A Hybrid Video Codec Based on Extended Block Sizes, Recursive Integer Transforms, Improved Interpolation, and Flexible Motion Representation

Marta Karczewicz, Peisong Chen, Rajan Joshi, Xianglin Wang, Wei-Jung Chien, Rahul Panchal, Muhammed Coban, In Suk Chong, and Yuriy A. Reznik
Qualcomm Inc., 5775 Morehouse Drive, San Diego, CA, USA 92121;

ABSTRACT

This paper describes video coding technology proposal submitted by Qualcomm Inc. in response to a joint call for proposal (CfP) issued by ITU-T SG16 Q.6 (VCEG) and ISO/IEC JTC1/SC29/WG11 (MPEG) in January 2010. Proposed video codec follows a hybrid coding approach based on temporal prediction, followed by transform, quantization, and entropy coding of the residual. Some of its key features are extended block sizes (up to 64×64), recursive integer transforms, single pass switched interpolation filters with offsets (single pass SIFO), mode dependent directional transform (MDDT) for intra-coding, luma and chroma high precision filtering, geometry motion partitioning, adaptive motion vector resolution. It also incorporates internal bit-depth increase (IBDI), and modified quadtree based adaptive loop filtering (QALF). Simulation results are presented for a variety of bit rates, resolutions and coding configurations to demonstrate the high compression efficiency achieved by the proposed video codec at moderate level of encoding and decoding complexity. For random access hierarchical B configuration (HierB), the proposed video codec achieves an average BD-rate reduction of 30.88% compared to the H.264/AVC alpha anchor. For low delay hierarchical P (HierP) configuration, the proposed video codec achieves an average BD-rate reduction of 32.96% and 48.57%, compared to the H.264/AVC beta and gamma anchors, respectively.

Keywords: video coding, extended block sizes, transforms, factorizations, geometry motion partition, MDDT, switched interpolation filters with offsets

1. INTRODUCTION

With the increasing demand for video content over wired/wireless networks, there is an increasing need for video coding techniques that can provide significantly higher coding efficiency compared to the existing video coding standard H.264/AVC[1,2]. To address this need, a call for proposal (CfP) for video compression technology[3,4] was issued jointly by ISO/IEC JTC 1/SC 29/WG 11 (MPEG) and ITU-T SG 16 Question 6. In response to the CfP, Qualcomm Inc. submitted a video coding technology proposal[5]. This paper describes some of the key features of Qualcomm's proposal:

- Block sizes larger than 16×16
- Integer transforms of sizes 16×16, 16×8, and 8×16 in addition to 4×4 and 8×8
- Mode dependent directional transform (MDDT) for intra-coding
- Luma high precision filtering
- Single pass switch interpolation filters with offsets (single pass SIFO)
- Geometry motion partitioning
- Adaptive motion vector resolution

The proposed video codec utilized some coding tools proposed by other companies and adopted into the JM Key Technology Areas (JMKTA) software[6], including internal bit-depth increase (IBDI)[7], and modified quadtree based adaptive loop filtering (QALF)[8]. Several other tools such as high precision chroma filtering, direct mode for P slices, motion vector scaling, and changes to the H.264/AVC mode syntax for B slices were also included in the proposed video codec. For detailed description of these additional techniques the reader is referred to[5].

The rest of the paper is organized as follows. In Section 2, we explain details of key features of the proposed codec, presenting them in the context of relevant functional blocks in the video codec. In Section 3, we present experimental results. The performance of the proposed video codec is evaluated under low delay and random access conditions. The coding results are compared to the H.264/AVC anchors provided by CfP[3,4]. Conclusions are presented in Section 4.

2. CODEC DESIGN

2.1 Intra-block coding

Intra-prediction used in the proposed video codec is identical to that of H.264/AVC[1,2]. For 4×4 and 8×8 block sizes, 9 prediction modes are used and for 16×16 block size, 4 prediction modes are used. However, after intra-prediction using H.264/AVC modes, there is usually still significant directional information left in the intra-prediction residual. For this reason, the proposed codec uses mode dependent directional transform (MDDT) and adaptive coefficient scanning to maximally compact the intra-prediction residual energy and increase entropy coding efficiency, as described in the following subsections.

1) Mode dependent directional transform for intra prediction residuals

To exploit the directionality of the intra-prediction residuals, the proposed video codec uses mode dependent directional transform (MDDT). Since the transform is dependent on the mode, no side information is necessary, which is rather important for smaller block sizes such as 4x4 and 8x8. MDDT was first proposed in [9], [10]. We briefly describe the design and the implementation of the MDDT.

MDDT is based on the Karhunen-Loève transform (KLT). Ideally KLT derived from the statistics of the intra-prediction residuals for a particular intra prediction mode, would be the optimal choice from a rate-distortion perspective for that mode. However, KLT is a non-separable transform. For an N×N block, the KLT matrix size is N^2×N^2. Thus, KLT is prohibitively expensive in terms of storage and computational requirement. Our proposed video codec uses a separable N×N directional transform, which can be described as

$$Y = C_i X R_i, \tag{1}$$

where C_i and R_i are the column and row transform matrices for intra prediction mode i, respectively. Singular Value Decomposition (SVD) is applied to the training set of intra-prediction residuals for a particular intra-prediction mode i, first in the row direction, and then in the column direction to determine the transform matrices C_i and R_i. The proposed codec uses fixed-point approximations of the transform matrices.

2) Adaptive coefficient scanning

After applying a transform to the intra-prediction residuals, the 2-D transform coefficient matrix is converted into a 1-D array. In H.264/AVC, zigzag scanning order is used so that lower frequency coefficients are positioned earlier in the scan. However, in case of MDDT, even after separable directional transform is applied, the resulting 2-D transform coefficient matrix has some directionality. For example, consider the vertical prediction mode (mode 0). After intra-prediction, transform and quantization, the nonzero coefficients tend to exist along the horizontal direction. By using a coefficient scanning process oriented in the horizontal direction instead of the zigzag scan the non-zero coefficients in the 2-D matrix can be positioned towards the beginning of the 1-D array. This in turn improves entropy coding efficiency. Quantized transform coefficients of different intra-prediction modes carry different statistics. Therefore, for each mode, adaptive coefficient scanning is used. This is accomplished as follows:

1) At the beginning of each video slice, coefficient scanning order for each intra-prediction mode is initialized;
2) For each non-zero coefficient coded, the count at the corresponding position is incremented by one;
3) After each macroblock is coded, the scanning order is updated according to the count statistics collected;
4) The collected count statistics are normalized if necessary;
5) The updated scanning order is used for the coding of future blocks. The control returns to step 2 until the encoding of the slice is completed.

2.2 Inter-block coding

The proposed video codec introduces a number of coding tools to improve the inter-block coding efficiency. Extended block size motion partitions and geometry motion partitions are introduced to better align the motion partition to the video content. Also, use of higher precision motion vector representation and improved sub-pixel interpolation, further improve the efficiency of motion compensation.

1) *Mode dependent directional transform for intra prediction residuals*

For higher resolution sequences such as 720p and 1080p, it is much more likely that spatial areas larger than 16×16 have homogeneous motion. Thus, it is advantageous to allow for motion partition sizes larger than 16×16. Such an extended block size motion partitioning scheme was proposed in [11] and has been adopted by JMKTA. The proposed video codec uses this scheme where the largest motion partition size is set to 64×64. At 64×64 block size, motion partitions of 64×64, 64×32, 32×64, and 32×32 are permitted. If the motion partition of 32×32 is chosen, each 32×32 block can have motion partitions of 32×32, 32×16, 16×32, and 16×16. If a 16×16 partition is chosen at the 32×32 block level, each 16×16 block can be further partitioned in accordance with the existing motion partition sizes in AVC/H.264 (16×16, 16×8, 8×16, 8×8, 8×4, 4×8, and 4×4). In addition, for 64×64 and 32×32 blocks, skip and direct modes are also used as in the case of 16×16 macroblocks in H.264/AVC. This tool will be referred to as *BigBlocks* throughout this paper.

The motion partition is determined by performing a bottom-up search. First the minimum rate-distortion (RD) cost for each 16×16 macroblock is determined. Then, the combined RD cost for 4 16×16 blocks is compared with the RD costs for 32×32, 32×16, and 16×32 partitions. By choosing the minimum RD cost, we obtain the optimal partition for the 32×32 block. Then, this process is repeated for the 4 neighboring 32×32 blocks, to obtain the optimal motion partition for the 64×64 block. It should be noted that if the best motion partition contains 16×16 blocks, then 16×16 blocks may be intra-coded.

2) Geometry Motion Partition

In H.264/AVC, a translational motion model is assumed for rectangular blocks. But this model is not accurate when a motion boundary is present within a rectangular block. This problem is exacerbated when extended block size motion partitions are used. One way to overcome this problem is to divide a block containing a motion boundary into smaller rectangular blocks so that the motion boundary affects only a few of the smaller blocks. But in this case, the number of motion vectors that are needed to be sent to the decoder is much larger, resulting in higher rate. Another solution that was proposed in [12], [13] is to use another kind of motion partition known as a geometry motion partition. This motion partition divides the block into 2 regions. The boundary separating the 2 regions is defined by a straight line. One motion vector is sent for each region. In our proposed codec, geometric motion partition is introduced at block sizes of 64×64, 32×32 and 16×16.

Now we describe how the various geometry partitions are created. The origin is assumed to be at the center of the block. Then, each geometry partition is defined by a line passing through the origin that is perpendicular to the line defining the partition boundary. This is shown in Figure 1.a. The geometry partition is defined by the angle subtended by the perpendicular line with the X axis (θ) and the distance of the partition line from the origin (ρ). The equation of the line defining the partition boundary can be specified as

$$y = \frac{-1}{\tan\theta}x + \frac{\rho}{\sin\theta} = \mathrm{m}x + \mathrm{c}. \tag{2}$$

We use two 32 bit lookup tables, one to store the slope $-1/\tan\theta$, and the other to store the scaled Y-intercept, $1/\sin\theta$. The region to which each pixel belongs is calculated on the fly.

At each block size, 32 different values of θ are permitted (from 0 to 360^o in steps of 11.25^o). The number of values ρ can take depends on the block size. For block size of 16×16, ρ can take 8 possible values (0,…,7). For block sizes of 32×32 and 64×64, ρ can take 16 and 32 possible values, respectively. Thus for block sizes of 16×16, 32×32, and 64×64, there are 256, 512, and 1024 possible geometry partitions, respectively.

a) Motion search for geometry partition

Since there are so many possible geometry partitions for each block size, it is prohibitively expensive for the encoder to do motion estimation for each region of each geometry partition and then, perform rate-distortion optimization. To overcome this difficulty, whenever possible, motion vectors from the rectangular partitions at all block sizes are reused

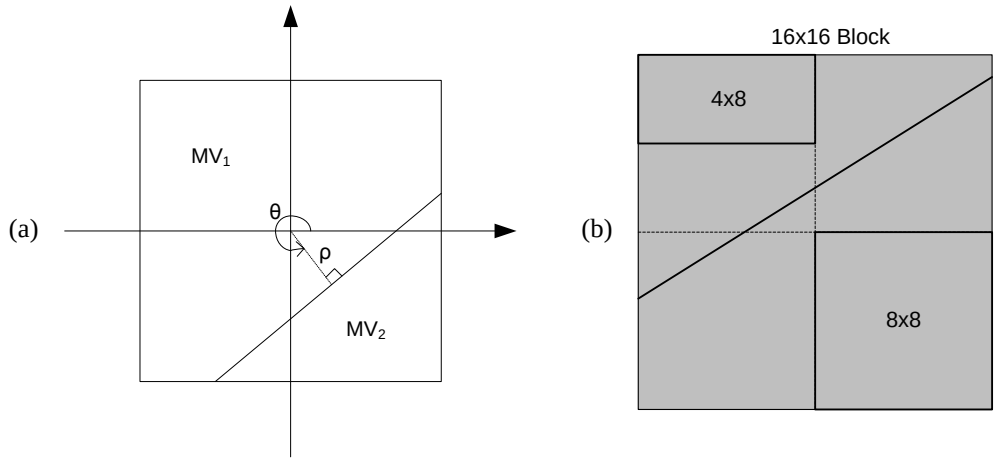

Figure 1. (a) Parameters defining geometry of motion partition. (b) Reusing rectangular partition motion vectors for geometry partition motion search.

to speed-up the motion vector search for geometry partitions. The encoder is structured in such a manner that the motion estimation for all the rectangular motion partitions is performed before the motion search for the geometry partitions. For each geometry partition region, we find the largest rectangular block that lies entirely inside the region and for which a motion vector is available. The estimated motion vector for that block is used as the motion vector for the partition region. If there are multiple blocks of the same size that lie entirely inside the region, the first block in the scan order is chosen. Figure 1.b shows an example of this process. If a geometry partition at block size of 16×16 is being considered, all blocks of sizes 16×16, 16×8, 8×16, 8×8, 8×4, 4×8, and 4×4 are tested to see whether they lie entirely inside the partition region.

To further reduce the amount of computation, a hierarchical search strategy is used.. After choosing a motion vector for each region of the geometry partition and performing overlapped motion compensation as described below, the motion cost is evaluated using sum of absolute differences (SAD). For a 16×16 block, a motion cost is calculated for each possible geometry partition. Then, 16 geometry partitions with best motion costs are selected. Full enhanced predictive zonal search (EPZS) is performed on each partition region of each of the 16 partitions. This is followed by calculation of the true rate-distortion (RD) cost. The geometry partition with the lowest RD cost is chosen. This is compared against the RD cost for optimal rectangular partitioning of the 16×16 block to determine whether geometry partitioning should be used for that particular block.

b) Overlapped motion compensation for geometry partition

Since two different motion vectors are used for motion compensation inside a block with geometry partition, the pixels at the partition boundary may have large discontinuities that can produce visual artifacts similar to blockiness. Furthermore, since the geometry partition boundary may not be aligned with macroblock and sub-macroblock boundaries, it is likely that the deblocking filter may not be able to reduce the blockiness resulting from geometry motion partitions. To alleviate this, we use the concept of overlapped block motion compensation (OBMC) for geometry motion partitions. Let the two regions created by a geometry partition be denoted by region 1 and region 2. Let the corresponding motion vectors be denoted by MV_1 and MV_2, respectively. A pixel from region 1 (2) is defined to be a boundary pixel if any of its four connected neighbors (left, top, right, and bottom) belongs to region 2 (1).

Figure 2.a shows an example where light mesh squares belong to the boundary of region 1 and dark mesh squares belong to the boundary of region 2. If a pixel is not a boundary pixel, normal motion compensation is performed using the appropriate motion vector. But if a pixel is a boundary pixel, the motion compensation is performed using a weighted sum of the motion predictions from the two motion vectors, MV_1 and MV_2. The weights are 2⁄3 for the region containing the boundary pixel and 1⁄3 for the other region. The overlapped boundaries improve the visual quality of the reconstructed video while also providing small coding gain.

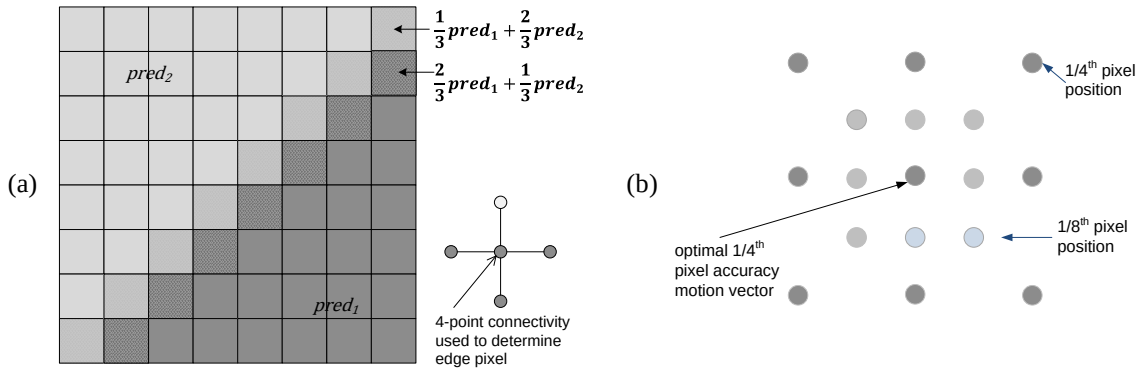

Figure 2. (a) Overlapped motion compensation for geometry partitions. (b) Motion search for 1/8th pixel accuracy.

3) Motion Accuracy

The H.264/AVC standard[1,2] allows motion vectors to be specified with 1/4th pixel accuracy. But the 1/4th pixel positions are interpolated using bilinear interpolation from full pixel and half pixel positions. Using separate filters designed to perform 1/4th and 3/4th pixel interpolation results in more accurate interpolation. Furthermore, in certain sequences and certain regions, it is beneficial to have higher (1/8th pixel) accuracy motion vectors. For the proposed video codec, for each region in a motion partition, the motion accuracy can be adaptively chosen to be 1/4th pixel or 1/8th pixel. We will refer to this as adaptive motion vector resolution. The choice of motion vector resolution is signaled to the decoder. The details of how to encode the motion vector resolution flag as well as motion vector differences are provided in Section 2.5.4).

The motion search at the encoder is modified as follows. For every block in a motion partition, first a 1/4th pixel accuracy motion vector is found using EPZS (or any other preferred motion search algorithm). Then, as shown in Figure 2.b, eight surrounding 1/8th pixel positions are searched to find the best 1/8th pixel accuracy motion vector. The motion vector (1/4th or 1/8th pixel accuracy) with the lowest RD cost is selected. Thus, the added complexity for adaptive motion vector resolution is mainly due to the interpolation and the RD cost calculations corresponding to the eight 1/8th pixel positions. The details of interpolation will be discussed in the next subsection.

4) Interpolation

In the proposed video codec, single pass switched interpolation filters with offsets (single pass SIFO) are used to interpolate the reference frame to 1/4th pixel accuracy for luma component. The single pass SIFO filters were first proposed in[14]. First we review the interpolation methods used in the H.264/AVC standard. Then, the proposed interpolation method is described in greater detail.

a) H.264/AVC interpolation

The H.264/AVC standard[1,2] uses 1/4th pixel accuracy for the luma motion vectors. Figure 3 shows the integer-pixel samples (also called full pixel, shown in gray blocks with upper-case letters) from the reference frame, which are used to interpolate the fractional pixel (shown in white blocks with lower-case letters) samples. There are altogether 15 fractional pixel positions, labeled "a" through "o" in Figure 5. To obtain luma component at 1/2 pixel positions (b, h, and j), a 6-tap Wiener filter with coefficients [1, -5, 20, 20, -5, 1]/32 is used. For position j, the interpolation filter is applied first in the horizontal direction and then, in the vertical direction. To obtain luma component at 1/4th pixel locations, bilinear interpolation is used. To perform bilinear interpolation, the neighboring 1/2 pixel positions are calculated and intermediate rounding and clipping is performed. After that, upward rounding is always used for averaging. The combination of intermediate rounding and clipping of the 1/2 pixel positions and the biased upward rounding during bilinear interpolation effectively reduces the precision of the interpolation filters for the 1/4th pixel positions. By maintaining the 1/2 pixel positions in higher precision, the interpolation of the 1/4th pixel positions can be improved. This is referred to as high precision interpolation filtering and was first proposed in[14].

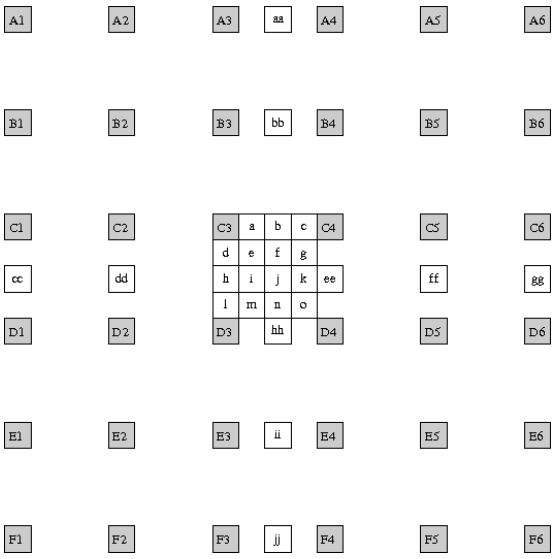

Figure 3. Fractional pixel positions for 1/4th pixel accuracy motion interpolation.

b) Single pass switched interpolation filters with offsets (single pass SIFO)

The basic idea behind switched interpolation filters is that at each of the 15 fractional pixel positions, an interpolation filter can be chosen from a set. For each fractional pixel position, the choice of the filter is signaled at the slice level. In addition, choice of offsets is also signaled for each slice as described in the following subsection. In our proposal, 4 different filter sets are defined. Each filter set consists of 15 filters, one for each fractional pixel position. The full pixel position is not filtered.

1. Filter set 0: This uses high precision filtering with the same filters as in H.264/AVC with the exception of position 'g', where a non-separable filter is used. Although the same filters as H.264/AVC are used, the intermediate data is kept in higher precision, eliminating intermediate rounding and clipping of the 1/2 pixel positions and the biased upward rounding during bilinear interpolation. For position 'g', the following filter is used (followed by right shift by 7 bits):

Table 1. Filter for fractional pixel position 'g' for filter set 0.

0	5	5	0
5	22	22	5
5	22	22	5
0	5	5	0

2. Filter sets 1 and 2: These filter sets are derived by using a set of training video sequences. For each set, positions a, b, and c use a six-tap horizontal filters. Positions d, h, and l use six-tap vertical filters. For the remaining fractional pixel positions, 4×4 non-separable filters are used. Each of the non-separable filters has horizontal, vertical or diagonal symmetry.

3. Filter set 3: This filter set uses an 8-tap separable filter in both horizontal and vertical directions for all the fractional pixel positions. Separate 8-tap filters are used for 1/4th pixel, 1/2 pixel, and 3/4th pixel positions. The 3 8-tap filters are as follows:

Table 2. Filter coefficients for filter set 3.

1/4th pixel position	[-3	12	-37	229	71	-21	6	-1]
1/2 pixel position	[-3	12	-39	158	158	-39	12	-3]
3/4th pixel position	[-1	6	-21	71	229	-37	12	-3]

After filtering, the result is normalized by adding 128 and shifting the result down by 8 bits and then clipped to the pixel range.

c) Direct filtering for 1/8th pixel accuracy motion vectors

To perform interpolation with 1/8th pixel accuracy, our proposal uses direct filters. For any 1/8th pixel position, these filters are derived from the filters used for the 1/4th pixel positions assuming bilinear interpolation. For determining the 1/8th pixel filters, it is assumed that filter set 3 is used for all the 1/4th pixel positions. Thus, interpolation for any 1/8th pixel position requires filtering with at most 2 8-tap filters (horizontal and vertical). There is no offset associated with the 1/8th pixel positions.

d) Choice of filter sets and offsets

Before encoding a frame, the encoder selects a filter for each fractional pixel position based on statistics gathered from previously encoded frames. In our proposal, the filter that minimizes prediction error for the previously encoded frames is selected. For each fractional pixel position, the minimization is performed only on blocks whose motion vector points to that fractional pixel location. The choice of filter remains the same irrespective of the reference frame in which the motion search is being performed.

For reference frame 0 from each list, offsets are sent to the decoder for each of the 15 fractional pixel positions as well as the full pixel position For other reference frames only one frame offset is sent.

5) Transforms for inter-prediction residuals

We will first discuss transforms for encoding inter-prediction residuals for non-geometry motion partitions. For motion partitions of size 8×8 and lower, the transform choices are identical to H.264/AVC. We reuse the 4×4 and 8×8 transforms from H.264/AVC[1,2]. As in H.264/AVC, these transforms can not be applied across motion boundaries. For motion partitions of sizes 16×16, 16×8, and 8×16, in addition to the 4×4 and 8×8 transforms, it is possible to apply a larger transform that is matched to the size of the motion partition. As an example, for an 8×16 motion partition, the transform choices are 4×4, 8×8, and 8×16. The choice of the transform is signaled to the decoder. For motion partitions of size 64×64, 64×32, and 32×32, only 16×16 transform can be used. Here we have adopted a variation of the encoder simplification suggested in [15] to disallow 4×4 and 8×8 transforms in motion partitions larger than 16×16. This speeds up the encoder substantially with very little effect on compression efficiency. We now will describe the design of the 16×16, 16×8, and 8×16 transforms, hereafter referred to as *Big Transforms*, in greater detail.

Our codec uses separable two-dimensional (2-D) transforms proposed in[16]. The transforms are integer scaled transforms that approximate Type II DCT[17]. Each transform coefficient needs to be multiplied by a scale factor to make the resulting transform orthogonal. The design of the proposed transforms is fully recursive, coinciding with LLM factorizations for 4-point and 8-point transforms[18]. In odd parts we use a modification of Plonka-Tashe Type-IV DCT factorization[19] in which all factors $1/\sqrt{2}$ are moved to the scaling stage[20].

a) Proposed 16-point transform

Figure 6 shows detailed flow-graph of the proposed one dimensional (1-D) 16 point transform. The upper (even) part of the transform uses a scaled 8-point transform (shown with a bounding box with solid lines), which in turn uses a scaled 4-point transform (shown with a bounding box with dotted lines). Similarly, the lower (odd) part of the transform uses two scaled 4-point transforms. The scaling factors for the 16-point transform are shown on the right hand side. The factors A, B, … , N satisfy the following relations:

$$\xi = \sqrt{A^2 + B^2},\ \varsigma = \sqrt{C^2 + D^2} = \sqrt{E^2 + F^2},\ \text{and}$$
$$\eta = \sqrt{G^2 + H^2} = \sqrt{I^2 + J^2} = \sqrt{K^2 + L^2} = \sqrt{M^2 + N^2}. \quad (3)$$

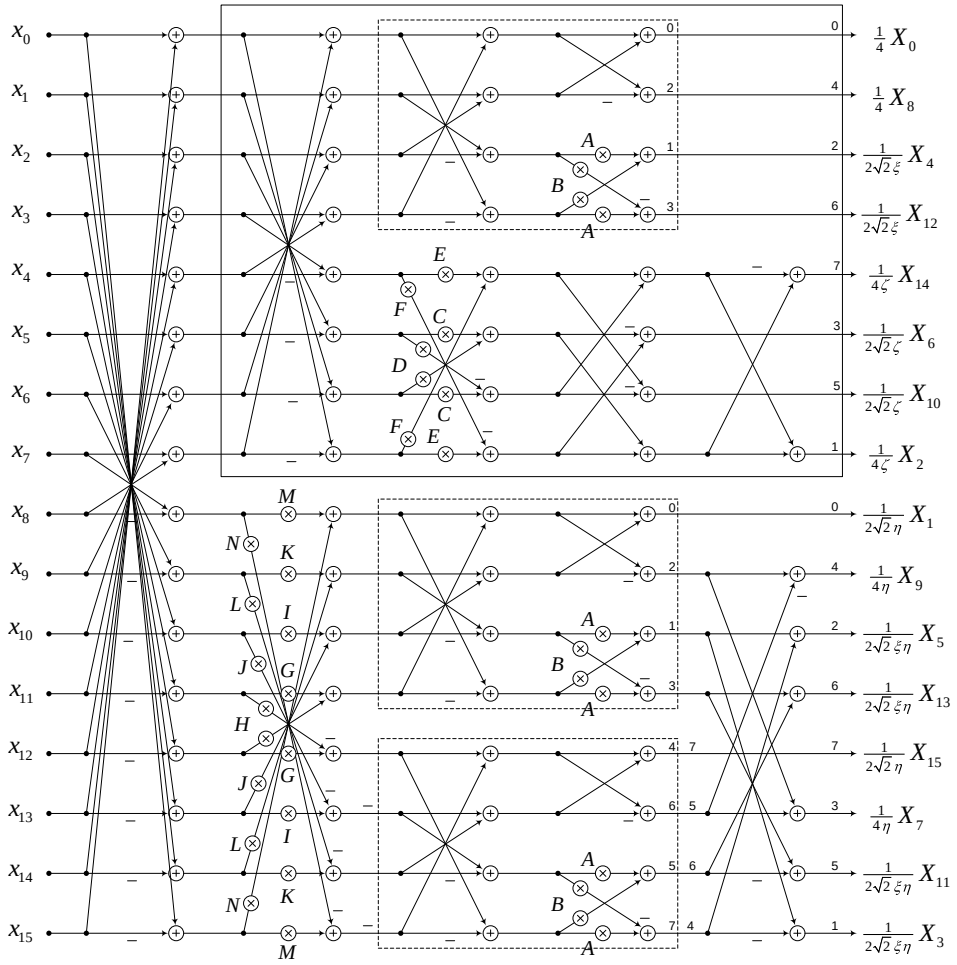

Figure 4. Recursive factorization of the 16-point transform. Dotted boxes show embedded 4-point transforms, and solid box in the upper part shows factorization of the 8-point transform.

This factorization involves 72 additions and 36 multiplications, which matches the complexity of best known rotation-based factorizations[17,18,21], and has the advantage of a fully recursive structure, reusing scaled 4-point and 8-point transforms. It should be noted that the described factorization only specifies a scaled 16-point transform, and that in order to map its output into actual transform coefficients, we will need to multiply them by the following factors:

$$S = \mathrm{diag}\left(\frac{1}{4}, \frac{1}{2\sqrt{2}\,\eta}, \frac{1}{4\,\zeta}, \frac{1}{2\sqrt{2}\,\xi\eta}, \frac{1}{2\sqrt{2}\,\xi}, \frac{1}{2\sqrt{2}\,\xi\eta}, \frac{1}{2\sqrt{2}\,\zeta}, \frac{1}{4\,\eta}, \frac{1}{4}, \frac{1}{4\,\eta}, \frac{1}{2\sqrt{2}\,\zeta}, \frac{1}{2\sqrt{2}\,\xi\eta}, \frac{1}{2\sqrt{2}\,\xi}, \frac{1}{2\sqrt{2}\,\xi\eta}, \frac{1}{4\,\zeta}, \frac{1}{2\sqrt{2}\,\eta} \right);$$

where ξ, ζ, η are normalization factors associated with 3 groups of factors inside the transforms[22]. These scaling factors are shown on the right hand side in Figure 4.

b) Integer factors

For the 1-D 16 point transform we choose the following set of butterfly factors.

Table 3. Butterfly factors in proposed 16-point transform.

A	B	C	D	E	F	N	L	J	H	G	I	K	M
2/4	5/4	19/32	4/32	16/32	11/32	6/64	11/64	21/64	27/64	34/64	38/64	42/64	43/64

In order to balance the dynamic range across the transform, we have introduced right shifts after multiplies. This allows transform coefficients to fit in the range [-1.25, 1.25], which is tight enough for practical purposes. The values of the resulting scale factors are:

$$\left[\frac{1}{4}, \frac{16\sqrt{3770}}{1885}, \frac{8}{\sqrt{377}}, \frac{64\sqrt{130}}{1885}, \frac{\sqrt{58}}{29}, \frac{64\sqrt{130}}{1885}, \frac{8\sqrt{754}}{377}, \frac{16}{\sqrt{1885}}, \frac{1}{4}, \frac{16}{\sqrt{1885}}, \frac{8\sqrt{754}}{377}, \frac{64\sqrt{130}}{1885}, \frac{\sqrt{58}}{29}, \frac{64\sqrt{130}}{1885}, \frac{8}{\sqrt{377}}, \frac{16\sqrt{3770}}{1885}\right] \quad (4)$$

After multiplication by 4 and conversion to floating point, these factors are:

$$[1.0, 2.085, 1.648, 1.548, 1.05, 1.548, 2.33, 1.474, 1.0, 1.474, 2.33, 1.548, 1.05, 1.548, 1.648, 2.085]\ . \quad (5)$$

Since constants A..N are integers (or dyadic rational numbers), we can replace multiplications with simple series of additions and shift operations. Moreover, we can do it for each pair of multiplies performed for each input variable in butterflies. Table 4 summarizes such multiplier less algorithms that can be employed in our designs.

Table 4. Implementation of multiplications by additions and shifts.

Factors		Algorithms: y=x*[A,C,E,G,I,K,M]; z=x*[B,D,F,H,J,L,N]	Complexity	Times used
A=2/4	B=5/4	y = x >> 1; z = x + (x >> 2);	1 add + 2 shifts	6
C=19/32	D=4/32	z = x >> 3; y = (x >> 1) + z- (z >> 2);	2 adds + 3 shifts	2
E=16/32	F=11/32	y = x >> 1; z = ((x + y) >> 2) - (y >> 4);	2 adds + 3 shifts	2
G=34/64	H=27/64	x1 = x >> 5; z = x1 + (x >> 1); x2 = x1 + z; y = x2 - (x2 >> 2);	3 adds + 3 shifts	2
I=38/64	J=21/64	x1 = x >> 4; x2 = x + x1; y = x1 + (x2 >> 1); z = y - (x2 >> 2);	3 adds + 3 shifts	2
K=42/64	L=11/64	x1 = x - (x >> 3); y = x1 - (x1 >> 2); z = (x – y) >> 1;	3 adds + 3 shifts	2
M=43/64	N=6/64	x1 = x - (x >> 2); z = x1 >> 3; x2 = x1 – z; y = x - (x2 >> 1);	3 adds + 3 shifts	2
Total:			38 adds +48 shifts	

As illustrated by this table, each pair of multiplications is computable with at most 3 additions and 3 shifts, making the maximum cost of each individual multiplication being half that much (i.e. 1.5 additions and 1.5 shifts). The complexity of the entire multiplier-less 16-point transform with such factors becomes 110 additions and 48 shifts. Indeed, on platforms with fast multiplications one can also implement it by using 72 additions and 32 multiplications by simply following the flow graph.

c) Proposed 8-point transform

We reuse the scaled 8 point transform from the upper (even) part of the 16 point transform. The butterfly factors are summarized in Table 5.

Table 5. Butterfly factors in proposed 8-point transform.

A	B	C	D	E	F
2/4	5/4	19/32	4/32	16/32	11/32

As follows from the flow-graph in Figure 4, the proposed 8-point transform needs 26 additions and 12 multiplications. If desired, the multiplication can be replaced by additions and shifts as in case of 16 point transform.

When implementing the transform to avoid accumulation of rounding errors, we pre-shift the 2-D input matrix to the left by 8 bits. After the transform, the transform coefficients are shifted to the right by the same amount after appropriate rounding. All the large transforms, namely, 16×16, 16×8, and 8×16 can be performed with 32 bits of precision.

2.3 Quantization

Quantization methods are unchanged from H.264/AVC[1,2]. On the encoder side, our proposal uses RD based quantization (RDO_Q) first proposed in[23] and discussed in greater detail in[24]. The RD based quantization mainly consists of 2 parts:

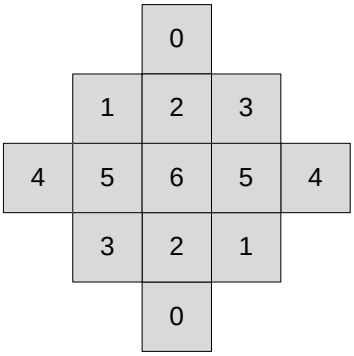

Figure 5. 5×5 symmetric filter with diamond shape support.

1. Trellis-based optimization of the quantization operation for transform coefficients: In trellis-based optimization, the quantizer index is chosen based on the rate-distortion cost of coding that index in a Lagrangian framework. Due to the manner in which entropy coding of quantized coefficients is performed in H.264/AVC, it is sometimes advantageous to quantize a coefficient to zero instead of rounding to the nearest quantizer level. This is because the rate may be lowered sufficiently to offset the increase in distortion. To keep the computation complexity manageable in the proposed video codec, in most cases, only 3 candidate quantizer indices are considered. These are 0, round-up, and round-down. For CABAC, the optimization is performed in 2 steps. In the first step, the last non-zero coefficient is chosen. Then in the 2nd step, the quantizer indices for individual coefficients are chosen.
2. Quantizing and coding a block with multiple quantizer step-sizes: Each block is encoded using a range of QP values. Then the QP value with the best rate-distortion cost is chosen and signaled to the decoder. In the proposed codec, for I and P slices, only a single QP value is used. For B slices, 3 QP values are used: ("QP, QP+2, and QP+3").

2.4 In-loop filtering

1) Deblocking filter

Our proposal uses the same deblocking filter as H.264/AVC with suitable modification for *BigBlocks*. Recall that for block sizes greater than 16×16, only 16×16 transform is used. Thus, for 32×32 and 64×64 blocks, deblocking filter is applied only along the 16×16 block edges. This reduces the computational complexity for the deblocking operation if the bigger blocks are chosen frequently.

2) Adaptive loop filtering

We used a modified form of the Quadtree-based Adaptive Loop Filter (QALF) proposed in[25]. Instead of a single filter used in QALF, our proposal uses a set of M filters. The set of M filters is transmitted to the decoder for each frame or a group of frames (GOP). Whenever the QALF segmentation map indicates that a block should be filtered, for each pixel, a specific filter from the set is chosen based on a measure of local characteristic of an image, called activity measure. Our proposal uses the sum-modified Laplacian measure as described in[25]. The sum-modified Laplacian for pixel (i,j) is calculated as follows:

$$var(i,j) = \sum_{k=-K}^{K} \sum_{l=-L}^{L} \left|2R_{i+k,j+l} - R_{i+k-1,j+l} + R_{i+k+1,j+l}\right| + \left|2R_{i+k,j+l} - R_{i+k,j+l-1} + R_{i+k,j+l+1}\right|, \qquad (6)$$

where $R_{i,j}$ refers to the reconstructed frame value for pixel (i,j). A 7×7 $(K, L = 3)$ neighborhood is used for calculation of the sum-modified Laplacian. The ranges of sum-modified Laplacian measure have to be sent to the decoder. Filter coefficients are coded using prediction from coefficients transmitted for previous frames. Our proposal uses 5×5, 7×7, and 9×9 filters with diamond shape support and symmetry as shown in Figure 5. Adaptive loop filtering for chroma is the same as that in the original QALF.

2.5 Lossless coding

Context adaptive binary arithmetic coding (CABAC)[26] is used in our proposed video codec for encoding of information such as block type, coded block pattern, motion vectors and transform coefficients. The context is based on the neighboring blocks in a similar way as in H.264/AVC. The differences from H.264/AVC are highlighted below.

1) Macroblock type

For a 64×64 block, a new syntax element, **mb64_type**, is introduced to indicate the motion partition for the block. This can be SKIP, DIRECT, 64×64, 64×32, 32×64 or P32x32. An **mb64_type** of P32×32 for a 64×64 block indicates that the block is split into four 32×32 blocks. Then, for each 32×32 block, a new syntax element, **mb32_type**, is sent indicating the motion partition for the block. An **mb32_type** of P16×16 for a 32×32 block indicates that the block is split into four 16×16 blocks. In that case, for each 16×16 block, the macroblock type, **mb_type**, is sent to the decoder.

*2) Coded block pattern (**cbp64** and **cbp32**)*

For a 64×64 block, a new one bit syntax element, **cbp64**, is introduced to indicate whether the whole 64×64 block has any nonzero coefficients. A nonzero **cbp64** value indicates that there is at least one nonzero transform coefficient. If **cbp64** is 1, for each 32×32 block, **cbp32** is encoded to indicate whether the whole 32×32 block has any nonzero coefficients. If **cbp32** is 1, for each 16×16 block, the current AVC/H.264 cbp is encoded to indicate its status.

*3) Change in luminance quantizer step-size (**mb64_delta_qp** and **mb32_delta_qp**)*

Our proposal permits the luminance quantizer step-size to change as follows. If a 64×64 block is partitioned into 4 separate 32×32 blocks, each 32×32 block can have its own QP. If a 32×32 is further partitioned into four 16×16 blocks, each 16×16 block can also have its own QP. This information is signaled to the decoder using **delta_qp** syntax. For a 64x64 block, if the **mb64_type** is not P32×32, **mb64_delta_qp** is encoded to signal the relative change in luminance quantizer step-size with respect to the block on the top-left side of the current block. The decoded value of **mb64_delta_qp** is restricted to be in the range [-26,25]. The **mb64_delta_qp** value is inferred to be equal to 0 when it is not present for any macroblock (including **P_Skip** and **B_Skip** macroblock types). The value of luminance quantization for the current block, QP_Y is derived as

$$QP_{\mathrm{Y}} = \left(QP_{\mathrm{Y,\,prev}} + \mathrm{mb64\_qp\_delta} + 52\right) \,\%\, 52. \tag{7}$$

where $QP_{\mathrm{Y,\,prev}}$ is the luminance quantization parameter of the previous 64x64 block in the decoding order in the current slice. For the first 64×64 block in the slice, $QP_{\mathrm{Y,\,prev}}$ is set equal to the slice QP sent in the slice header.

If **mb64_type** is P32×32, for each 32×32 block, the same process is repeated. That is, if **mb32_type** is not P16×16, **mb32_delta_qp** is encoded. Otherwise, **delta_qp** for each 16×16 macroblock is sent to the decoder as in AVC/H.264. It should be noted that when **delta_qp** is signaled at the 64×64 or 32×32 block size, it is applicable to all the blocks in the motion partition.

4) Adaptive motion vector resolution

For each block in a motion partition, a motion vector resolution flag is encoded. A value of 1 (0) implies 1/8th pixel (1/4th pixel) motion vector resolution is used for that motion vector. We will describe the contexts used for CABAC encoding of motion vector resolution flag and motion vector differences (MVD).

For CABAC encoding of the motion vector resolution flag, four contexts are used. The contexts are defined based on the motion resolution of neighboring partitions. Let A be the left neighboring partition and let B be the upper neighboring partition. The precise definition of neighboring partitions is the same as that used in the H.264/AVC MVD encoding. The four contexts used to encode the motion vector resolution flag for the current block are:

1. Both A and B have 1/8th pixel motion accuracy.
2. A has 1/4th pixel motion accuracy and B has 1/8th pixel motion accuracy.
3. A has 1/8th pixel motion accuracy and B has 1/4th pixel motion accuracy.
4. Both A and B have 1/4th pixel motion accuracy.

The encoder always maintains the motion vector (MV) and MVD information at 1/8th pixel resolution (by left-shift if necessary). Then, the MV prediction for the current block is formed with 1/8th pixel accuracy. If the current block has

only 1/4th pixel motion accuracy, the MV prediction is converted to 1/4th pixel accuracy be right-shifting and then the MVD is formed. On the other hand if the current block has 1/8th pixel motion accuracy, the MVD is formed directly by subtracting the MV prediction from the motion vector for the current block. Once the MVD is formed, if the current block has 1/4th pixel accuracy, for all the neighboring blocks used for determining the MVD contexts, the MVDs are converted to 1/4th pixel accuracy. Similar procedure is followed for 1/8th pixel accuracy. The encoding of MVD is performed as specified in H.264/AVC.

3. EXPERIMENTAL RESULTS

In this section, the quantitative results from our proposal in response to CfP are presented. The CfP[3,4] defined two sets of test conditions, Constraint case 1 and Constraint set 2 as follows:

- Constraint set 1: Structural delay of processing units not larger than 8-picture "groups of pictures (GOPs)" (e.g., dyadic hierarchical B usage with 4 levels), and random access intervals of 1.1 seconds or less.
- Constraint set 2: No picture reordering between decoder processing and output, with bit rate fluctuation characteristics and any frame-level multi-pass encoding techniques to be described with the proposal. (A metric to measure bit rate fluctuation is implemented in the Excel file to be submitted for each proposal.)

Three anchors were generated using JM16.2[27]:

- Alpha anchor (satisfies constraint set 1),
- Beta anchor (satisfies constraint set 2),
- Gamma anchor (satisfies constraint set 2).

Constraint set 1 results from our proposal are compared with the alpha anchor. Constraint set 2 results from our proposal are compared with the beta anchor and gamma anchor. There are five RD points generated for each sequence. The results are reported in terms of BD-rate[28,29]. In Tables 6-8, the results are reported both for high BD-rate and low BD-rate. For calculation of high BD-rate, rates R5-R4-R3-R2 are used and for calculation of low BD-rate, rates R4-R3-R2-R1 are used.

Table 6. Coding gain relative to alpha anchor (constraint set 1).

Class	Seq. Name	BD Low (%)	BD High (%)
Class A 4kx2k	Traffic	-33.23	-31.15
	PeopleOnStreet	-19.55	-20.33
	Avg_4kx2k	-26.39	-25.74
Class B 1080p	Kimono1	-39.38	-39.79
	ParkScene	-29.00	-26.58
	Cactus	-31.24	-30.62
	BasketballDrive	-35.67	-34.90
	BQTerrace	-34.45	-45.36
	Avg_1080p	-33.95	-35.45
Class C WVGA	BasketballDrill	-30.60	-30.69
	BQMall	-33.46	-31.84
	PartyScene	-33.16	-32.10
	RaceHorses	-29.69	-26.81
	Avg_WVGA	-31.73	-30.36
Class D WQVGA	BasketballPass	-22.81	-21.73
	BQSquare	-43.55	-44.41
	BlowingBubbles	-26.32	-27.08
	RaceHorses	-21.05	-19.83
	Avg_WQVGA	-28.43	-28.26
	Overall Avg	**-30.88**	**-30.88**

Table 7. Coding gain relative to beta anchor (constraint set 2).

Class	Seq. Name	BD Low (%)	BD High (%)
Class B 1080p	Kimono1	-40.42	-44.78
	ParkScene	-29.48	-26.28
	Cactus	-31.71	-31.57
	BasketballDrive	-41.39	-40.69
	BQTerrace	-45.58	-50.43
	Avg_1080p	-37.72	-38.75
Class C WVGA	BasketballDrill	-28.92	-28.35
	BQMall	-31.92	-30.87
	PartyScene	-28.54	-24.10
	RaceHorses	-27.96	-26.76
	Avg_WVGA	-29.34	-27.52
Class D WQVGA	BasketballPass	-23.29	-22.70
	BQSquare	-34.85	-32.96
	BlowingBubbles	-15.42	-16.66
	RaceHorses	-20.44	-20.27
	Avg_WQVGA	-23.50	-23.15
Class E 720p	Vidyo1	-46.57	-46.24
	Vidyo3	-40.15	-42.55
	Vidyo4	-37.30	-45.31
	Avg_720p	-41.34	-44.70
	Overall Avg	**-32.75**	**-33.16**

Table 8. Coding gain relative to gamma anchor (constraint set 2).

Class	Seq. Name	BD Low (%)	BD High (%)
Class B 1080p	Kimono1	-53.79	-61.11
	ParkScene	-48.97	-42.88
	Cactus	-50.90	-48.65
	BasketballDrive	-55.00	-56.21
	BQTerrace	-47.57	-68.66
	Avg_1080p	-51.25	-55.50
Class C WVGA	BasketballDrill	-47.25	-47.02
	BQMall	-45.36	-46.30
	PartyScene	-49.30	-47.60
	RaceHorses	-36.65	-35.47
	Avg_WVGA	-44.64	-44.10
Class D WQVGA	BasketballPass	-34.36	-33.77
	BQSquare	-63.71	-59.53
	BlowingBubbles	-40.44	-39.69
	RaceHorses	-28.24	-26.65
	Avg_WQVGA	-41.69	-39.91
Class E 720p	Vidyo1	-60.05	-60.38
	Vidyo3	-54.25	-56.85
	Vidyo4	-46.24	-61.24
	Avg_720p	-53.51	-59.49
	Overall Avg	**-47.63**	**-49.50**

4. CONCLUSIONS

In response to the joint call for proposal issued by ISO/IEC JTC 1/SC 29/WG 11 (MPEG) and ITU-T SG 16 Q6, Qualcomm Inc. submitted a video coding technology proposal. In this paper, the key features and algorithms of the proposed codec have been described. We have also presented performance results in comparison to the anchors generated by the H.264/AVC JM16.2 reference encoder. For constraint set 1 (random access), an average BD-rate reduction of 30.88% is achieved by the proposed codec. Similarly, for constraint set 2, compared to the beta and gamma anchors, average BD-rate reductions of 32.96% and 48.57%, respectively, are achieved. Thus, the proposed codec significantly outperforms H.264/AVC both for random access and low delay constraints.

REFERENCES

[1] ITU-T Recommendation H.264, "Advanced video coding for generic audiovisual services," Mar 2009.

[2] ISO/IEC 14496-10:2009, "Information technology -- Coding of audio-visual objects -- Part 10: Advanced Video Coding," 2009.

[3] ISO/IEC JTC1/SC29/WG11, "Joint Call for Proposals on Video Compression Technology," MPEG output document N11113, January 2010.

[4] ITU-T SG16/Q6, "Joint Call for Proposals on Video Compression Technology," ITU-T SG16/Q6 output document VCEG-AM91, January 2010.

[5] M. Karczewicz, P. Chen, R. Joshi, X. Wang, W-J. Chien, and R. Panchal, "Video coding technology proposal by Qualcomm Inc.," JCT-VC input document JCTVC-A121, Dresden, Germany, April 2010.

[6] JMKTA software, http://iphome.hhi.de/suehring/tml/download/KTA/

[7] T. Chujoh and R. Noda, "Internal bit depth increase for coding efficiency," ITU-T SG16/Q6 input document VCEG-AE13, Marrakech, Morocco, January 2007.

[8] T. Chujoh, N. Wada and G. Yasuda, "Quadtree-based Adaptive Loop Filter," ITU-T SG16/Q6 input document C181, Geneva, January 2009.

[9] Y. Ye and M. Karczewicz, "Improved Intra Coding," ITU-T SG16/Q6 input document C257, Geneva, Switzerland, June 2007.

[10] Y. Ye and M. Karczewicz, "Improved Intra Coding," ITU-T SG16/Q6 input document VCEG-AG11, Shenzhen, China, 20 October, 2007.

[11] P. Chen, Y. Ye, and M. Karczewicz, "Video Coding Using Extended Block Sizes," ITU-T SG16/Q6 input document C123, Jan. 2009.

[12] Ò. Divorra, P. Yin, C. Dai, and X. Li, "Geometry-adaptive block partitioning for video coding," Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing, 2007, pp. I-657-660.

[13] Ò. Divorra, P. Yin, and C. Gomila "Geometry-adaptive Block Partitioning", ITU-T SG16/Q6 input document, VCEG-AF10, Geneva, Switzerland, April 2007.

[14] M. Karczewicz, Y. Ye, P. Chen, and G. Motta, "Single Pass Encoding using Switched Interpolation Filters with Offset, " ITU-T SG16/Q6 input document VCEG-AJ29, San Diego, CA, USA, Oct. 2008.

[15] T. Yamamoto, Y. Yasugi, and T. Ikai, "Further result on constraining transform candidate in Extended Block Sizes," VCEG input document VCEG-AL19, London, UK / Geneva, CH, July 2009.

[16] R. Joshi, Y. Reznik, and M. Karczewicz, "Simplified Transforms for Extended Block Sizes," ITU-T SG16/Q6 input document VCEG-AL19, London, UK / Geneva, CH, July 2009.

[17] V. Britanak, P. Yip, and K. R. Rao, "Discrete Cosine and Sine Transforms: General Properties, Fast Algorithms and Integer Approximations," Academic Press, 2006.

[18] C. Loeffler, A. Ligtenberg, and G. S. Moschytz, "Algorithm-architecture mapping for custom DCT chips," Proc. Int. Symp. Circuits Syst., pp. 1953-1956, Helsinki, Finland, June 1988.

[19] G. Plonka and M. Tashe, "Fast and numerically stable algorithms for discrete cosine transforms," Linear Algebra and Applications, vol. 394, no. 1, pp. 309-345, 2005.

[20] R. Joshi, Y. Reznik, and M. Karczewicz, "Efficient Large Size Transforms for High-Performance Video Coding," Proc. SPIE 7798, pp. 779831 1-7.

[21] Y. A. Reznik and R. K. Chivukula, "Design of Fast Transforms for High-Resolution and Video Coding," Proc. SPIE 7443, pp. 744312 1-17, 2009.

[22] Y. A. Reznik, A. T. Hinds, and J. L. Mitchell, "Improved precision of fixed-point algorithms by means of common factors," Proc. International Conference on Image Processing, pp. 2344-2347, 2008.
[23] M. Karczewicz, Y. Ye, and I. Chong, "Rate distortion optimized quantization, " ITU-T SG16/Q6 input document VCEG-AH21, Jan. 2008
[24] M. Karczewicz, P. Chen, Y. Ye, and R. Joshi, "R-D based quantization in H.264," Proc SPIE 7443, pp. 744314 1-8, 2009.
[25] W-J. Chien, M. Karczewicz, "Adaptive Filter Based on Combination of Sum-Modified Laplacian Filter Indexing and Quadtree Partitioning," ITU-T SG16/Q6 input document VCEG-AL27r1, London, UK, July 2009.
[26] D. Marpe, H. Schwarz, and T. Wiegand, "Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC video compression standard," IEEE Transactions on Circuits and Systems for Video Technology, vol. 13, No. 7, pp. 620-636, July 2003.
[27] H.264/AVC Reference Software (JM16.2), http://iphome.hhi.de/suehring/tml/download/old_jm/jm16.2.zip
[28] G. Bjontegaard, "Calculation of average PSNR differences between RD-curves," ITU-T SG16/Q6 input document VCEG-M33, Austin, USA, April 2001
[29] G. Bjøntegaard, "Improvements of the BD-PSNR model", ITU-T SG16/Q6 input document VCEG-AI11, Berlin, Germany, July, 2008.