SMPTE Meeting Presentation

A Technical Overview of VP9 – the Latest Open-source Video Codec

Debargha Mukherjee, Ph. D., Software Engineer

Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043; debargha@google.com

Jingning Han, Ph. D., Software Engineer

Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043; jingning@google.com

Jim Bankoski, Engineering Manager

Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043; jimbankoski@google.com

Ronald Bultje, Ph. D., Software Engineer

Two Sigma, 379 W. Broadway, 5th floor, New York, NY 10012; rsbultje@gmail.com

Adrian Grange, Software Engineer

Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043; agrange@google.com

John Koleszar, Software Engineer

Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043; jkoleszar@google.com

Paul Wilkins, Senior Staff Software Engineer

Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043; paulwilkins@google.com

Yaowu Xu, Ph. D., Engineering Manager

Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043, yaowu@google.com

Written for presentation at the

SMPTE 2013 Annual Technical Conference & Exhibition

Abstract. *Google has recently finalized a next generation open-source video codec called VP9, as part of the libvpx repository of the WebM project (http://www.webmproject.org/). Starting from the VP8 video codec released by Google in 2010 as the baseline, various enhancements and new tools were added, resulting in the next-generation bit-stream VP9. The bit-stream was finalized with the exception of essential bug-fixes, in June 2013. Prior to the release however, all technical developments in fact were being conducted openly in the public experimental branch of the repository for many months. This paper provides a brief technical overview of the coding tools included in VP9, along with coding performance comparisons with other state-of-the-art video codecs – namely H.264/AVC and HEVC – on standard test sets. While a completely fair comparison is impossible to conduct because of the limitations of the respective encoder implementations, the tests show VP9 to be quite competitive with main-stream state-of-the-art codecs.*

Keywords. Video compression, motion-compensated prediction, transforms, quantization, entropy coding, quad-tree partitioning, DCT, ADST.

Introduction

The explosive growth in consumption of video over the Internet, including professional and amateur video-on-demand, as well as conversational video content, continues to stress the limits of available bandwidth and delivery infrastructures. To address this, ISO/IECMPEG/ITU-T VCEG developed a next generation video coding standard called HEVC (High-Efficiency Video Coding) [1] starting from the current generation codec H.264/AVC [2] as the baseline. Likewise, Google embarked on a project in late 2011 to develop their next-generation open-source video codec VP9 starting from predecessor VP8 [3] as the baseline.. The coding tools in VP9 have since matured, and the bit-stream was finalized in June 2013. This paper provides a technical overview of the tools that are in the final VP9 bit-stream. Coding results comparing VP9, to H.264/AVC and HEVC are also presented.

The WebM Project

A key factor in the success of the web is that its core technologies such as HTML, HTTP, and TCP/IP are open and freely implementable. Though video is also now core to the web experience and consumes the majority of all Internet traffic, there is unfortunately no open and free video format that is on par with the leading commercial choices. To that end, Google started the WebM project [4] in 2010, as a community effort to develop an open web media format. WebM is an open, royalty-free, media file format that defines not only the file container structure, but also the video and audio formats. Specifically, WebM files consist of video streams compressed with the open-source VP8 video codec [3] and audio streams compressed with the open-source Vorbis audio codec [5], in a file structure based on the Matroska container [6].

However, to cope with the ever-increasing demand for high-quality video content for consumption on the web, it was obvious that VP8 could not be the end of the road as far as video coding technology is concerned. In late 2011, Google started working towards a next-generation video codec, aka VP9, with the explicit goal of creating a format that produces a much more compact bit-stream than predecessor VP8 especially for high-definition content, with only a modest increase in decoding complexity. All technical development related to VP9 was conducted openly in the public experimental branch of the libvpx repository of the WebM project since mid-2012. Like the rest of the WebM project, VP9 is an open-source software project that anyone may participate in and contribute to.

VP9 Coding Tools

A large part of the advances made by VP9 over VP8 is natural progression from current generation video codecs to the next. Yet the technical challenge is to ensure that all coding tools work together in a highly efficient manner. This section guides the reader through the main tools included in VP9.

Prediction Block-sizes

A large part of the coding efficiency improvements achieved in VP9 can be attributed to incorporation of larger prediction block sizes. VP9 introduces super-blocks (SB) of size up to 64x64 and allows breakdown using a recursive decomposition all the way down to 4x4 blocks as follows. As shown in Figure 1, at each square block level within the prediction structure, there are four options. Three of these represent endpoints after which no further breakdown is allowed and include: treating the parent block as a single square block, or two vertically

adjacent rectangular blocks or two horizontally adjacent rectangular blocks. The fourth option splits the parent square block into four square blocks of size half the parent blocksize. The partition process then repeats in a recursive manner all the way down, if necessary, to the 4x4 block size. Thus there are a total of 13 different endpoint block sizes for the purpose of defining various prediction signals: 64x64, 64x32, 32x64, 32x32, 32x16, 16x32, 16x16, 16x8, 8x16, 8x8, 8x4, 4x8, and 4x4.

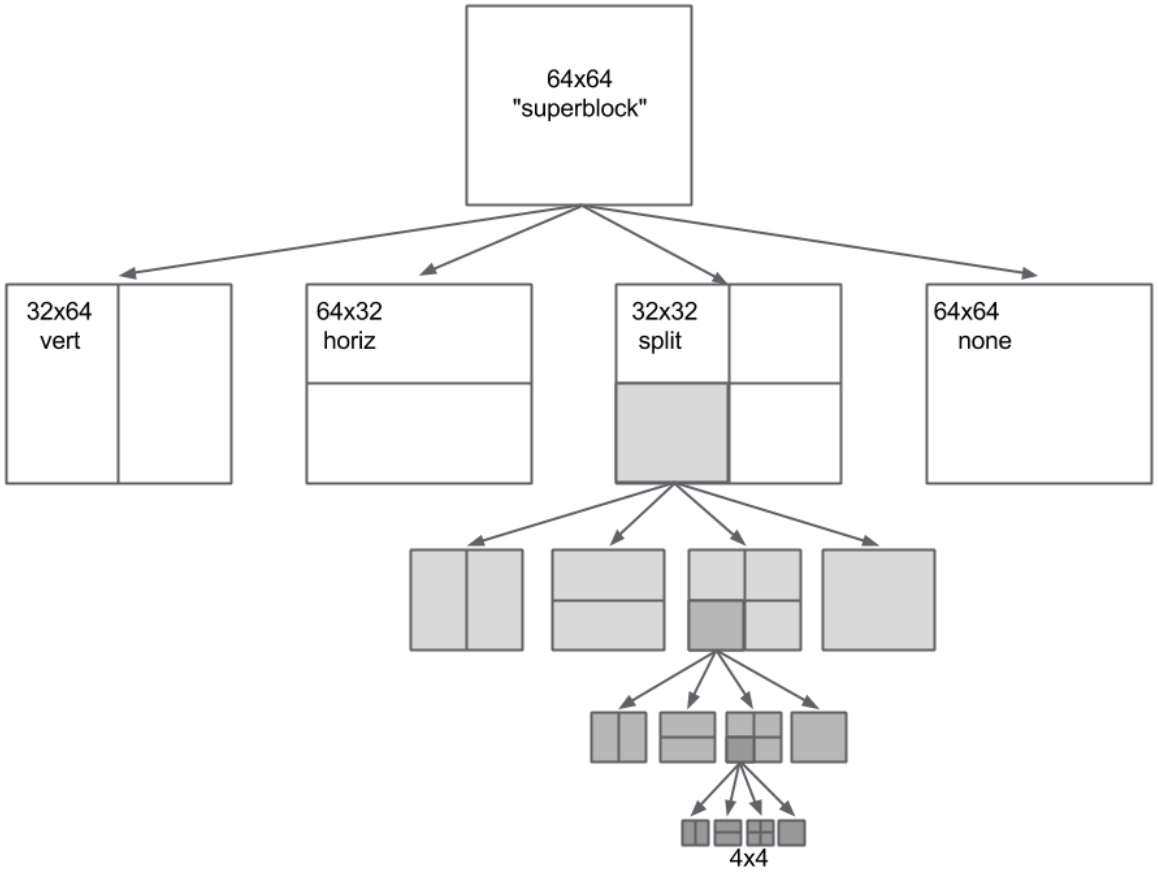

Figure 1. Recursive super block partitioning structure in VP9

At any endpoint from 64x64 down to 4x4 the following prediction signals may be encoded:

1. A prediction mode.
2. Prediction reference frame(s) to be used if an inter mode is used (Note: in case of compound prediction two reference frames are conveyed).
3. Motion vector(s) for selected reference frame(s) if an inter mode is used.
4. Subpel prediction filter selection. [See below for section on subpel prediction filter]
5. A skip flag indicating if there is any non-zero coefficient for residual signal.
6. The transform size to be used when coding the residual. [See below for section on transform size]
7. The segment (if any) to which the block belongs. [See below for section on segmentation].

For endpoints below 8x8 (i.e., blocksizes 4x8, 8x4 and 4x4), all the blocks inside an 8x8 on-grid block share all the encoded information above excepting 1 and 3 above. In other words, the prediction mode, and the motion vectors for the selected reference frame(s) if the prediction mode is inter, are further encoded per endpoint block smaller than 8x8, but the other five attributes are shared at the 8x8 level.

Sub-blocks within a larger parent block are encoded in raster order and each is fully reconstructed before encoding of the next sub-block commences. Thus, reconstructed pixel data from previously encoded blocks/sub-blocks is available for predicting neighboring block/sub-blocks that are encoded subsequently.

Prediction Modes

Intra modes

VP9 supports a set of 10 Intra prediction modes for block sizes ranging from 4x4 up to 32x32:

- DC_PRED (DC prediction),
- TM_PRED (True-motion prediction),
- H_PRED (Horizontal prediction),
- V_PRED (Vertical prediction), and
- D27, D153, D135, D117, D63, D45 – 6 oblique directional modes corresponding approximately to angles 27, 153, 135, 117, 63, and 45 degrees (counter-clockwise measured against the horizontal axis).

The horizontal, vertical and oblique directional prediction modes involve copying (or estimating) pixel values from surrounding blocks into the current block along the angle specified by the prediction mode.

For blocks using Intra prediction modes, the coding (encoding and decoding) process operates on units of the transform size used. For example, when a 16x16 Intra coded block is coded using 4x4 transforms, each 4x4 block in raster scan order within the 16x16 area goes through entropy decoding, prediction and reconstruction before starting the next 4x4 block. Subsequent 4x4 blocks can thus use reconstructed pixel values from previous blocks for Intra prediction.

Inter modes

VP9 supports a set of 4 inter prediction modes for block sizes ranging from 4x4 up to 64x64 pixels:

- NEARESTMV,
- NEARMV,
- ZEROMV, and
- NEWMV.

Previously coded motion vectors in the neighborhood of the current prediction unit is surveyed to define two candidate vectors – NEARESTMV and NEARMV – which can be used directly in the first two modes above. NEARESTMV and NEARMV refer respectively to the most likely and the second most likely motion vector in the neighborhood. ZEROMV mode indicates usage of the (0, 0) motion vector while NEWMV indicates that a new motion vector will be encoded in the

stream. In the latter case, the NEARESTMV motion vector is also used as the predictor. The motion vectors for chroma blocks of sizes up to 8x8 are calculated as the average of those within the corresponding luminance block.

Each INTER frame may make reference to three reference frame buffers selected from a pool of up to eight stored reference frames as signaled by a syntax element in the frame header. Each decoded frame may update any subset of these eight reference frames as signaled by another syntax element in the frame header. The choice of which reference frames to use and update, depends on the strategy used by a compliant encoder.

Some encoded frames, known as Alternative Reference Frames (ARFs) may be used purely for reference and are never actually displayed. These frames are identified as *invisible* using a flag in the frame header. A possible use of an ARF is to code a future frame out-of-order to help improve coding efficiency by incorporating forward prediction, but is not limited to that. The framework is flexible enough to be adopted for more sophisticated encoding choices, for instance, to achieve a form of temporally hierarchical prediction.

Subpel interpolation

The filters used for subpel interpolation for fractional motion are critical to performance of a video codec. In VP9, the maximum motion vector precision supported is ⅛-pel, with the option of switching between ¼-pel and ⅛-pel precision by a frame level flag. If ⅛-pel precision is on in a frame however, it is only used for small motion, as indicated by the magnitude of the reference motion vector. For larger motion - indicated by a larger reference - there is often motion blur, which obviates the need for higher precision motion. Note that for prediction of chrominance blocks with 4:2:0 color sampling, the motion vector precision increases by 1 bit, so that the highest precision required is 1/16-pel.

For the actual interpolation for subpel motion, VP9 supports a family of three 8-tap filters – called EIGHTTAP, EIGHTTAP_SHARP and EIGHTTAP_SMOOTH – along with a simpler Bilinear filter, each with 1/16-pel accuracy. The latter is for use in fast encode/decode scenarios. The specific filter used can be selected at a frame level, but there is also the option of indicating one of three 8-tap filter types to use with every super-block. Each filter is represented in 7-bit precision.

Transform Types

VP9 uses three types of transforms: Discrete Cosine Transform (DCT), a modified Asymmetric Discrete Sine Transform (ADST) [7], and Walsh-Hadamard Transform (WHT). Note that the actual variant of ADST that is used is not exactly what was proposed in [7], but a modified version for which a butterfly implementation is readily possible. This variant of ADST has also been referred to as DST-IV in the literature.

Specifically, all inter coded blocks in VP9 are coded using one of four 2D-DCTs of size 4x4, 8x8, 16x16 and 32x32. Intra coded blocks in VP9 additionally can be coded using a 4-pt, 8-pt and 16-pt modified Asymmetric Discrete Sine Transforms (ADSTs) [7], in combination with a 1-D DCT of the same size – depending on the chosen intra prediction mode – to form a 2-D hybrid transform. The possible transform combinations used in the vertical and horizontal directions for INTRA coded residuals are thus: (ADST, ADST), (ADST, DCT), (DCT, ADST) and (DCT, DCT), for 4x4, 8x8 and 16x16 blocksizes, as well as the 32x32 DCT. Note that the (DCT, DCT) combination at a given block size is the same as a 2D-DCT. A final transform supported by VP9 is the 4x4 Walsh Hadamard Transform (WHT), which is used only at the lowest quantizer value

to provide lossless encoding. Thus, there are a total of 14 possible 2D-transforms supported in VP9, each of square size, as follows:

- 4x4 WHT
- 4x4 (DCT, DCT)
- 4x4 (DCT, ADST)
- 4x4 (ADST, DCT)
- 4x4 (ADST, ADST)
- 8x8 (DCT, DCT)
- 8x8 (DCT, ADST)
- 8x8 (ADST, DCT)
- 16x16 (DCT, DCT)
- 16x16 (DCT, ADST)
- 16x16 (ADST, DCT)
- 32x32 (DCT, DCT)

Transform type, i.e. DCT, or Hybrid of DCT/ADST or WHT, used in coding any block is deterministically dependent on the prediction mode and transform size used for the block, and is not coded in the bit-stream. However transform size is explicitly encoded. A frame level flag indicates one of four maximum transform sizes allowed in the frame – called ONLY_4X4, ALLOW_8X8, ALLOW_16X16, and ALLOW_32X32 – or a special mode called TX_MODE_SELECT that allows the transform size to be specified for every prediction block. For the first 4 modes above, the transform size of each block is the largest allowed transform size that is less than or equal to the prediction blocksize. For example, a 8x8 block in a frame with mode ALLOW_16X16 is coded using 8x8 transform. The TX_MODE_SELECT mode produces the best results, but at the expense of additional search for the best transform size.

Entropy Coding and Adaptation

The VP9 bit-stream is arithmetic encoded. Given a symbol from any n-ary alphabet, a binary tree is constructed with n-1 non-leaf nodes, and a binary arithmetic encoder is run on each such node to encode a symbol. The probabilities at each node use 8-bit precision and are in [1, 255] – corresponding to actual probabilities in the range 1/256 – 255/256. The set of n-1 probabilities for coding the symbol is referred to as the entropy coding context of the symbol. Most of the coding choices conveyed in the VP9 bit-stream - modes, motion vectors, reference frames, transform sizes, prediction residuals, etc. - use this methodology.

Video content is inherently non-stationary in nature. So a critical element in any codec is a mechanism to track the real statistics of various symbols encoded through the entropy coding contexts used. Most modern codecs today use a form of backward adaptation where probabilities are updated based on accumulated symbol counts while encoding/decoding on a per symbol basis. This strategy however produces a burden on decoding computation because intermediate computations to update the probabilities are necessary for each symbol decoded. VP9, in contrast, allows a combination of frame-level forward and backward updates to these probabilities to strike a good balance between decoding performance and coding efficiency.

The coding contexts in VP9 are initialized to a set of defaults at every key frame, where-after they propagate temporally based on forward and backward updates as conducted. A syntax element in the frame header first conveys forward updates from a set of baseline probabilities held in the context at the start of the frame. This syntax allows certain node probabilities to be updated selectively while leaving others unchanged. Since forward updates for all nodes would be too expensive to convey in the bit-stream, only a small subset of probabilities over a frame are practically conveyed. All symbols in the frame are then encoded/decoded based on the updated set of probabilities using a non-adaptive arithmetic encoder. At the end of the frame, there is a backward adaptation step conducted to update the probabilities again based on actual accumulated counts for various symbols transmitted over the frame. In particular, the forward updates are discarded, and the backward updated values are propagated as the baseline values for the next frame encoded.

Coefficient coding

Further details on the specifics of forward and backward adaptation, and the entropy coding of each symbol type, is beyond the scope of this paper. However, we do present the specifics of coefficient encoding – which at most bit-rates for most videos is the largest proportion of the bit-stream. Quantized coefficients at sizes 4x4, 8x8, 16x16 and 32x32 are scanned in a pre-defined scan order that depends on the transform type and size. For each coefficient a token symbol is transmitted to indicate the value or value range of its magnitude, followed by the sign bit and any extra bits if needed. The magnitude tokens transmitted are: “0”, “1”, “2”, “3”, “4” “5-6”, “7-10”, “11-18”, “19-34”, “35-66”, “67+” and EOB (end of block), and is coded using a binary tree as shown in Figure 2. Note each gray circle in the figure is an internal node where an arithmetic coder runs with a certain node probability value. The tree has 11 internal nodes.

For each token coded in scan order, the coding context (node probabilities) used is chosen based on a combination of:

- Transform_size: whether it is a 4x4, 8x8, 16x16 or 32x32 transform block
- Block_type: whether it is a Y or UV block, 2 possible values
- Ref_type: whether it is an INTRA or INTER coded block, 2 possible values
- Coef_band: based on position of the coefficient in the block, 6 possible values where the band assignment is provided in a predefined table for given transform block size and type.
- Prev_token: based on magnitudes of previously encountered coefficients in scan order in the same block, in the frequency neighborhood of the current coefficient location; 6 possible values. Specifically, prev_token context is a function of the coefficients to the left and top of the current coefficient if they exist and have already been encoded in scan order.

Thus there are 576 different contexts, each representing a 11-ary coding context. Needless to say, the forward and backward entropy updates of such a large coding context, necessitates maintaining a large number of counts, which is a non-trivial burden - particularly for the decoder. To alleviate the problem, VP9 uses a modeled update approach, where only the 3 top nodes of the tree are actually updated based on real statistics. The remaining 8 nodes covering less frequent higher magnitude tokens are simply derived from the probability of the “peg node” as depicted in the figure.

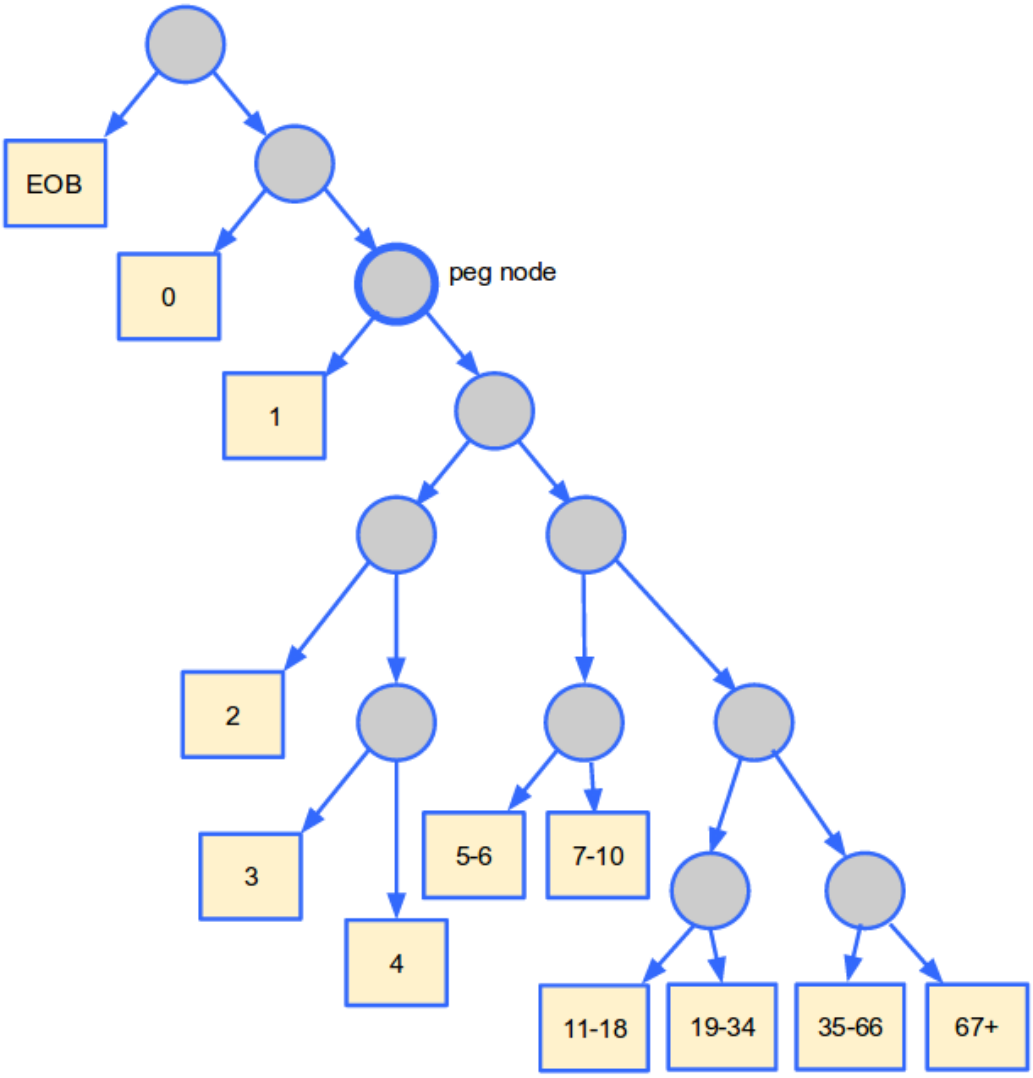

Figure 2. Coefficient token coding tree

Specifically, a fixed look up table is maintained that provides a vector of 8 probabilities given the left-probability of the peg node prob(peg) in 8-bit precision. The look up table is derived from a modeled distribution of the quantized coefficients in a block. Specifically, the model used is a symmetric Pareto distribution that is heavier tailed than a generalized Gaussian and matches real data closer. The cdf of the distribution is:

$$cdf(x) = 0.5 + 0.5 * sgn(x) * [1 - \{a/(a + |x|)\} ^ b], \text{ with } b = 8.$$

Coefficients from this model are assumed to be quantized using a uniform quantizer with normalized step size of 1. The method used to derive the values of the probabilities in the look-up-table is as follows. Assume that the left-probability value of the peg node - denoted prob(peg) - is known, in 8-bit precision in the range [1, 255] . Since this probability is really 256 * prob(1)/(1 - prob(0)), assuming a unit-step normalized quantizer, the value of the parameter a can be solved numerically. Once a is obtained, the probabilities of all other quantization bins and therefore the node probabilities can be computed and stored in the look-up-table as an

8-ary vector in 8-bit precision, indexed by the 8-bit value of prob(peg). To further reduce the cost of storing this table in the encoder and decoder, only 8-ary vectors corresponding to odd values of prob(peg) = 1, 3, 5, …, 253, 255 are stored in the table. The vectors for the even values are interpolated linearly from the nearest odd values in the table.

Loop-Filtering

Because VP9 uses a variety of transform sizes ranging from 4x4 up to 32x32, the loop filter was designed to eliminate blocking artifacts on boundaries of all different sizes of blocks. On a reconstructed VP9 frame, the loop filter operates on a raster scan order of 64x64 superblocks (SB64) within the video frame. For each SB64, the loop filter filters all internal transform block boundaries and the boundaries adjacent to SB64 above and to the left of the current SB64 in two passes – the first for all vertical boundaries, and the second for all the horizontal boundaries. In general, filtering is not applied on boundaries of a 4x4 or 8x8 or 16x16 area unless those boundaries are also boundaries of any transform block, but filtering is always applied on boundaries of all SB32s and SB64s. Within a prediction block, filtering is applied on a transform block boundary if the transform block has at least one non-zero coefficient or the block is intra coded.

Specifically, 3 different filters are used depending on the transform size and a flatness measure around an edge. These include a 15-tap smoothing filter, a 7-tap smoothing filter and a 4-tap adaptive thresholded blur filter. The thresholds and limits used in the above filters can be changed based on a filter strength value coded in the bit-stream.

Segmentation

VP9 supports a segmentation framework that provides a flexible set of tools that can be used in an application specific way to improve perceptual quality for a given compression ratio. In particular, VP9 allows segmenting an image into up to 8 segments by indicating a segment affiliation for blocks 8x8 or larger, and then conveying various common signals or adjustments at the segment level in a compact manner. For each segment it is possible to specify: (1) a quantizer (Q) - absolute value or delta, (2) a loop filter strength (3) a prediction reference frame, and (4) a block skip mode that implies both the use of a (0, 0) motion vector and that no residual is coded. Each of these attributes for each segment may be selectively updated at the frame level. The exceptions to this are key frames, intra only frames or other frames where independence from past frame values is required for example in error resilient mode. In all such cases values are reset to defaults. Updates to this segment map are coded using either a temporal coding or a direct coding strategy (chosen at the frame level). If no explicit update is coded for a block's segment affiliation, then it persists to the next frame.

VP9 Bit-stream features

Apart from providing high compression efficiency using the core coding tools as described above, the VP9 bit-stream also includes various features designed to support specific use-cases that are important to Internet video delivery and consumption. This section provides an overview of these features.

Error Resilience

Transmission of conversational video with low latency over an unreliable network, necessitates a coding mode where decoding can still continue without errors even when arbitrary frames are lost. Specifically, the arithmetic encoder should still be able to decode symbols correctly in

frames subsequent to lost frames, even though frame buffers have been corrupted leading to encode-decode mismatch. The hope is that the drift between the encoder and decoder will still be manageable until such time as a key frame is sent or other corrective action, such as reference picture selection, can be taken. VP9 supports a frame level error_resilient_mode flag which when turned on, will only allow coding modes where this is possible to achieve. In particular, the following restrictions are imposed: (1) The entropy coding contexts are reset to defaults at the beginning of each frame. (This effectively prevents propagation of forward as well as backward updates). (2) For MV reference selection, the co-located MV from a previously encoded reference frame can no longer be included in the reference candidate list. (3) If segmentation is used, differential update of the segmentation map is disabled. These restrictions impose a modest performance drop in the order of 4-5%.

Frame Parallelism

A frame level frame_parallel_mode flag when turned on, enables an encoding mode where the entropy decoding for successive frames can be conducted in a quasi-parallel manner just by parsing the frame headers, before these frames actually need to be reconstructed. In this mode, only the frame headers need to be decoded sequentially. But beyond that, the entropy decoding for each frame can be conducted in lagged parallel mode. The reconstruction of the frames can then be conducted sequentially in coding order as they are required to be displayed. This mode will enable multi-threaded decoder implementations that enable smoother playback performance. Specifically, the mode imposes the restriction that backward entropy adaptations are disabled.

Tiling

In addition to making provisions for decoding multiple frames in parallel, VP9 also has support for decoding single frames using multiple threads. For this, VP9 introduces tiles, which are independently coded and decodable sub-units of the video frame. When enabled, a frame can be split into, for example, 2 or 4 column-based tiles. Each tile shares the same frame entropy model, but all contexts and pixel values (for intra prediction) that cross tile-boundaries take the same value as those at the left, top or right edge of the frame. Each tile can thus be decoded independently, which is expected to enable significant speed-ups in multi-threaded encoders/decoders, without introducing any additional latency. Note that loop-filtering across tile-edges can still be applied, assuming a decoder implementation model where the loop-filtering operation lags the decoder's reconstruction of the individual tiles within the frame so as not to use any pixel that is not already reconstructed. Further, backward entropy adaptation can still be conducted for the whole frame after entropy decoding for all tiles have finished.

Alternate Resolution Reference Frames

The VP9 bit-stream provides a number of flexible features that can be combined in application-specific ways to efficiently support various forms of scalability. Each frame in VP9 may use 3 other reference frames, but the number of available references from which those 3 are chosen is as many as 8. In addition, each coded frame may be re-sampled and coded at a resolution different from the references, allowing internal spatial resolution changes on the fly without a keyframe.

Scaling is done at prediction time for only those blocks referencing frames of a different resolution. Scaling and subpel prediction are done in a single step, using the same filters as are used in unscaled motion compensation. Since these are only defined in 16-th pel precision, the

smallest scaling ratio allowed is 1/16 (i.e. the current frame is 16x larger in dimension than the reference frame), while the largest allowed scaling ratio is 2/1, (i.e. the reference size is 2x larger than the current frame).

Furthermore, VP9 allows one of four different entropy coding contexts to be selected and optionally updated on every frame. These flexible features together enable an encoder/decoder to implement various forms of coarse-grained scalability - including temporal, spatial, or combined spatio-temporal scalability, without explicitly creating spatially scalable encoding modes.

YUV444/Alpha/Depth Channel

In addition to the common YUV 4:2:0 format, VP9 adds support for an optional additional plane, as well as additional chroma sub-sampling controls. Identified uses for this additional plane include alpha (transparency) channel and the z (depth) channel, though other applications are possible. Starting from the base luminance frame size, two additional bits – one for each dimension – are sent for each of the additional components (chrominance and optional alpha channel) to indicate whether the component is to be sub-sampled by a factor 2 in that dimension.

Coding Results

In this section, we compare the coding results in a PSNR sense obtained with VP9 (Sep 24, 2013 version) on two standard CIF and HD test sets, against those obtained by the X.264 [8] implementation of H.264/AVC (~April 2012 version) and the HEVC reference software (HM 11.0) in Main Profile. It is to be noted as a disclaimer however that it is extremely hard to compare different codecs in a fair and unbiased manner because of so many intrinsic differences in their implementations. For instance, the HEVC reference software only supports 1-pass encoding. In contrast, the libvpx VP9 encoder currently supports 2-pass encoding only; but a variety of speed focused optimizations have been incorporated to reduce encode time rather than make an exhaustive rd-optimal search over all possible encoding choices. Therefore at this stage, we can only compare codecs using a black-box approach with the available implementations. In course of time, we expect more varied implementations to be available for VP9 and HEVC that will make fairer comparisons easier.

All the comparisons in the tables below present a modified Bjøntegaard Delta Rate (BDRATE) [10] metric - which is a measure of the integral of the rate difference between two RD curves obtained from two codecs, one baseline and one test, expressed as a bit-rate percentage difference needed by the baseline codec over the test codec to achieve equivalent PSNR. VP9 is used as the baseline codec is all tables below. So, if the BDRATE number is x% for a test codec, it means that the VP9 codec requires (1 + x/100) times the bit-rate required by test codec to achieve equivalent PSNR. Thus a positive (negative) x indicates VP9 is less (more) efficient, than the test codec.

With this background, we test the three codecs (VP9 – libvpx, H.264/AVC – X.264, HEVC – HM 11.0) in two different encoding modes - *constant quality* and a *bitrate-controlled*. The specific command lines used are specifically mentioned in the sections below.

Constant Quality Tests

In this encoding mode, each coder is run with a single “target quality” parameter that controls the quality of encoding without enforcing any constraint on the bitrate. Besides, all encoders are

run with infinite keyframe intervals for consistency, where only the first frame needs to be a keyframe and thereafter keyframes are inserted only if needed.

Specifically, the RD curves for the libvpx encoder for VP9 (Sep 24, 2013), called `vpxenc`, is run with the following parameters:

```
vpxenc –end-usage=3 --cpu-used=0 --kf_max_dist=9999
--cq-level=<QP>
```

with varying QP.

The constant quality mode used for X.264 is the crf (constant rate factor) mode. The encoder `x264`, is run with the following parameters:

```
x264 --preset veryslow --tune psnr --keyint infinite --profile
high --crf <CRF_value>
```

with varying CRF_value. Note in X.264, the CRF_value is in the same range as the quantizer.

For HEVC (HM 11.0), the constant quality mode is invoked by varying the QP parameter and turning off rate control. The parameters used with the encoder `TAppEncoderStatic`, are:

```
TAppEncoderStatic -c encoder_randomaccess_main.cfg -c seq.cfg -ip
-1 --SearchRange=256 --QP=<QP>
```

with varying QP. This uses the default random access configuration of HM 11.0 as the baseline, and modifies it to use infinite keyframe interval, and also increases the search range to 256 to be consistent with that in VP9.

Table 1 presents BDRATE comparison in Average PSNR and SSIM terms comparing X.264 and HM 11.0 to the baseline VP9 Sep 24, 2013 head, over a set of 29 standard CIF (352x288) or SIF (320x240) resolution sequences, in the constant quality encoding mode. Table 2 presents the corresponding results over a set of 16 standard HD (1920x1080 or 1280x720) resolution sequences.

Table 1. Percentage BDRATE of H.264 and HEVC compared against VP9 for CIF/SIF sequences for constant quality encoding.

Sequence	**BDRATE (%) H.264 vs. VP9 [Av. PSNR]**	**BDRATE (%) H.264 vs. VP9 [SSIM]**	**BDRATE (%) HEVC vs. VP9 [Av. PSNR]**	**BDRATE (%) HEVC vs. VP9 [SSIM]**
Akiyo	-34.15	-52.22	-28.32	-50.94
Bowing	-39.55	-51.58	-18.77	-31.39
Bus	-6.63	-13.73	17.80	18.33
Cheer	-6.33	-9.51	4.92	2.96
City	-28.22	-23.10	2.96	4.75
Coastguard	-5.67	-15.07	18.00	22.74
Container	-25.50	-35.06	6.63	21.13
Deadline	-24.63	-39.34	-17.64	-27.85
Flower	-9.38	-28.45	21.12	14.10
Football	-14.05	-21.18	4.03	6.98
Foreman	-23.70	-29.74	5.95	-1.86
Hall_Monitor	-35.62	-52.34	-20.31	-27.47
Harbor	-4.15	-9.74	15.66	16.36
Highway	-24.68	-36.86	-0.12	-13.89
Husky	-1.44	-6.15	9.24	11.75
Ice	-9.59	-18.55	13.80	10.14

Mobile	-23.05	-36.70	11.01	1.15
Mother_Daughter	-22.14	-26.67	2.35	-5.63
News	-14.47	-27.25	-4.03	-18.32
Pamphlet	-36.12	-47.47	-21.38	-33.21
Paris	-18.05	-37.48	-10.55	-27.26
Sign_irene	-15.16	-23.95	3.08	-0.87
Silent	-21.84	-31.82	-23.50	-31.66
Soccer	-16.98	-24.07	7.22	7.01
Stefan	-8.31	-19.13	14.15	14.70
Students	-27.74	-39.31	-13.73	-26.27
Tempete	-12.09	-28.59	14.95	4.07
Tennis	-11.25	-23.10	-4.73	-14.26
Waterfall	-48.01	-53.40	-12.88	-19.87
OVERALL	**-19.60**	**-29.71**	**-0.11**	**-6.02**

Table 2. Percentage BDRATE of H.264 and HEVC compared against VP9 for HD sequences for constant quality encoding.

Sequence	**BDRATE (%) H.264 vs. VP9 [Av. PSNR]**	**BDRATE (%) H.264 vs. VP9 [SSIM]**	**BDRATE (%) HEVC vs. VP9 [Av. PSNR]**	**BDRATE (%) HEVC vs. VP9 [SSIM]**
Blue_Sky	-31.30	-40.72	14.27	17.27
City	-27.19	-45.69	-2.23	-17.89
Crew	-26.03	-29.27	5.28	3.49
Crowd_run	-8.31	-17.80	9.73	6.33
Cyclists	-36.44	-41.03	14.39	14.39
Jets	-49.13	-55.80	7.74	-1.00
Mobcal	-53.16	-62.10	-14.03	-27.12
Night	-19.39	-33.13	-2.56	-8.71
Old_Town_Cross	-47.68	-51.72	-14.20	-19.57
Parkjoy	-3.31	-15.63	13.53	11.76
Parkrun	-4.91	-19.84	9.24	5.98
Pedestrian	-40.54	-41.35	-4.77	-3.20
Riverbed	-23.72	-27.33	-2.60	-9.72
Sheriff	-19.02	-27.90	12.31	19.76
Shields	-38.76	-46.83	-2.15	-9.71
Sunflower	-57.16	-62.80	-4.09	-8.92
OVERALL	**-30.38**	**-38.68**	**2.49**	**-1.68**

Overall VP9 is seen to be quite competitive with the reference implementation of HEVC while vastly outperforming the X.264 implementation of H.264/AVC. Generally speaking in terms of metrics, VP9 typically performs better in SSIM than PSNR.

Bitrate-Control Tests

In this encoding mode, each coder is run with a “target bitrate” parameter, and a rate control mechanism employed during the encode process actually produces a bit-stream with bitrate close to the target specified. Besides, all encoders are run with infinite keyframe intervals for consistency as before.

Specifically, the RD curves for the libvpx encoder for VP9 (Sep 24, 2013), called `vpxenc`, is run with the following parameters:

```
vpxenc --end-usage=1 --cpu-used=0 --kf_max_dist=9999
--target-bitrate=<Bitrate>
```

with varying Bitrate values.

For X.264, a 2-pass encoding strategy is used. The `x264` encoder is run with the following parameters:

```
x264 --preset veryslow --tune psnr --keyint infinite --profile
high --pass 1 --bitrate <Bitrate>
```

```
x264 --preset veryslow --tune psnr --keyint infinite --profile
high --pass 2 --bitrate <Bitrate>
```

with varying Bitrate values. Note in X.264, the CRF_value is in the same range as the quantizer.

For HEVC (HM 11.0), the bitrate control mode is invoked by turning rate control on and providing a target bitrate parameter. The parameters used with the encoder `TAppEncoderStatic`, are:

```
TAppEncoderStatic -c encoder_randomaccess_main.cfg -c seq.cfg -ip
-1 --SearchRange=256 --RateControl=1 --TargetBitrate=<Bitrate>
```

with varying Bitrate values. This uses the default random access configuration of HM 11.0 as the baseline, and modifies it to use infinite keyframe interval, increases the search range to 256 to be consistent with that in VP9, and turns rate control on.

Table 3 presents BDRATE comparison in Average PSNR and SSIM terms comparing X.264 and HM 11.0 to the baseline VP9 Sep 24, 2013 head, over a set of 29 standard CIF (352x288) or SIF (320x240) resolution sequences, in the bitrate-controlled encoding mode. Table 4 presents the corresponding results over a set of 16 standard HD (1920x1080 or 1280x720) resolution sequences.

Table 3. Percentage BDRATE of H.264 and HEVC compared against VP9 for CIF/SIF sequences for bitrate-controlled encoding.

Sequence	**BDRATE (%) H.264 vs. VP9 [Av. PSNR]**	**BDRATE (%) H.264 vs. VP9 [SSIM]**	**BDRATE (%) HEVC vs. VP9 [Av. PSNR]**	**BDRATE (%) HEVC vs. VP9 [SSIM]**
Akiyo	-27.75	-42.20	-30.06	-49.46
Bowing	-47.52	-34.11	-31.26	-22.85
Bus	-7.01	-15.76	16.86	15.11
Cheer	-7.51	-10.70	4.96	2.73
City	-25.99	-25.40	3.78	-2.58
Coastguard	-6.97	-14.98	17.69	23.27
Container	-24.62	-34.26	1.54	13.84
Deadline	-24.71	-34.20	-13.33	-20.87
Flower	-12.62	-28.23	18.73	11.41
Football	-16.67	-24.79	5.10	3.36
Foreman	-21.06	-27.97	6.89	-2.53
Hall_Monitor	-34.23	-54.33	-17.87	-27.51
Harbor	-4.62	-12.33	15.67	13.19
Highway	-25.65	-37.07	-4.52	-14.68
Husky	-4.40	-11.62	13.75	13.12
Ice	-12.41	-16.50	11.02	5.38
Mobile	-24.54	-30.12	10.41	6.95
Mother_Daughter	-18.29	-19.50	-0.98	-4.44
News	-13.29	-26.39	-5.74	-17.25

Pamphlet	-33.86	-22.00	-23.27	-9.88
Paris	-19.08	-33.62	-9.76	-21.56
Sign_irene	-15.18	-23.11	4.03	0.94
Silent	-17.86	-22.17	-22.95	-26.35
Soccer	-17.96	-26.78	7.56	2.01
Stefan	-10.04	-17.84	15.13	7.03
Students	-26.50	-31.23	-10.33	-19.26
Tempete	-13.27	-26.73	15.02	6.87
Tennis	-10.75	-19.49	-5.72	-15.73
Waterfall	-42.68	-32.88	-10.21	-6.94
OVERALL	**-19.57**	**-26.08**	**-0.62**	**-4.71**

Table 4. Percentage BDRATE of H.264 and HEVC compared against VP9 for HD sequences for bitrate-controlled encoding.

Sequence	**BDRATE (%) H.264 vs. VP9 [Av. PSNR]**	**BDRATE (%) H.264 vs. VP9 [SSIM]**	**BDRATE (%) HEVC vs. VP9 [Av. PSNR]**	**BDRATE (%) HEVC vs. VP9 [SSIM]**
Blue_Sky	-42.86	-44.23	18.91	12.44
City	-32.86	-40.53	0.76	-4.42
Crew	-34.90	-35.84	9.37	11.23
Crowd_run	-11.66	-20.44	9.72	7.33
Cyclists	-44.60	-47.93	16.50	9.42
Jets	-46.80	-50.75	14.21	1.80
Mobcal	-53.46	-51.42	-0.22	3.01
Night	-18.48	-29.55	8.69	7.64
Old_Town_Cross	-37.13	-42.00	7.37	4.53
Parkjoy	-7.05	-15.08	17.76	14.24
Parkrun	-2.80	-19.65	14.66	7.81
Pedestrian	-41.51	-46.61	2.85	2.01
Riverbed	-25.86	-31.32	-0.72	-8.68
Sheriff	-25.55	-32.36	10.83	21.94
Shields	-41.36	-44.35	1.65	2.33
Sunflower	-52.37	-50.39	4.50	1.50
OVERALL	**-32.45**	**-37.65**	**8.55**	**5.88**

Overall VP9 is seen to be performing a little worse than the reference implementation of HEVC for HD sequences while comfortably outperforming the X.264 implementation of H.264/AVC.

Conclusion

In this overview paper, we have presented the main coding tools and bit-stream features in VP9, and also presented coding performance comparisons for mainstream video codecs H.264/AVC and HEVC against VP9. The results show VP9 to be close in performance to the latest mainstream codec HEVC from MPEG/ITU.

References

[1] Gary J. Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand, “Overview of the High Efficiency Video Coding (HEVC) Standard,” IEEE Trans. on Circuits and Systems for Video Tech., vol. 22, No. 12, Dec 2012.

[2] Thomas Wiegand, Gary J. Sullivan, Gisle Bjøntegaard; Ajay Luthra. "Overview of the H.264/AVC Video Coding Standard," IEEE Trans. on Circuits and Systems for Video Technology, vol. 13 No. 7, Jan 2011.

[3] J. Bankoski, J. Koleszar, L. Quillio, J. Salonen, P. Wilkins, Y. Xu, VP8 Data Format and Decoding Guide, RFC 6386, http://datatracker.ietf.org/doc/rfc6386/

[4] http://www.webmproject.org/

[5] http://xiph.org/vorbis/

[6] http://corecodec.com/products/matroska

[7] J. Han, A. Saxena, and K. Rose, “Towards jointly optimal spatial prediction and adaptive transform in video/image coding,” Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP), pp. 726–729, March 2010.

[8] http://www.videolan.org/developers/x264.html

[9] Frank Bossen, Davin Flynn, Karsten Suhring, HM Software Manual, https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/branches/HM-9.2-dev/doc/software-manual.pdf