# TECHNICAL OVERVIEW OF VP8, AN OPEN SOURCE VIDEO CODEC FOR THE WEB

*Jim Bankoski, Paul Wilkins, Yaowu Xu*
*Google Inc. 1600 Amphitheatre Parkway, Mountain View, CA, USA*
*{jimbankoski, paulwilkins, yaowu}@google.com*

## ABSTRACT

VP8 is an open source video compression format supported by a consortium of technology companies. This paper provides a technical overview of the format, with an emphasis on its unique features. The paper also discusses how these features benefit VP8 in achieving high compression efficiency and low decoding complexity at the same time.

***Index Terms*****—**VP8, WebM, Video Codec, Web Video

## 1. INTRODUCTION

In May 2010, Google announced the start of a new open media project “WebM”, which is dedicated to developing a high-quality, open media format for the web that is freely available to everyone. At the core of the project is a new open source video compression format, VP8. The VP8 format was originally developed by a small research team at On2 Technologies, Inc. as a successor of its VPx family of video codecs. Compared to other video coding formats, VP8 has many distinctive technical features that help it to achieve high compression efficiency and low computational complexity for decoding at the same time. Since the WebM announcement, not only has VP8 gained strong support from a long list of major industry players, but it has also started to attract broad interest in the video coding research community from both industry and academia.

This paper aims to provide a technical overview of the VP8 compression format, with an emphasis on VP8’s unique features. Section 2 briefly reviews VP8’s design assumptions and overall architecture; section 3 to section 7 describes VP8’s key technical features: transform and quantization scheme, reference frame types, prediction techniques, adaptive loop filtering, entropy coding and parallel processing friendly data partitioning; section 8 provides a short summary with experimental results and some thoughts on future work.

## 2. DESIGN ASSUMPTIONS AND FEATURE HIGHLIGHTS

From the very beginning of VP8’s development, the developers were focused on Internet/web-based video applications. This focus has led to a number of basic assumptions in VP8’s overall design:

**Low bandwidth requirement:** One of the basic design assumptions is that for the foreseeable future, available network bandwidth will be limited. With this assumption, VP8 was specifically designed to operate mainly in a quality range from “watchable video” (~30dB in the PSNR metric) to “visually lossless” (~45dB).

**Heterogeneous client hardware:** There is a broad spectrum of client hardware connected to the web, ranging from low power mobile and embedded devices to the most advanced desktop computers with many processor cores. It must, therefore, be possible to create efficient implementations for a wide range of client devices.

**Web video format:** VP8 was designed to handle the image format used by the vast majority of web videos: 420 color sampling, 8 bit per channel color depth, progressive scan (not interlaced), and image dimensions up to a maximum of 16383x16383 pixels.

The push for compression efficiency and decoder simplicity under these design assumptions led to a number of distinctive technical features in VP8 [1], relative to other known video compression formats, such as MPEG-2 [2], H.263 [3] and H.264/AVC [4]. The following list highlights the technical innovations in VP8:

**Hybrid transform with adaptive quantization**: VP8 uses 4x4 block-based discrete cosine transform (DCT) for all luma and chroma residual signal. Depending on the prediction mode, the DC coefficients from a 16x16 macroblock may then undergo a 4x4 Walsh-Hadamard transform.

**Flexible reference frames:** VP8 uses three reference frames for inter prediction, but the scheme is somewhat different from the multiple reference motion compensation scheme seen in other formats. VP8’s design limits the buffer size requirement to three reference frame buffers and still achieves effective de-correlation in motion compensation.

**Efficient intra prediction and inter prediction:** VP8 makes extensive uses of intra and inter prediction. VP8’s *intra* prediction features a new “TM_PRED” mode as one of the many simple and effective intra prediction methods. For *inter* prediction, VP8 features a flexible “SPLITMV” mode capable of coding arbitrary block patterns within a macroblock.

**High performance sub-pixel interpolation:** VP8’s motion compensation uses quarter-pixel accurate motion vectors for luma pixels and up to one-eighth pixel accurate motion vectors for chroma pixels. The sub-pixel interpolation of VP8 features a single stage interpolation process and a set of high performance six-tap interpolation filters.

**Adaptive in-loop deblocking filtering:** VP8 has a highly adaptive in-loop deblocking filter. The type and strength of the filtering can be adjusted for different prediction modes and reference frame types.

**Frame level adaptive entropy coding:** VP8 uses binary arithmetic coding extensively for almost all data values except a few header bits. Entropy contexts are adaptive at the frame level, striking a balance between compression efficiency and computational complexity.

**Parallel processing friendly data partitioning:** VP8 can pack entropy coded transform coefficients into multiple partitions, to facilitate parallel processing in decoders. This design improves decoder performance on multi-core processors, with close to zero impact to compression efficiency and no impact to decoding performance on single core processors.

## 3. HYBRID TRANSFORM WITH ADAPTIVE QUANTIZATION

Similarly to previous image and video coding schemes [2-4], VP8 uses transform coding to code residue signal after intra or inter predictions. A typical video image frame is divided into macroblocks, each macroblock consists of one 16x16 block of luma pixels (Y) and two 8x8 blocks of chroma pixels (U, V). VP8 further divides these luma and chroma blocks into 4x4 size blocks in the transform and quantization processes. A discrete cosine transform is applied on all 4x4 size luma and chroma blocks to convert the residue signal into transform coefficients. For macroblocks using 16x16 luma prediction modes, the DC coefficients from each of the 16 4x4 luma blocks within the macroblock are extracted and used to form a new 4x4 block, to which a 4x4 Walsh-Hadamard transform (WHT) is applied. Hence the 4x4 Walsh-Hadamard transform is used as a second order transform to further reduce the redundancy among the DC coefficients within the 16x16 luma area.

### 3.1. DCT

It is well known that the Karhunen Loeve Transform (KLT) approximates to achieve the optimal energy compaction, but its dependency on input content and its computational complexity make its application in video coding rather difficult [5]. On the other hand, the DCT is an orthogonal transform independent of input signal and proven to be only slightly worse in energy compaction than the KLT for natural video signals. In addition, the two dimensional (2-D) DCT is separable and has fast implementations. VP8 uses a 2-D DCT as the base for its transform coding and defines a 4x4 inverse 2-D DCT process as part of its bit stream format and decoding process. Essentially, the inverse 2-D DCT process defined in VP8 is a 4x4 variant of the LLM implementations as described in [6]. The signal flow graph of its one dimensional (1-D) portion is shown in Fig. 1.

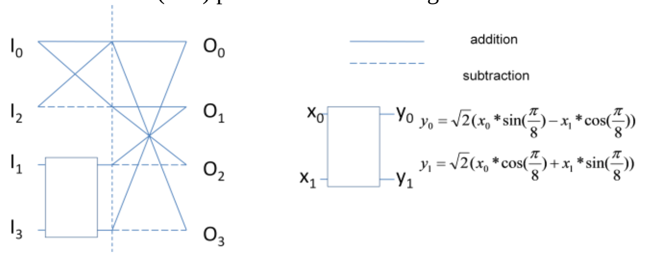

$I_0$
$I_2$
$I_1$
$I_3$
$O_0$
$O_1$
$O_2$
$O_3$
addition
subtraction
$x_0$
$x_1$
$y_0$
$y_1$
$y_0 = \sqrt{2}(x_0 * \sin(\frac{\pi}{8}) - x_1 * \cos(\frac{\pi}{8}))$
$y_1 = \sqrt{2}(x_0 * \cos(\frac{\pi}{8}) + x_1 * \sin(\frac{\pi}{8}))$

**Fig. 1.** VP8 inverse DCT's 1-D signal graph

On most modern processors with SIMD capability, the inverse DCT in VP8 can be implemented very efficiently. Provided that such implementations are possible, VP8's transform scheme provides slightly better energy compaction than the multiplication-less integer transform used in H.264/AVC [7], and is still computationally competitive.

### 3.2. WHT

The second order transform in VP8 is a 4x4 Walsh-Hadamard transform, which is designed to take advantage of the correlation among the first order DC coefficients of the 16 4x4 blocks within one macroblock. Similar to how the first order transform is defined, VP8 specifies the inverse transform in its decoding process. The inverse WHT used in VP8 can be described in matrix form as:

$$Y = HXH^{T}$$

Where X and Y are the 4x4 size input and output and H is defined as:

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

And $H^T$ is the transpose of $H$.

### 3.3. Adaptive Quantization

The quantizer of VP8 is designed to operate mainly in a quality range from ~30dB to ~45dB. Within this range, VP8 defines 128 quantization levels in its scalar quantization process. For each video frame, VP8 allows different quantization levels to be used for six frequency components: 1st order luma DC, 1st order luma AC, 2nd order luma DC, 2nd order luma AC, chroma DC and chroma AC. In addition, VP8's design includes a simple and effective region adaptive quantization scheme, in which the bitstream provides the capability of classifying macroblocks within a frame into 4 different segments, with each segment having its own quantization parameter set.

By carefully selecting appropriate scaling factors in transform and quantization design, VP8 can perform all the calculations using 16-bit operations in the full pipeline of transform, quantization, dequantization and inverse transform.

## 4. REFERENCE FRAMES

VP8 uses three types of reference frames for inter prediction: the "last frame", a "golden frame" (one frame worth of decompressed data from the arbitrarily distant past) and an "alternate reference frame." Overall, this design has a much smaller memory footprint on both encoder and decoder than designs with many more reference frames.

### 4.1 Golden Reference Frame

We have found experimentally that it is very rare for more than three reference frames to provide significant quality benefit, but the undesirable increase in memory footprint from the extra reference frames is substantial. And very often, the most beneficial reference frames are not the last three frames encoded. Depending on content, a frame from the distant past can be very beneficial in terms of inter prediction when objects re-appear after disappearing for a number of frames. Based on such observations, VP8 was designed to use one reference frame buffer to store a video frame from an arbitrary point in the past. This buffer is known as the "Golden Reference Frame." The format also defines a number of flags in the bitstream to notify a decoder when and how to update this buffer.

VP8 encoders can use the Golden Reference Frame in many ways to improve coding efficiency. It can be used to maintain a copy of the background when there are objects moving in the foreground, so that occluded regions can be easily and cheaply reconstructed when a foreground object moves away. Together with the last reference frame, the Golden Reference Frame may also be used to create a background sprite. Such an arrangement is helpful to compression efficiency in many video scenes. Another use of the golden frame is the coding of back and forth cut of two scenes, where the golden frame buffer can be used to maintain a copy of the second scene. Finally, the golden frame can also be used for error recovery in a real-time video conference, or even in a multi-party video conference for scalability [8].

## 4.2 Alternate (Constructed) Reference Frame

Unlike other types of reference frames used in video compression, which are always displayed to the user by the decoder, the VP8 alternate reference frame is decoded normally but may or may not be shown in the decoder. It can be used solely as a reference to improve inter prediction for other coded frames. Because alternate reference frames have the option of not being displayed, VP8 encoders can use them to transmit any data that is helpful to compression. For example, a VP8 encoder can construct one alternate reference frame from multiple source frames, or it can create an alternate reference frame using different macroblocks from many different video frames. The flexibility in VP8 specification allows many types of usage of the alternate reference frame for improving coding efficiency. Here are two illustrative examples:

**Noise-Reduced Prediction:** The alternate reference frame is transmitted and decoded similarly to other frames, hence its usage does not increase computational complexity in the decoder. However, in off-line applications the VP8 encoder is free to use more sophisticated processing to create them. One application of the alternate reference frame is for noise-reduced prediction. In this application, the VP8 encoder uses multiple input source frames to construct one reference frame through temporal or spatial noise filtering. This "noise-free" alternate reference frame is then used to improve prediction for encoding subsequent frames.

**Improving Prediction without B-Frames**: The lack of B frames has led to discussion in the research community about VP8's ability to achieve high compression efficiency. The VP8 format, however, supports intelligent use of the golden reference and the alternate reference frames together to compensate for this. The VP8 encoder can choose to transmit an alternate reference frame assembled with content from many "future" frames using sophisticated filtering. Encoding of subsequent frames can then make use of information from the past (last frame and golden frame) and from the future (alternate reference frame). Effectively, this helps the encoder to achieve compression efficiency without requiring frame reordering in the decoder.

## 5. PREDICTION TECHNIQUES

VP8 uses two classes of prediction modes: *Intra prediction* uses data within a single video frame, and *Inter prediction* uses data from previously encoded frames.

### 5.1 VP8 Intra Prediction Modes

VP8 intra prediction modes are used with three types of blocks:

- 4x4 luma
- 16x16 luma
- 8x8 chroma

Four common intra prediction modes are shared by these blocks:

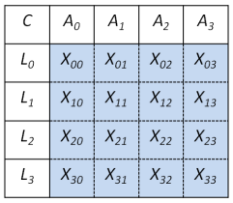
● **H_PRED (horizontal prediction)**: Fills each column of the block with a copy of the left column, *L*.

● **V_PRED (vertical prediction)**: Fills each row of the block with a copy of the above row, *A*.

● **DC_PRED (DC prediction)**: Fills the block with a single value using the average of the pixels in the row above *A* and the column to the left of *L*.

● **TM_PRED (TrueMotion prediction)**: In addition to the row *A* and column *L*, TM_PRED uses the pixel *C* above and to the left of the block. Horizontal differences between pixels in *A* and vertical differences between pixels in *L* are propagated (starting from *C*) to form the prediction block.

For 4x4 luma blocks, there are six additional intra modes corresponding to predicting pixels in different directions. As mentioned above, the TM_PRED mode is unique to VP8. Fig. 2 uses a 4x4 block as example to illustrate how the TM_PRED mode works:



| $C$ | $A_0$ | $A_1$ | $A_2$ | $A_3$ |
|---|---|---|---|---|
| $L_0$ | $X_{00}$ | $X_{01}$ | $X_{02}$ | $X_{03}$ |
| $L_1$ | $X_{10}$ | $X_{11}$ | $X_{12}$ | $X_{13}$ |
| $L_2$ | $X_{20}$ | $X_{21}$ | $X_{22}$ | $X_{23}$ |
| $L_3$ | $X_{30}$ | $X_{31}$ | $X_{32}$ | $X_{33}$ |

**Fig. 2.** Illustration of intra prediction mode TM_PRED

In Fig. 2, *C*, *A* and *L* represent reconstructed pixel values from previously coded blocks, and $X_{00}$ through $X_{33}$ represent predicted values for the current block. TM_PRED uses the following equation to calculate $X_{ij}$:

$$X_{ij} = L_i + A_j - C \ (i, j=0, 1, 2, 3)$$

The TM_PRED prediction mode for 8x8 and 16x16 blocks works in a similar fashion. Among all the intra prediction modes, TM_PRED is one of the more frequently used modes in VP8. For natural video sequences, it is typically used by 20% to 45% of all intra coded blocks. Together, these intra prediction modes help VP8 to achieve high compression efficiency, especially for key frames, which can only use intra modes.

### 5.2 VP8 Inter Prediction Modes

In VP8, inter prediction modes are used on inter frames (non-key frames). For any VP8 inter frame, there are typically three previously coded reference frames that can be used for prediction. A typical inter prediction block is constructed using a motion vector to copy a block from one of the three frames. The motion vector points to the location of a pixel block to be copied. In video compression schemes, a good portion of the bits is spent on encoding motion vectors; the portion can be especially large for video encoded at lower data rates. VP8 provides efficient motion vector coding by reusing vectors from neighboring macroblocks. For example, the prediction modes "NEAREST" and "NEAR" make use of last and second-to-last, non-zero motion vectors from neighboring macroblocks. These inter prediction modes can be used in combination with any of the three different reference frames.

In addition, VP8 has a sophisticated, flexible inter prediction mode called SPLITMV. This mode was designed to enable flexible partitioning of a macroblock into sub-blocks to achieve better inter prediction. SPLITMV is useful when objects within a macroblock have different motion characteristics. Within a macroblock coded using the SPLITMV mode, each sub-block can have its own motion vector. Similar to the strategy of reusing without transmitting motion vectors at the macroblock level, a sub-block can also use motion vectors from neighboring sub-blocks above or left of the current block without transmitting the motion vectors. This strategy is very flexible and can encode any shape of sub-macroblock partitioning. Fig. 3 (a) shows an example of a macroblock with 16x16 luma pixels that is partitioned to 16 4x4 blocks:

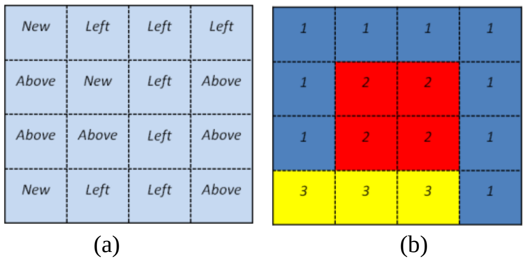


**Fig. 3.** Illustration of VP8 inter prediction mode SPLITMV

In Fig. 3 (a), *New* represents a 4x4 bock coded with a new motion vector, and *Left* and *Above* represent a 4x4 block coded using the motion vector from the left and above, respectively. This example effectively partitions the 16x16 macroblock into three different segments with three different motion vectors (represented by *1*, *2* and *3*), as seen in Fig. 3 (b).

### 5.3 Sub-pixel Interpolation

VP8's motion compensation uses quarter pixel accurate motion vectors for luma pixels. The sub-pixel interpolation of VP8 features a single-stage interpolation process and a set of high performance six-tap interpolation filters. The filter taps used for the six tap filters are:

[3, -16, 77, 77, -16, 3]/128 for ½ pixel positions
[2, -11, 108, 36, -8, 1]/128 for ¼ pixel positions
[1, -8, 36, 108, -11, 2]/128 for ¾ pixel positions

Chroma motion vectors in VP8 are calculated from their luma counterparts by averaging motion vectors within a macroblock, and have up to one eighth pixel accuracy. VP8 uses four-tap bicubic filters for the 1/8, 3/8, 5/8 and 7/8 pixel positions. Overall, the VP8 interpolation filtering process achieves optimal frequency response with high computation efficiency.

## 6. ADAPTIVE LOOP FILTERING

Loop filtering is a process of removing blocking artifacts introduced by quantization of the DCT coefficients from block transforms. VP8 brings several loop-filtering innovations that speed up decoding by not applying any loop filter at all in some situations. VP8 also supports a method of implicit segmentation where different loop filter strengths can be applied for different parts of the image, according to the prediction modes or reference frames used to encode each macroblock. For example it would be possible to apply stronger filtering to intra-coded blocks and at the same time specify that inter coded blocks that use the Golden Frame as a reference and are coded using a (0,0) motion vector should use a weaker filter. The choice of loop filter strengths in a variety of situations is fully adjustable on a frame-by-frame basis, so the encoder can adapt the filtering strategy in order to get the best possible results. In addition, similar to the region-based adaptive quantization in section 3, VP8 supports the adjustment of loop filter strength for each segment. Fig. 4 shows an example where the encoder can adapt the filtering strategy based on content.

## 7. ENTROPY CODING AND DATA PARTITIONING

Except for very few header bits that are coded directly as raw values, the majority of compressed VP8 data values are coded using a boolean arithmetic coder. The boolean arithmetic coder encodes one boolean value (0/1) at a time. It is used to losslessly compress a sequence of bools for which the probability of their being 0 or 1 can be well-estimated.

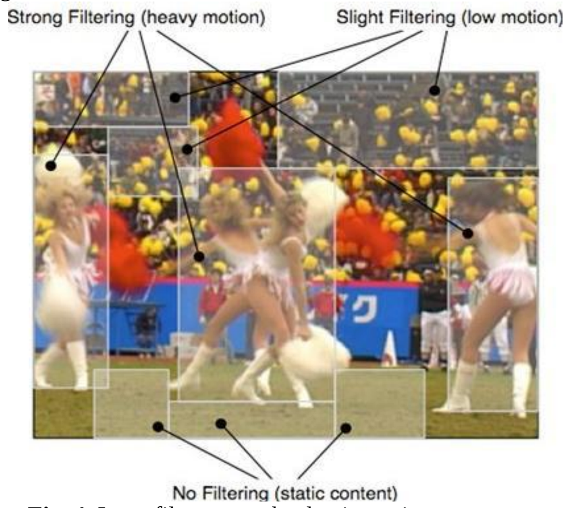

Strong Filtering (heavy motion)
Slight Filtering (low motion)
No Filtering (static content)

**Fig. 4.** Loop filter strength adaptive to image content

Most symbol values in VP8 are binarized into a series of boolean values using a pseudo Huffman Tree scheme. In such a scheme, a binary tree is first created for a set of symbols similarly to how a Huffman coding tree is created, and any symbol in the set can be represented by a series of binary values generated by traversing the binary tree from the root node to the corresponding leaf node. Each non-leaf node in the binary tree has a probability assigned based on the likelihood of taking the left (0) branch for traversing. Through such a binarization scheme and storing data in pseudo Huffman tree structures, the encoding/decoding style in VP8 is very consistent for all the values in the bitstream, such as macroblock coding modes, reference frame types, motion vectors, quantized coefficients, and so on. Such a scheme improves module reusability in both hardware and software implementations of VP8 entropy encoder or decoder.

### 7.1 Frame Adaptive Entropy Context

VP8 uses conditional probability distribution to model the context used for entropy coding of information such as macroblock coding modes, motion vectors, and quantized transform coefficients, and so on. For the purpose of decoding simplicity, VP8 keeps the probability distributions for entropy coding stable within one frame, and supports distribution updates on a per-frame basis. On a key frame, all probability distributions are reset to the default baseline, and then on each subsequent frame, these probabilities are combined with individual updates for use in coefficient coding within the frame. The bitstream also has mechanisms to signal how the updated probabilities affect the baseline distributions for decoding the subsequent frames. On the one hand, the combined probability distributions may become the new baseline for decoding of subsequent frame; on the other hand, the probability updates can also be discarded after decoding of the current frame and the baseline probability distributions prior to the updates are then used for decoding of subsequent frames. Compared to context-based binary arithmetic coding in H.264/AVC [9], the design of frame-level adaptive entropy context achieves lower complexity in decoder implementations, and also allows better error recovery in decoders.

## 7.2 Parallel Processing-Friendly Coefficient Partitioning

One of the recent trends in micro-processor development is processors having multiple "cores." To make effective use of the computational power of multi-core processors, VP8 has data partitioning features that support parallel processing. In essence, the VP8 bitstream first separates the compressed data into two categories, one for macroblock coding modes and motion vectors and one for quantized transform coefficients. In addition, VP8 allows transform coefficients to be packed into more than one partition without changing the inter macroblock dependency in coding. For example, in the FOUR_TOKEN_PARTITION mode, transform coefficients from macroblock row 0, 4, 8, …, are packed into the first coefficient partition; coefficients from macroblock row 1, 5, 9, …, are packed into the second coefficient partition, and so on. Though the transform coefficients are packed into different partitions, the entropy contexts used to encoding them are the same as the case where all transform coefficients are packed into a single partition with all macroblocks coded in raster scan order.

Hence, VP8 allows decoders to make efficient use of multiple cores to decode several macroblock rows at the same time. The separation of mode/motion vector data from the transform coefficients allow reference data prefetching in many hardware platforms and the coefficient data partitioning in VP8 supports the parallel decoding of more than one macroblock rows. VP8 supports up to eight token partitions in any given frame, which enables a decoder to efficiently use as many as eight cores. The number eight is not an arbitrary number, but based on research into technology trends and predictions by leading micro-processor vendors, the vast majority of processors in the coming decade will have no more than eight cores.

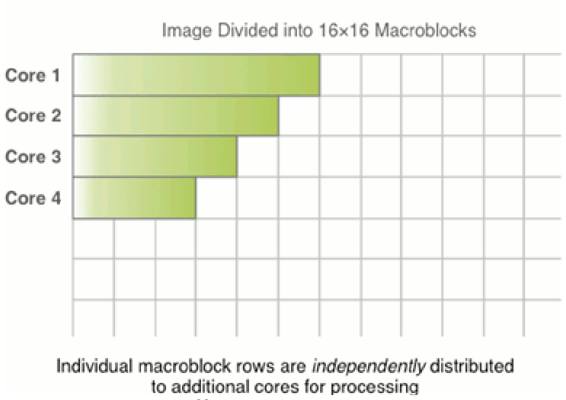


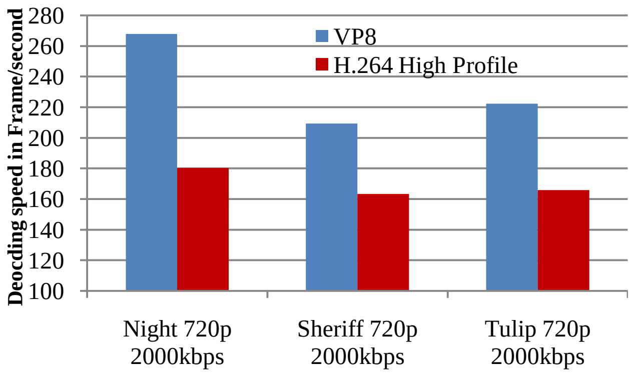
Image Divided into 16×16 Macroblocks
Core 1
Core 2
Core 3
Core 4
Individual macroblock rows are *independently* distributed to additional cores for processing

**Fig. 5.** VP8 coefficients partitioning and parallel decoding

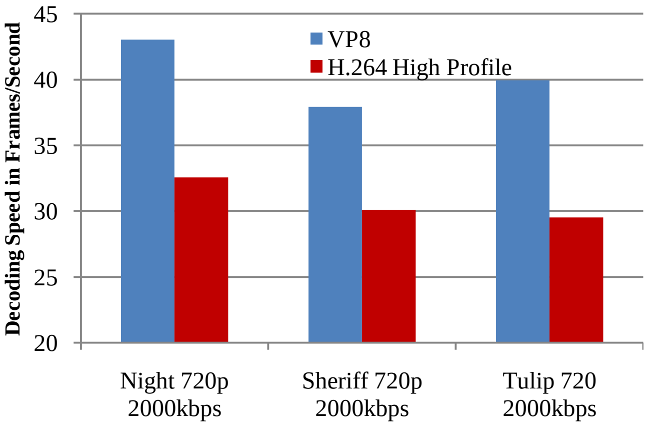
## 8. CONCLUSIONS AND FUTURE WORK

As a result of the many advanced coding features, VP8 can make the best use of computation power in modern hardware for improving compression efficiency while maintaining fast decoding speed on majority devices connected to the web.

Figure 6 shows the decoding speed test results on two different hardware platforms for video files encoded in VP8 and H.264 (high profile) at similar bitrates. The WebM Project libvpx reference encoder and x264 software encoder were used to produce the VP8 and H.264 files, respectively. For decoding, the latest FFmpeg decoder software (version SVN-r264000) with libavcodec (version 52.108.0) was used for both VP8 and H.264 in the test. By using the same software decoder, one can expect the level of optimization for performance to be similar for both VP8 and H.264; therefore the decoding speeds may reflect the intrinsic decoding complexity. As shown in Fig. 6, the decoding speeds of VP8 encoded files are consistently faster, average around 30%, than those of H.264 encoded files at a similar bitrate across the two different hardware platforms.



(a) Intel Core i7 3.2GHz



(b) Intel Atom N270 1.66GHz

**Fig. 6.** Decoding speed test results

Fig. 7 shows the results of encoding quality comparison between VP8 and H.264 (high profile) on a number of standard video test clips. For VP8, the encoder software used was the libvpx software from the WebM Project. For H.264 high profile, the encoder software used was the latest x264 encoder software, which is arguably the best encoder implementation for H.264/AVC. The H.264 encoding setting used was recommended by x264 developers and tuned to produce the best possible PSNR results. The x264 encoder used options:

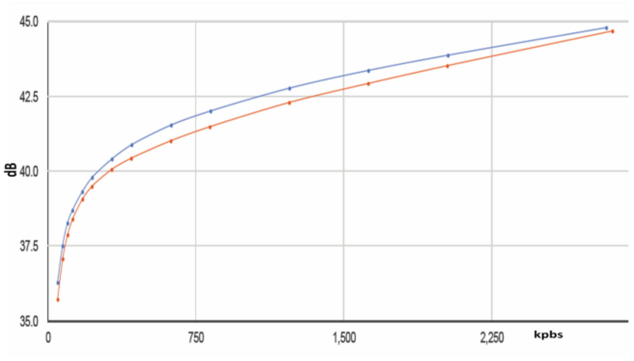
```
--preset=veryslow --tune psnr
```

The WebM VP8 encoder used options:

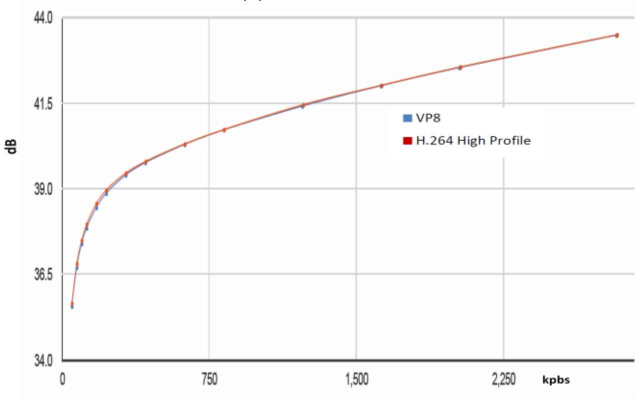
```
-p 2 --best --auto-alt-ref=1 --minsection-pct=0
--maxsection-pct=800 --lag-in-frames=25 --kf-min-dist=0
--kf-max-dist=99999 --static-thresh=0 --min-q=0
--max-q=63 --drop-frame=0 --bias-pct=50 --psnr
--arnr-maxframes=7 --arnr-strength=6 --arnr-type=3
```

Please refer to the respective project websites for detailed description of the options used above. As shown in Fig. 7, the reference encoder implementation of VP8 achieves very competitive quality results against the H.264/AVC encoder.

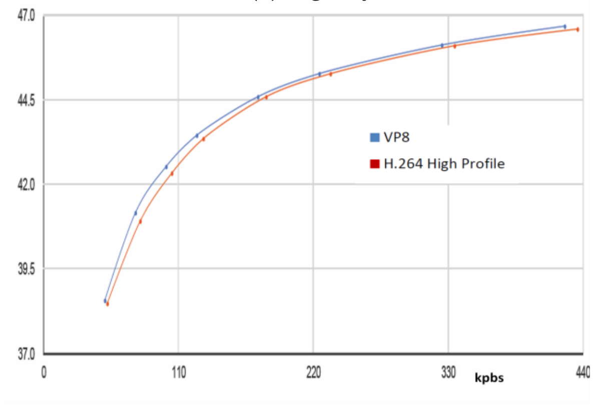
It is not difficult to conclude from the test results that, in the designed operating range of web video, VP8 can achieve compression efficiency that is competitive to the best H.264/AVC



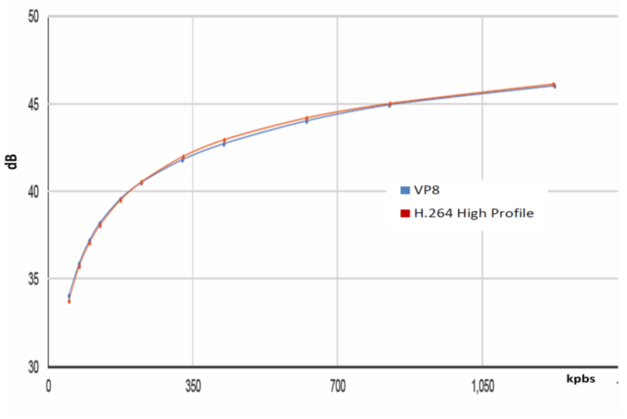
(a) Hall Monitor



(b) Highway



(c) Pamphlet



(d) Deadline

**Fig. 7.** Encoding quality test results

encoder available. At the same time, however, the low complexity design of the VP8 format enables decoder implementations to achieve much faster decoding speeds than H.264/AVC on various platforms.

The VP8 reference encoder implementation from the WebM Project is not yet making full use of all the VP8 features described in this paper. In addition, many techniques used in other modern video encoders, such as advanced rate control strategies, rate distortion optimization methods, motion estimation methods and so on, are directly applicable to VP8. As a result, there is great potential for innovations in future versions of VP8 encoder and decoder.

## ACKNOWLEDGEMENTS

The authors would like to thank Prof. Bastiaan Kleijn at the School of Electrical Engineering at KTH (the Royal Institute of Technology) in Stockholm, Sweden for reviewing early drafts, the paper has benefited a great deal from his valuable feedback. This paper has also benefited from helpful comments by our colleagues at Google, Dr. Pascal Massimino and Mr. John Luther, as well as insightful comments from four anonymous reviewers.

## REFERENCE

[1] J. Bankoski, P. Wilkins, Y. Xu, “VP8 Data Format and Decoding Guide,” http://www.ietf.org/internet-drafts/draft-bankoski-vp8-bitstream-01.txt, Jan 2011.

[2] “Generic Coding of Moving Pictures and Associated Audio Information- Part 2: Video,” ITU-T and ISO/IEC JTC 1, ITU-T Recommendation H.262 and ISO/IEC 13 818-2 (MPEG-2), 1994.

[3] “Video Coding for Low Bit Rate Communication,” ITU-T, ITU-T Recommendation H.263 version 1, 1995.

[4] “Advanced Video Coding for Generic Audiovisual Services,” ITU-T, ITU-T Recommendation H.264, November 2007.

[5] J. Saghri, A. Tescher, and J. Reagan, “Practical transform coding of multispectral imagery,” Signal Processing Magazine, IEEE, pp. 32–43, 2005.

[6] C. Loeffler, A. Ligtenberg, and G. S. Moschytz, “Practical fast 1-D DCT algorithms with 11 multiplications,” Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '89), vol. 2, pp. 988–991, Glasgow, UK, May 1989.

[7] H. S. Malvar, A. Hallapuro, M. Karczewicz, L. Kerofsky, “Low-complexity Transform and Quantization in H.264/AVC,” IEEE Transactions on Circuit and Systems for Video Technology, Vol. 13, No. 7, pp. 598-603, July 2003.

[8] J. Bankoski, “On2’s Truemotion VP7 video codec and golden frames”, EE Times, Jul 2008.

[9] D. Marpe, T. Wiegand, “Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard,” IEEE Transactions on Circuit and Systems for Video Technology, Vol. 13, No. 7, pp. 620-635, July 2003