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