A hybrid video coder based on extended macroblock sizes, improved interpolation and flexible motion representation

Marta Karczewicz, Peisong Chen, Rajan L. Joshi, Xianglin Wang, Wei-Jung Chien, Rahul Panchal, Yuriy Reznik, Muhammed Coban, and In Suk Chong

Abstract— This paper describes a video coding technology proposal submitted by Qualcomm 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. The 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), single pass switched interpolation filters with offsets, mode dependent directional transforms for intra-coding, luma and chroma high precision filtering, geometric motion partitions, adaptive motion vector resolution and efficient 16 point transforms. It also incorporates internal bit-depth increase and modified quadtree-based adaptive loop filtering. Simulation results are presented to demonstrate the high compression efficiency achieved by the proposed video codec at the expense of moderate increase in encoding and decoding complexity compared to the AVC/H.264 standard. For the Random Access and Low Delay configurations, it achieved average bit rate reductions of 30.9% and 33.0% for equivalent PSNR, respectively, compared to the corresponding AVC anchors. The proposed codec scored highly in both subjective evaluations and objective metrics and was among the best-performing CfP proposals.

***Index Terms* — Geometric motion partitions, MDDT, adaptive motion vector resolution, switched interpolation filters with offsets, video coding**

I. INTRODUCTION

This paper describes a video coding technology proposal submitted by Qualcomm in response to a joint call for proposal (CfP) [1], [2] issued by ITU-T SG16 Q.6 (VCEG) and ISO/IEC JTC1/SC29/WG11 (MPEG) in January 2010. Details regarding the CfP process, test set, coding constraint conditions, subjective evaluation methodology, the AVC/H.264 [3], [4] anchors used, etc. can be found in the introduction to the special section on the call for proposals on high efficiency video coding standardization of this issue [5]. Qualcomm's proposal [6] scored highly in both subjective evaluations and objective metrics and was among the best-performing CfP proposals. Some of the key features of Qualcomm's proposal are block sizes larger than the traditional 16×16 macroblock structure, transforms of sizes 16×16, 16×8, and 8×16, in addition to 4×4 and 8×8, mode dependent directional transforms (MDDT) for intra-coding, luma high precision filtering, single-pass switched interpolation filters with offsets (single-pass SIFO), geometric motion partitions, adaptive motion vector resolution, and efficient 16 point transforms.

The proposed video codec utilized some coding tools proposed by other companies and adopted into the JM Key Technology Areas (JMKTA) software [7] such as internal bit-depth increase (IBDI) [8], and modified quadtree-based adaptive loop filtering (QALF) [9]. Several other tools such as chroma high precision filtering, direct mode for P slices, motion vector scaling, and changes to the AVC/H.264 mode syntax for B slices were also included in the proposed video codec. However, due to a limitation of space, these techniques will not be described in this paper. Interested readers are referred to [6] for a complete description of these techniques.

The rest of this paper is organized as follows. In Section II, the details of the key features of the proposed codec are provided in the context of the different functional blocks. In Section III, experimental results are presented. The performance of the proposed video codec is evaluated under low delay and random access conditions. The coding results are compared to the AVC/H.264 anchors provided by the CfP. Results of subjective evaluations and complexity comparisons are also provided. Finally, conclusions are presented in Section IV.

II. CODEC DESIGN

The proposed codec is based on the traditional hybrid coding approach, utilizing motion-compensated temporal prediction between video frames as well as intra-frame prediction, followed by 2-D transformation of the spatial

Manuscript received August 2, 2010; revised October 11, 2010; accepted October 19, 2010.

Authors are with Qualcomm, Inc., 5775 Morehouse Dr., San Diego, CA 92121, USA. The corresponding author is Marta Karczewicz (Tel. +1-858-658-5673; e-mail: martak@qualcomm.com).

residual signals, quantization, and entropy coding. The codec operates in a closed-loop and uses deblocking as well as quadtree-based adaptive loop filtering. The different functional blocks of the proposed codec are discussed in greater detail in the following subsections.

A. *Intra-block coding*

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

1) *Mode dependent directional transforms for intra-prediction residuals*

To exploit the directionality of the intra-prediction residuals, the proposed video codec uses mode dependent directional transforms (MDDT). Since the transform is dependent on the mode, no side information is necessary, which is rather important for the smaller block sizes such as 4x4 and 8x8. MDDT was first proposed in [10], [11]. We briefly describe the design and 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, for a 2-D residual block, KLT is a non-separable transform. For an $N \times N$ block, the KLT matrix size is $N^2 \times N^2$. Thus, KLT is prohibitively expensive in terms of storage and computational requirement. Our proposed video codec uses a separable $N \times N$ directional transform, which can be described as

$$Y = C_i \; X \; R_i \qquad (1)$$

where C_i and R_i are the column and row transform matrices for the 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. The training set used to design the MDDT matrices consisted of QCIF and CIF sequences. It did not include any of the sequences from the CfP test set.

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 AVC/H.264, zigzag scanning order is used so that the lower frequency coefficients are positioned earlier in the scan. However, in the 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 non-zero 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 the entropy coding efficiency. Quantized transform coefficients corresponding to 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, the 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 coefficient scanning order is updated according to the count statistics collected.
4. The collected count statistics are scaled down if the maximum count exceeds a threshold. This gives more importance to the recent past, resulting in better adaptivity.
5. The updated scanning order is used for the coding of the future blocks. The control returns to step 2 until the encoding of the slice is completed.

The initialization is performed based on the probability of each transform coefficient being non-zero. The scanning order is initialized in the decreasing order of the probability of a coefficient being non-zero. The probabilities are derived from the same training set used in the design of the MDDT matrices.

B. *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 geometric motion partitions are introduced to better align the motion partition to the video content. Also, the use of higher precision motion vector representation and improved sub-pixel interpolation, further improve the efficiency of motion compensation.

1) *Extended block size motion partition*

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 [12] 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 the 64×64 and 32×32 blocks, skip and direct modes are also used as in the case of 16×16 macroblocks in AVC/H.264.

In the proposed codec, 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. 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 the 16×16 blocks may be intra-coded.

2) Geometric motion partitions

In AVC/H.264, 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 [13], [14] is to use another kind of motion partitioning known as *geometric* motion partitions. This motion partitioning 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.

The geometric motion partitions are created as follows. The origin is assumed to be at the center of the block. Then, each geometric partition is defined by a line passing through the origin that is perpendicular to the line defining the partition boundary. This is shown in Fig. 1. The geometric 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

$$y = \frac{-1}{\tan\theta} x + \frac{\rho}{\sin\theta} = m x + c \quad (2)$$

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.

For each block size, 32 different values of θ are permitted (from 0 to 360 in steps of 11.25). The number of different values for ρ depends on the block size. For the 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 geometric partitions, respectively.

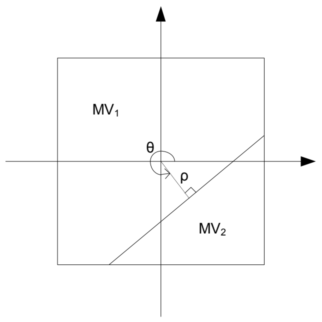


Fig.1. Parameters defining a geometric motion partition.

a) Motion search for geometric motion partitions

Since there are so many possible geometric partitions for each block size, it is prohibitively expensive for the encoder to do motion estimation for each region of each geometric 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 to speed-up the motion vector search for geometric 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 geometric partitions. For each geometric 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. Fig. 2 shows an example of this process. Suppose that we are interested in calculating the RD cost of a geometric motion partition represented by the line shown in Fig. 2. In that case, for the region above the line, the 4×8 block is the largest block for which a motion vector is already available. The motion vector for the 4×8 block is assigned to the region above the line. Similarly the motion vector for the 8×8 block shown in Fig. 2 is assigned to the geometric partition region below the line. If a geometric partition at block size of 16×16 is being considered, all the blocks of sizes 16×8, 8×16, 8×8, 8×4, 4×8, and 4×4 are considered 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 geometric partition and performing overlapped motion compensation as described below, the motion cost is evaluated using the sum of absolute differences (SAD). For a 16×16 block, a motion cost is calculated for each possible geometric partition. Then, 16 geometric partitions with the best motion costs are selected from 256 possible partitions. Full enhanced predictive zonal search (EPZS) [15] is performed on each partition region of each of the 16 partitions. This is followed by the calculation of the true rate-distortion (RD) cost. The geometric 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

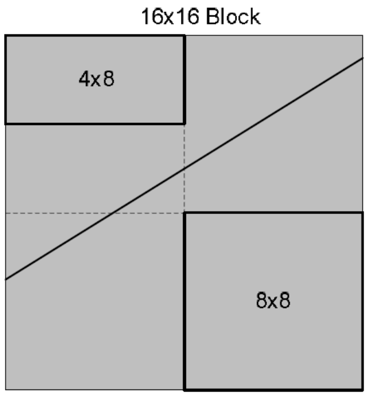


Fig. 2. Reusing rectangular partition motion vectors for geometric partition motion search.

determine whether geometric partitioning should be used for that particular block.

For the 32×32 and 64×64 blocks, a similar strategy is followed with a slight variation. For these block sizes, instead of evaluating the SAD motion cost for each geometric partition, it is evaluated for a subset of the possible geometric partitions. This subset is obtained by subsampling ρ and θ by 2. Thus, the SAD motion cost is evaluated for 128 geometric partitions for a 32×32 block and 256 geometric partitions for a 64×64 block. Then, for each extended block size, 2 geometric partitions having the best SAD motion costs are chosen. Let one of the geometric partitions chosen have parameters ρ_1 and θ_1. Then, true RD costs for geometric partitions with $\rho = \rho_1 - 1, \rho_1 + 1$ and $\theta = \theta_1 - 11.25^\circ, \theta_1, \theta_1 + 11.25^\circ$ are evaluated using the EPZS search. Similar process is repeated for the other geometric partition chosen in the first stage. The geometric partition with the lowest RD cost is chosen and compared against the RD cost for the optimal rectangular partitioning of the corresponding block size.

b) Overlapped motion compensation for geometric partitions

Since two different motion vectors are used for motion compensation inside a block with geometric partition, the pixels at the partition boundary may have large discontinuities that can produce visual artifacts similar to blockiness. Furthermore, since the geometric partition boundary may not be aligned with the macroblock and sub-macroblock boundaries, it is likely that the deblocking filter may not be able to reduce the blockiness resulting from the geometric motion partitions. To alleviate this, we apply the principle behind overlapped block motion compensation (OBMC) to the geometric motion partitions. Let the two regions created by a geometric 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). Fig. 3 shows an example where dark mesh squares belong to the boundary of

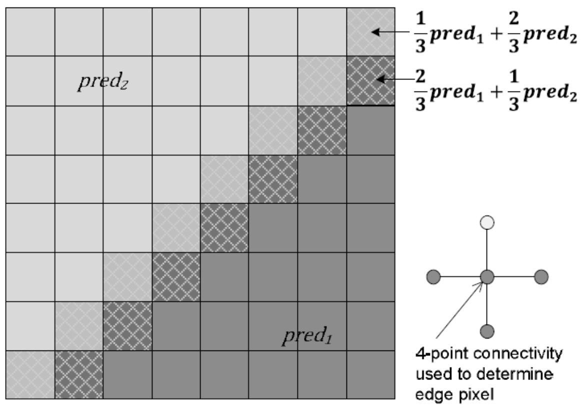

$\frac{1}{3}pred_1 + \frac{2}{3}pred_2$
$\frac{2}{3}pred_1 + \frac{1}{3}pred_2$
$pred_2$
$pred_1$
4-point connectivity used to determine edge pixel

Fig. 3. Overlapped motion compensation for geometric partitions.

region 1 and light 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, 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.

3) Motion accuracy

The AVC/H.264 standard allows motion vectors having 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 the 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 will be provided in section II.E.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 Fig. 4, 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

a) Luma interpolation

In the proposed video codec, single pass *switched*

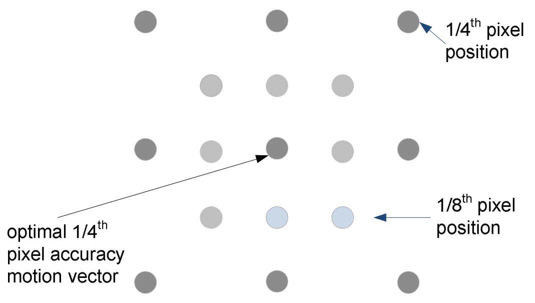


Fig. 4. Motion search for 1/8th pixel accuracy.

interpolation filters with offsets (single pass SIFO) are used to interpolate the reference frame to 1/4th pixel accuracy for the luma component. The single pass SIFO filters were first proposed in [16]. First we review the interpolation methods used in the AVC/H.264 standard. Then, the proposed interpolation method is described in greater detail.

(1) AVC/H.264 interpolation

The AVC/H.264 standard uses 1/4th pixel accuracy for the luma motion vectors. Fig. 5 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 Fig. 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. These are rounded and clipped to the original input bit-depth (for example 8 bits). After that, the 1/4th pixel locations are obtained by averaging using upward rounding. 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 16 bit or higher precision, the interpolation of the 1/4th pixel positions can be improved by eliminating intermediate rounding and clipping of the 1/2 pixel positions to input bit-depth and the biased upward rounding during bilinear interpolation This is referred to as high precision interpolation filtering. This was first proposed in [16] and used in the proposed video codec.

(2) 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

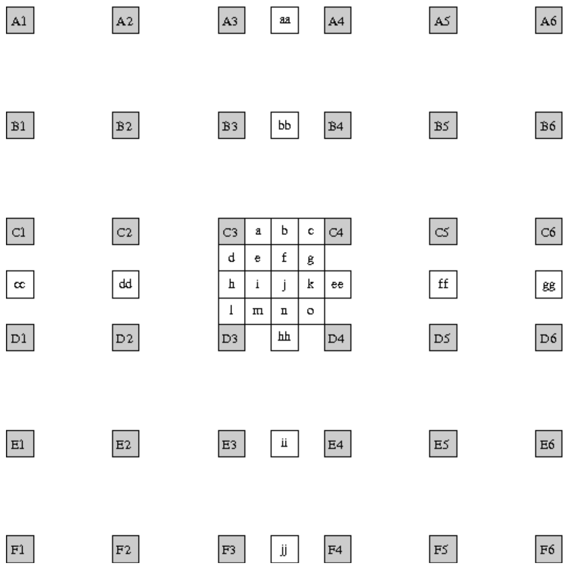


Fig. 5. Fractional pixel positions for 1/4th pixel accuracy motion interpolation.

the following subsection. The advantage of this interpolation method is that unlike various adaptive interpolation filters proposed in the literature, it does not require multiple passes through the frames. The frequency responses of filters allowed for each fractional pixel position need to have enough diversity to cater to different type of video content. At the same time, having too many filters in each set can increase the amount of information necessary to be signaled at the slice level. In our proposed codec, at each fractional pixel location, four different interpolation filters are permitted. Thus, four different filter sets are defined, each set consisting 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 AVC/H.264 with the exception of position 'g', where a non-separable filter, as shown in Table I, is used (followed by right shift by 7 bits).

TABLE I
FILTER FOR FRACTIONAL PIXEL POSITION 'G' FOR FILTER SET 0.

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

2. Filter set 1 and set 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 three 8-tap filters are shown in Table II. After filtering, the result is normalized by adding 128 and shifting the result down by 8 bits and then clipped to the pixel range.

TABLE II
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]

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

To perform interpolation with 1/8th pixel accuracy, our proposal uses direct filters for computationally efficiency. When IBDI is not being used, it also provides coding gains because of lack of intermediate rounding and clipping of the 1/4th pixel positions as well as lack of biased upward rounding in averaging. 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. As an example, for 3/8th pixel position the following 8-tap direct filter used is used:

[-6, 24, -76, 387, 229, -60, 18, -4].

Here we have not shown explicit rounding and clipping at the end of the filtering process. Thus, interpolation for any 1/8th pixel position requires filtering with at most two 8-tap filters (horizontal and vertical). The proposed codec uses no offset associated with the 1/8th pixel positions. This is done due to the trade-off between the amount of side information that needs to be sent and coding gain. For chroma interpolation, the motion vectors in the proposed codec were restricted to 1/8th pixel accuracy and high precision filtering was used with 6-tap filters. The details can be found in [6].

(4) Choice of filter set and offsets

Before encoding a frame, the encoder selects a filter for each fractional pixel position based on statistics gathered from previously encoded frames of the same type (P or B). In our proposal, the filter that minimizes the sum of squared prediction errors 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. The offsets provide significant gains for video sequences with illumination changes.

5) Transforms for inter-prediction residuals

We will first discuss transforms for encoding inter-prediction residuals for non-geometric motion partitions. For motion partitions of size 8×8 and lower, the transform choices are identical to AVC/H.264. We reuse the 4×4 and 8×8 transforms from AVC/H.264. As in AVC/H.264, these transforms can not be applied across motion boundaries. For motion partition 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 [17] 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 proposal uses separable two-dimensional (2-D) transforms discussed in [18]. The transforms are integer scaled transforms that approximate Type II DCT [19]. 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 and based on the LLM factorization for the 4 point and 8 point transforms [20].

a) Proposed 16-point transform

Fig. 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:

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

This factorization involves $4 \times 16 + 8 = 72$ additions and $16 + 8 + 3 \times 4 = 36$ multiplications, which matches the complexity of the best known rotation-based factorizations [19], [20], and has the advantage of a fully recursive structure, reusing scaled 4-point transforms. It should be noted that the above matrix only specifies a *scaled* 16-point transform, and that in order to map its output into full transform coefficients, we will need to multiply them by the scaling factors shown on the right hand side in Fig. 6.

b) Transform Matrix

For the 1-D 16 point transform we choose the butterfly factors shown in Table III. Here, in order to balance the dynamic range across the transform, we have introduced right shifts after multiplies. The transform coefficients now fit in the range [-1.25, 1.25], which is tight enough for practical purposes. 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. The complexity of the entire multiplier-less 16-point transform with such factors becomes 110 additions and 48 shifts. The details can be found in [21]. On the other hand, on platforms with fast multiplications one can also implement it by using 72 additions and 32 multiplications by simply following the flow graph.

TABLE III
BUTTERFLY FACTORS FOR THE 1-D 16 POINT TRANSFORM.

$A = 2/4$	$B = 5/4$	$C = 19/32$	$D = 4/32$
$E = 16/32$	$F = 11/32$	$G = 34/64$	$H = 27/64$
$I = 28/64$	$J = 21/64$	$K = 42/64$	$L = 11/64$
$M = 43/64$	$N = 6/64$		

c) Proposed 1-D 8 point transform

We reuse the scaled 8 point transform from the upper (even) part of the 16 point transform. The butterfly factors A – F are the same as in Table III. The 1-D 8 point transform needs 26 additions and 12 multiplications. If desired, the multiplication can be replaced by additions and shifts as in the case of the 16 point transform.

When implementing the transforms, 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 bigger transforms, namely, 16×16, 16×8, and 8×16 can be implemented with 32 bits of precision.

C. Quantization

Quantization methods are unchanged from AVC/H.264. Since 16×16, 16×8, and 8×16 transforms used in the proposed codec are scaled transforms, the scaling factor for each transform coefficient is absorbed into the quantization step. On the encoder side, our proposal uses RD based quantization (RDO_Q) first proposed in [22] and discussed in greater detail in [23]. The RD based quantization mainly consists of 2 parts:

1. Trellis-based optimization of the quantization operation for transform coefficients: In the trellis-based optimization, the quantizer index is chosen based on the RD cost of coding that index in a Lagrangian framework. Due to the manner in which entropy coding of quantized coefficients is performed in AVC/H.264, 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. In the proposed video codec, to keep the computation complexity manageable, only 3 candidate quantizer indices are considered in most cases. 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 RD 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, QP+3).

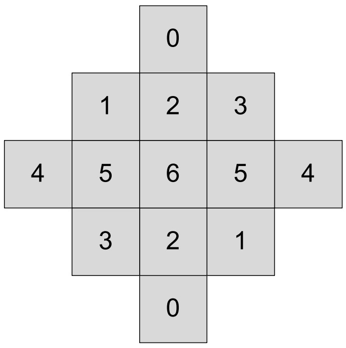


Fig. 7. 5×5 symmetric filter with diamond support.

D. In-loop filtering

1) Deblocking filter

Our proposal uses the same deblocking filter as AVC/H.264 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 [24]. 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 the activity measure. Our proposal uses the sum-modified Laplacian measure as described in [24]. The sum-modified Laplacian was first proposed in [25] as a measure of image focus. It is a discrete approximation to the modified Laplacian. The sum-modified Laplacian for pixel (i, j) is calculated as follows:

$$\mathrm{var}(i,j) = \sum_{k=-K}^{K}\sum_{l=-L}^{L}\left|2R_{i+k,j+l} - R_{i+k-1,j+l} + R_{i+k+1,j+l}\right| + \left|2R_{i+k,j+l} - R_{i+k,j+l-1} + R_{i+k,j+l+1}\right|, \tag{4}$$

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 Fig. 7. The numbers inside the squares indicate symmetry. Thus, pixels with index 1 have the same filter coefficient. The diamond shape support was chosen because it offers a good trade-off between complexity and performance. Adaptive loop filtering for chroma is the same as that in the original QALF.

E. *Entropy coding*

Context adaptive binary arithmetic coding (CABAC) [26] is used in the 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 AVC/H.264. The differences from AVC/H.264 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. A **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, $\mathrm{QP_Y}$, is derived as

$$\mathrm{QP_Y} = (\mathrm{QP_{Y,prev}} + \mathrm{mb64\_qp\_delta} + 52)\%52. \qquad (5)$$

where $\mathrm{QP_{Y,prev}}$ 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, $\mathrm{QP_{Y,prev}}$ 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 AVC/H.264 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 by 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 AVC/H.264.

III. Experimental Results

A. *Objective performance*

In this section, the quantitative results from our proposal in response to CfP are presented. The objective performance is compared with the Alpha and Beta anchors [5] generated by JM 16.2 [27] for constraint sets 1 and 2, respectively. Constraint set 1 (CS1) results for the proposed codec are compared with the alpha anchor. Constraint set 2 (CS2) results for the proposed codec are compared with the beta anchor. Five RD points were generated for each sequence. The results are reported in terms of BD-rate [28], [29]. In Table IV, the results are reported by averaging the high BD-rate and the low BD-rate. For the calculation of high BD-rate, the four highest rates are used and for the calculation of low BD-rate, four lowest rates are used.

TABLE IV
CODING GAIN IN TERMS OF BD-RATE RELATIVE TO THE ALPHA ANCHOR FOR CONSTRAINT SET 1 AND BETA ANCHOR FOR CONSTRAINT SET 2.

Class	Seq. Name	CS1 BD-rate (%)	CS2 BD-rate (%)
Class A 4kx2k	Traffic	-32.2	
	PeopleOnStreet	-19.9	
	Avg_4kx2k	-26.1	
Class B 1080p	Kimono1	-39.6	-42.6
	ParkScene	-27.8	-27.9
	Cactus	-30.9	-31.7
	BasketballDrive	-35.3	-41.1
	BQTerrace	-40.0	-48.0
	Avg_1080p	-34.7	-38.3
Class C WVGA	BasketballDrill	-30.7	-28.7
	BQMall	-32.7	-31.4
	PartyScene	-32.7	-26.3
	RaceHorses	-28.3	-27.4
	Avg_WVGA	-31.1	-28.5
Class D WQVGA	BasketballPass	-22.3	-23.0
	BQSquare	-44.0	-34.0
	BlowingBubbles	-26.7	-16.1
	RaceHorses	-20.5	-20.4
	Avg_WQVGA	-28.4	-23.4
Class E 720p	Vidyo1		-46.4
	Vidyo3		-41.4
	Vidyo4		-41.3
	Avg_720p		-43.0
	Overall Avg	**-30.9**	**-33.0**

B. *Subjective performance*

All the proposals that were submitted in response to the CfP underwent rigorous subjective evaluations. Table V compares the average mean opinion score (MOS) for the proposed codec against the corresponding AVC/H.264 anchors for different resolutions. From the table it can be seen that the average gain on MOS scale is in the range 1.1 – 2.6. The proposed codec scored highly in subjective evaluations and was among the best-performing CfP proposals.

TABLE V
COMPARISON OF THE SUBJECTIVE PERFORMANCE OF THE PROPOSED CODEC AGAINST ALPHA AND BETA ANCHORS FOR CS1 AND CS2, RESPECTIVELY.

Class	Average MOS			
	CS1		CS2	
	Proposed codec	Alpha anchor	Proposed codec	beta anchor
1080p	8.2	6.5	7.5	5.3
WVGA	6.2	4.5	5.5	3.9
WQVGA	6.4	5.3	5.4	4.1
720p			6.8	4.2

C. *Complexity comparison*

We have discussed the complexity of the different tools used in the proposed codec throughout the paper. Here we provide an estimate of the average encoding and decoding times for the proposed codec against JM 16.2.

The encodings were performed on a Linux cluster. The encoders for the proposed codec and JM 16.2 were compiled in 32 bits. The encoder software for the proposed codec was single-threaded with no assembly code. When averaged over different bit rates, sequences and constraint sets, the average encoding time for the proposed codec was roughly 3 times higher than that for JM 16.2.

The decoding was performed on a Windows PC running 32-bit Windows XP. The decoder software for the proposed codec was single-threaded with no assembly code and compiled using Microsoft® Visual Studio® 2005 Professional Edition. The I/O times (for output YUV file generation) were included in decoding times measurements. The average decoding time for the proposed codec was roughly 3 times higher than that for JM 16.2.

IV. CONCLUSION

In this paper, we have presented a video codec based on extended macroblock sizes, improved interpolation and flexible motion representation. This codec constituted Qualcomm's response to CfP for the next generation video coding standard. In this paper, the key features of the proposed codec have been described, including detailed algorithm descriptions. We have presented objective and subjective performance results for the proposed codec in comparison to the anchors generated by the AVC/H.264 reference software codec JM16.2. The proposed codec scored highly in both subjective evaluations and objective metrics and was among the best-performing CfP proposals.

REFERENCES

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

[2] "Joint Call for Proposals on Video Compression Technology", ITU-T Q6/16 document, VCEG-AM91, January 2010.

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

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

[5] Jens-Rainer Ohm and Gary Sullivan, "Special section on the Call for Proposals on High Efficiency Video Coding Standardization," IEEE Trans. Circuits Syst. Video Technol. this issue.

[6] Marta Karczewicz, Peisong Chen, Rajan Joshi, Xianglin Wang, Wei-Jung Chien, Rahul Panchal, "Video coding technology proposal by Qualcomm Inc" JCT-VC Contribution JCTVC-A121, Dresden, Germany, April 2010

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

[8] T. Chujoh, R. Noda "Internal bit depth increase for coding efficiency," ITU-T Q6/16 document, VCEG-AE13, Marrakech, Morocco, January 2007.

[9] T. Chujoh, N. Wada and G. Yasuda, "Quadtree-based Adaptive Loop Filter," ITU-T Q.6/SG16 document, C181, Geneva, January 2009.

[10] Y. Ye and M. Karczewicz, "Improved Intra Coding," ITU-T Q.6/SG16, Contribution C257, Geneva, Switzerland, June 2007.

[11] Y. Ye and M. Karczewicz, “Improved Intra Coding,” VCEG Contribution VCEG-AG11, Y. Ye and M. Karczewicz, Shenzhen, China, 20 October, 2007.

[12] Peisong Chen, Yan Ye, and Marta Karczewicz, “Video Coding Using Extended Block Sizes,” ITU-T Q.6/SG16 document, C123, Jan. 2009.

[13] 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.

[14] Ò. Divorra, P. Yin and C. Gomila "Geometry-adaptive Block Partitioning", ITU-T Q.6/SG16 document, VCEG-AF10, Geneva, Switzerland, April 2007.

[15] Alexis M. Tourapis, “Enhanced predictive zonal search for single and multiple frame motion estimation,” Proc. Visual Communications and Image Processing, pp. 1069 - 1079, 2002.

[16] Marta Karczewicz, Yan Ye, Peisong Chen, Giovanni Motta, “Single Pass Encoding using Switched Interpolation Filters with Offset,” -T Q6/16 document, VCEG-AJ29, San Diego, CA, USA, Oct. 2008.

[17] Tomoyuki Yamamoto, Yukinobu Yasugi, Tomohiro Ikai, “Further result on constraining transform candidate in Extended Block Sizes,” VCEG Contribution VCEG-AL19, London, UK / Geneva, CH, July 2009.

[18] R. Joshi, Y. Reznik, and M. Karczewicz, “Simplified Transforms for Extended Block Sizes,” VCEG Contribution VCEG-AL19, London, UK / Geneva, CH, July 2009.

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

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

[21] R. Joshi, Y. Reznik, and M. Karczewicz, “Efficient large size transforms for high performance video coding,” Proc. SPIE 7798, pp. 779831, 1-7, August 2010.

[22] M. Karczewicz, Y. Ye and I. Chong, Rate distortion optimized quantization, VCEG Contribution, VCEG-AH21, Jan. 2008

[23] Marta Karczewicz; Peisong Chen; Yan Ye; Rajan Joshi, “R-D based quantization in H.264,”, Applications of Digital Image Processing XXXII, Andrew G. Tescher, Editors, 744314, San Diego, June 2009.

[24] Wei-Jung Chien, Marta Karczewicz, “Adaptive Filter Based on Combination of Sum-Modified Laplacian Filter Indexing and Quadtree Partitioning,” VCEG Contribution VCEG-AL27r1, London, UK / Geneva, CH, July 2009.

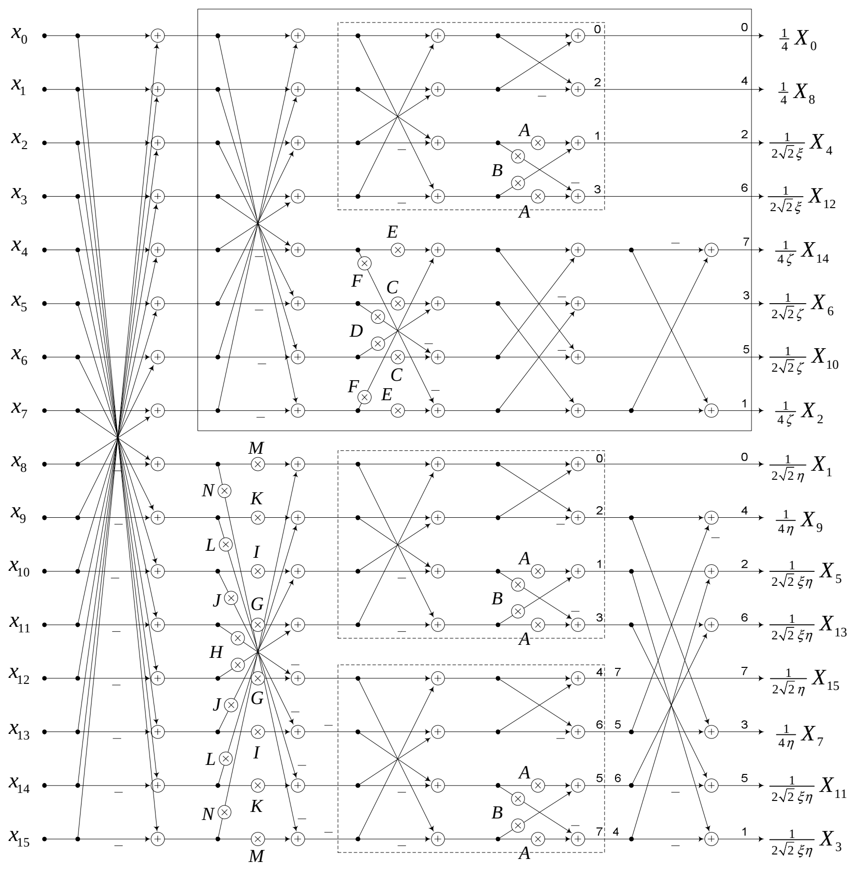
[25] S. K. Nayar and Y. Nakagawa, “Shape from focus,” IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 16, no. 8, pp. 824-831, 1994.

[26] Detlev Marpe, Heiko Schwarz, and Thomas Wiegand, “Context-Based Adaptive Binary Arithmetic Coding in the AVC/H.264 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. Bjøntegaard, “Calculation of average PSNR differences between RD-curves”, VCEG contribution VCEG-M33, Austin, USA, April 2001

[29] G. Bjøntegaard, "Improvements of the BD-PSNR model", VCEG contribution, VCEG-AI11, Berlin, Germany, July, 2008.



x_0
x_1
x_2
x_3
x_4
x_5
x_6
x_7
x_8
x_9
x_{10}
x_{11}
x_{12}
x_{13}
x_{14}
x_{15}
A
B
C
D
E
F
G
H
I
J
K
L
M
N
$\frac{1}{4}X_0$
$\frac{1}{4}X_8$
$\frac{1}{2\sqrt{2}\xi}X_4$
$\frac{1}{2\sqrt{2}\xi}X_{12}$
$\frac{1}{4\zeta}X_{14}$
$\frac{1}{2\sqrt{2}\zeta}X_6$
$\frac{1}{2\sqrt{2}\zeta}X_{10}$
$\frac{1}{4\zeta}X_2$
$\frac{1}{2\sqrt{2}\eta}X_1$
$\frac{1}{4\eta}X_9$
$\frac{1}{2\sqrt{2}\xi\eta}X_5$
$\frac{1}{2\sqrt{2}\xi\eta}X_{13}$
$\frac{1}{2\sqrt{2}\eta}X_{15}$
$\frac{1}{4\eta}X_7$
$\frac{1}{2\sqrt{2}\xi\eta}X_{11}$
$\frac{1}{2\sqrt{2}\xi\eta}X_3$

Fig. 6. Detailed flow graph for the proposed 1-D 16 point transform.