Low Complexity Integer Transform and High Definition Coding

Siwei Ma[a], Xiaopeng Fan[b], Wen Gao[a]

[a]Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China

[b]Harbin Institute of Technology, Harbin, China

E-mail: {swma, xpfan, wgao}@jdl.ac.cn

ABSTRACT

In H.264/AVC, an integer 4×4 transform is used instead of traditional float DCT transform due to its low complexity and exact reversibility. Combined with the normalization for the integerr transform together, a division-free quantization scheme is used in H.264/AVC. H.264/AVC is the most outstanding video coding standard at far. But at first H.264/AVC targets to low bit-rate coding, and almost all experimental results of the proposals for H.264/AVC are tested at low bit-rate. In the near, experimental results show that 8×8 transform can further improve the coding efficiency on high definition (HD) coding. In this paper a kind of 8×8 low complexity integer transforms are studied and corresponding quantization schemes are developed for HD coding. Compared with 4×4 transform/prediction based coder, the proposed 8×8 based coder can achieve better performance on HD coding while with much lower encoder/decoder complexity.

Keywords: DCT, BinDCT, IntDCT, integer transform, quantization

1. INTRODUCTION

Block-based transform coding is an important video coding technique and it has been widely used in many international video coding standards, such as MPEG-1/2 and H.261/2/3. Spatial redundancy is attenuated as the block of pixels are converted into uncorrelated coefficients through the orthogonal transform. After transform the block energy can be denoted by a few transform coefficients and compression can be achieved by following quantization and entropy coding. From energy compaction viewpoint, KLT (Karhunen-Loeve transform) is the best transform. However, it is difficult to use KLT in image and video coding because it is signal-dependent. DCT is a better approximation of KLT and it is easy for implementation due to its low complexity. But float point multiplication in DCT is too complex and can not map integer to integer losslessly due to float approximation. In the past, many researches have been done to integer-friendly approximation of the float DCT, such as binDCT[1,2] and IntDCT[3], where the float DCT coefficient is approximated as an integer coefficient multiplier and a right shift. So the DCT transform can be implemented only by using shifts and adds.

In the development of H.264, many proposals on integer cosine transform (ICT, or integer transform: IT) have been talked about, such as 16x16 ICT[4], 4×4 IT[5, 6] and adaptive block transform (ABT)[7]. In the last, a low complexity 4×4 integer transform[5, 6] is accepted. The integer transform has much virtue, such as low complexity, non-mismatch existing in float DCT transform etc. But, at first H.264 mainly targets to low bit rate coding and almost all test results are provided on QCIF/CIF resolution sequences. For low resolution coding, the variable block size motion prediction to 4×4 and 4×4 transform can achieve higher coding efficiency. But that would lead to higher encoder complexity especially for HD coding. Experimental results show that 8×8 based transform/prediction coder can achieve better performance on HD coding while with much lower complexity. In the current development of H.264, 8×8 transform[9, 10] has been proposed again for professional extension coding (PExt). This paper makes a study on the integer transform in H.264/AVC and a kind of 8×8 integer transforms and corresponding quantization scheme are developed on HD coding.

The rest of this paper is organized as follows. A study on the integer transform and corresponding quantization scheme in H.264/AVC is detailed in Section 2. Based on the study a kind of 8×8 integer transforms and corresponding quantization schemes on HD coding are discussed in Section 3. Experimental results are provided in Section 4. Section 5 concludes the paper.

2. Integer Transform and Quantization in H.264

According to DCT definition, a typical 4×4 DCT-like integer transform can be expressed as:

$$\begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} \quad (1)$$

b/c usually is between 1.25~2.5. A 4×4 transform[11] with a=13, b=17 and c=7 is proposed for H.264. In the last, a low complexity 4×4 integer transform is used in H.264 with a=1, b=2, and c=1. The forward transform is shown as follows:

$$Y = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} & & & \\ & X & & \\ & & & \\ & & & \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \quad (2)$$

As the integer transform is not a unitary matrix, Y must be normalized after transform, as follows:

$$W = \begin{bmatrix} & & \\ & Y & \\ & & \end{bmatrix} \otimes \begin{bmatrix} & & \\ & E & \\ & & \end{bmatrix} = \begin{bmatrix} & & \\ & Y & \\ & & \end{bmatrix} \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \quad (3)$$

, here $a = 1/2, b = 1/\sqrt{10}$.

In H.264/AVC the transform normalization is combined with quantization and the division is replaced with multiplication and right shift. For a quantization parameter QP, the quantization is implemented as follows:

$$LEVEL_{i,j} = round(W_{i,j}/Q_{step}) = round\,(Y_{i,j} \times scaler_{i,j}/2^{qbits}), \quad (4)$$

Where $scaler_{i,j}/qbits = E_{i,j}/Q_{step}$ and $qbits = 15 + floor(QP/6)$ and $Q_{step} = 2^{((QP-4)/6)}$.

The dequantization and transform normalization on decoder is combined together in the same way as on the encoder as follows:

$$COEF_{i,j} = round(LEVEL_{i,j} \times Q_{step} \times E_{i,j}) = round(LEVEL_{i,j} \times de\_scaler_{i,j}/2^{dqbits}) \quad (5)$$

Here, $dqbits$= 6-QP/6, and $scaler_{i,j} \times de\_scaler_{i,j} \times M_{i,j} = 2^{qbit+dqbitss}$. In H.264/AVC, $scaler_{i,j}$=A(QP%6,r), $de\_scale_{i,j}$=B(QP%6,r) A, B is 6x3 scale matrix, shown as follows:

$$A = \begin{bmatrix} 13107 & 5243 & 8066 \\ 11916 & 4660 & 7490 \\ 10082 & 4194 & 6554 \\ 9362 & 3647 & 5825 \\ 8192 & 3355 & 5243 \\ 7282 & 2893 & 4559 \end{bmatrix} \qquad B = \begin{bmatrix} 10 & 16 & 13 \\ 11 & 18 & 14 \\ 13 & 20 & 16 \\ 14 & 23 & 18 \\ 16 & 25 & 20 \\ 18 & 29 & 23 \end{bmatrix}$$

$M_{i,j}$ is the magnifier from transform. r=0 and $M_{i,j}$ = 16 for (i,j)={(0,0),(0,2),(2,0),(2,2)}; r=1 and $M_{i,j}$ = 100 for (i,j)={(1,1),(1,3),(3,1),(3,3)}; r=2 and $M_{i,j}$ = 40 for others.

3. Low Complexity 8×8 Integer Transform

Based on the study for the 4×4 integer transform, we propose a kind of low complexity 8×8 integer transform for HD coding. In general, an 8×8 transform can be expressed in following format [7, 8]:

$$\begin{bmatrix} a & a & a & a & a & a & a & a \\ b & c & d & e & -e & -d & -c & -b \\ f & g & -g & -f & -f & -g & g & f \\ d & e & -b & -c & c & b & -e & -d \\ a & -a & -a & a & a & -a & -a & a \\ c & -b & -e & d & -d & e & b & -c \\ g & -f & f & -g & -g & f & -g & g \\ e & -d & c & -b & b & -c & d & -e \end{bmatrix} \tag{7}$$

The 8×8 transform[7] with a=13, b=19, c=15, d=9, e=3, f=17, g=7, is named as T8×8-1 in Figure-1. Based on (5), this paper proposes a kind of 8×8 integer transform for HD coding. The proposed 8×8 transform is denoted as b:c:d:e = 5:4:3:1. Two examples are shown in Figure 1: the first is T8×8-2 with a=1,b=5,c=4,d=3,e=1,f=2,g=1; the second is T8×8-3 with a=7, b=10, c=8, d=6, e=2, f=9, g=4.

$$\begin{bmatrix} 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 \\ 19 & 15 & 9 & 3 & -3 & -9 & -15 & -19 \\ 17 & 7 & -7 & -17 & -17 & -7 & 7 & 17 \\ 9 & 3 & -19 & -15 & 15 & 19 & -3 & -9 \\ 13 & -13 & -13 & 13 & 13 & -13 & -13 & 13 \\ 15 & -19 & -3 & 9 & -9 & 3 & 19 & -15 \\ 7 & -17 & 17 & -7 & -7 & 17 & -17 & 7 \\ 3 & -9 & 15 & -19 & 19 & -15 & 9 & -3 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 5 & 4 & 3 & 1 & -1 & -3 & -4 & -5 \\ 2 & 1 & -1 & -2 & -2 & -1 & 1 & 2 \\ 3 & 1 & -5 & -4 & 4 & 5 & -1 & -3 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 4 & -5 & -1 & 3 & -3 & 1 & 5 & -4 \\ 1 & -2 & 2 & -1 & -1 & 2 & -2 & 1 \\ 1 & -3 & 4 & -5 & 5 & -4 & 3 & -1 \end{bmatrix} \quad \begin{bmatrix} 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 \\ 10 & 8 & 6 & 2 & -2 & -6 & -8 & -10 \\ 9 & 4 & -4 & -9 & -9 & -4 & 4 & 9 \\ 6 & 2 & -10 & -8 & 8 & 10 & -2 & -6 \\ 7 & -7 & -7 & 7 & 7 & -7 & -7 & 7 \\ 8 & -10 & -2 & 6 & -6 & 2 & 10 & -8 \\ 4 & -9 & 9 & -4 & -4 & 9 & -9 & 4 \\ 2 & -6 & 8 & -10 & 10 & -8 & 6 & -2 \end{bmatrix}$$

T8×8-1 T8×8-2 T8×8-3

Figure 1. Integer 8×8 transform

Three kinds of measurements[4] can be used to evaluate the performance of integer transform: transform coding gain, distortion from DCT and frequency distortion.

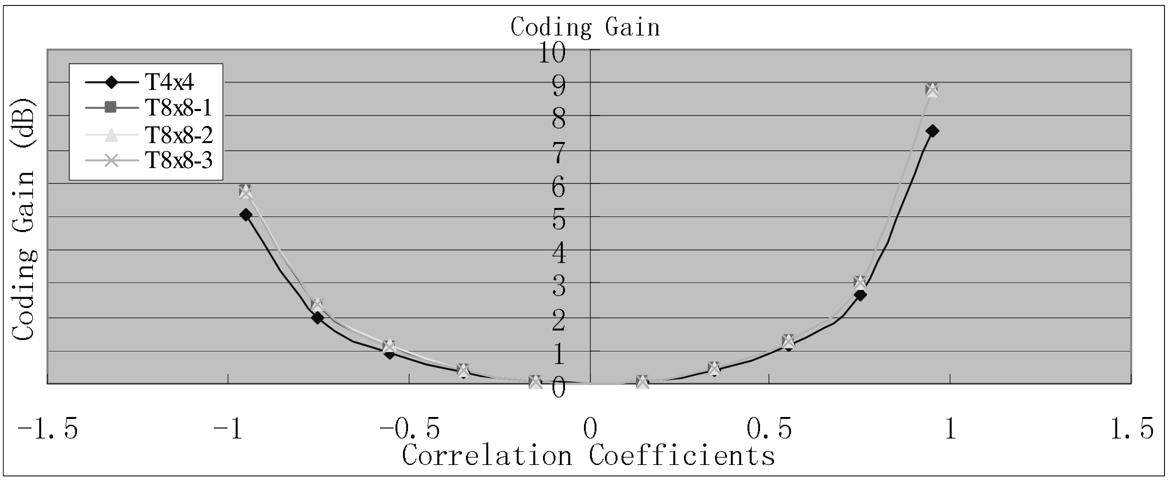

Figure 2. Coding gain for each ICT

For the coding gain, under the assumption of optimum quantization and optimum bit allocation, the coding gain of an orthonormal block transform can be computed as follows:

$$G_N = 10 \cdot \log_{10} \frac{\frac{1}{N}\sum_{n=0}^{N-1}\sigma_n^2}{\left(\prod_{n=0}^{N-1}\sigma_n^2\right)^{1/N}} \tag{8}$$

where σ_n^2 is the variance of the n-th transformed coefficient. In general the input signal can be modeled by an AR(1) process, and the AR(1) process is characterized by the correlation coefficient ρ. The correlation coefficient ρ is in the range [-0.95, 0.95]. Figure 2 depicts the coding gain in the same range. Table 1 lists the coding gain of T4×4 (the 4×4 integer transform in H.264), and other three 8×8 transforms.

Table 1. Coding gain for each ICT

ρ	T4×4 (dB)	T8×8-1 (dB)	T8×8-2 (dB)	T8×8-3 (dB)
-0.95	5.0627	5.7618	5.7512	5.6904
-0.75	1.9692	2.3223	2.3926	2.3600
-0.55	0.9314	1.1071	1.1368	1.1244
-0.35	0.3583	0.4219	0.4309	0.4278
-0.15	0.0625	0.0756	0.0769	0.0766
0.15	0.0685	0.0777	0.0788	0.0788
0.35	0.4039	0.4547	0.4603	0.4607
0.55	1.1370	1.2833	1.2951	1.2974
0.75	2.6517	3.0264	3.0414	3.0471
0.95	7.5541	8.7589	8.7639	8.7730

Except for the coding gain, a transform also can be evaluated by its approximation to the DCT transform. The squared distortion of the ICT matrices with respect to the DCT is defined as:

$$d_2 = 1 - \frac{1}{N}\left\| diag\left(T_{DCT} \cdot T^A\right)\right\|_2^2 \tag{9}$$

where $diag(X)$ denotes the main diagonal of matrix X. T_{DCT} is N×N DCT. T is the normalized ICT matrix. The distortion reflects the signal energy that is deferred from the individual DCT subbands. In Table 2 the distortion for each basis vector and the overall d_2 distortion are given for T4×4, T8×8-1, T8×8-2, and T8×8-3.

Another distortion measure is frequency distortion measure. The first-order frequency distortion d'_1 and the second-order frequency distortion d'_2 are defined as:

$$d'_1 = \frac{1}{N}\sum_{i=0}^{N}\sum_{\substack{j=0 \\ j\neq i}}^{N}\frac{|T_{ji}|}{|T_{ii}|} \tag{10}$$

and

$$d'_2 = \frac{1}{N}\sum_{i=0}^{N}\sum_{\substack{j=0 \\ j\neq i}}^{N}\frac{|T_{ji}|^2}{|T_{ii}|^2} \tag{11}$$

where T represents the normalized ICT matrices. Table 3 shows the frequency distortion for the four kinds of transform.

Table 2. Distortion of the DCT basis vectors for each ICT

	T8×8-1	T8×8-2	T8×8-3
$d_{2,0}$	0.0000	0.0000	0.0000
$d_{2,1}$	0.0042	0.0016	0.0016
$d_{2,2}$	0.0000	0.005	0.0007
$\mathrm{d}_{2,3}$	0.1517	0.129	0.129
$d_{2,4}$	0.0000	0.0000	0.0000
$d_{2,5}$	0.1517	0.129	0.129
$d_{2,6}$	0.0000	0.005	0.0007
$d_{2,7}$	0.0042	0.0016	0.0016
d_2	0.038975	0.0339	0.032825

Table 3. The first and second-order frequency distortions of each ICT

	T4×4	T8×8-1	T8×8-2	T8×8-3
d'_1	0.0355	0.1451	0.1411	0.0391
d'_2	0.0025	0.0458	0.0387	0.0017

For T8×8-1, all rows have the same module, and the following quantization scale matrixes A_1 and dequantization scale matrixes B_1 are used on encoder and decoder set in the proposal[7] for H.264/AVC. Here, $A_1[\mathrm{i}]\times B_1[\mathrm{i}]\times M\approx 2^{36}$ (i=0..5). M is the magnifier from transform, and M = 1352^2. $scaler_{i,j}$ and $de\_scale_{i,j}$ are defined as: $scaler_{i,j}=A_1(QP\%6)$, $de\_scale_{i,j}=B_1(QP\%6)$ (i, j=0..5).

$$
\mathrm{A}_1 = \begin{bmatrix} 2506 \\ 2211 \\ 1979 \\ 1709 \\ 1566 \\ 1392 \end{bmatrix} \qquad B_1 = \begin{bmatrix} 15 \\ 17 \\ 19 \\ 22 \\ 24 \\ 27 \end{bmatrix}
$$

T8×8-2 has the same feature as the 4×4 transform used in H.264 that is not all rows have the same norm. In our implementation the following scale matrixes A_2 and B_2 are used at encoder/decoder for quantization and dequantization, and $A_2[i, j]\times B_2[i, j]\times M_{i,j} = 2^{28}$. $M_{i,j}$ is the multiplier from transform: $scaler_{i,j}$ and $de\_scale_{i,j}$ are defined as: $scaler_{i,j}=A_1(QP\%6, r_{i,j})$, $de\_scale_{i,j}=B_1(QP\%6, r_{i,j})$ (i, j=0..5). r is mapped as follows:

$$
\mathrm{A}_2 = \begin{bmatrix} 21960 & 1720 & 8715 & 6092 & 13865 & 3870 \\ 19329 & 1518 & 7714 & 5393 & 12246 & 3463 \\ 17332 & 1358 & 6918 & 4838 & 10966 & 3060 \\ 14926 & 1173 & 5992 & 4164 & 9479 & 2632 \\ 13707 & 1075 & 5501 & 3825 & 8648 & 2437 \\ 12193 & 956 & 4863 & 3427 & 7696 & 2157 \end{bmatrix} \quad B_2 = \begin{bmatrix} 191 & 15 & 77 & 54 & 121 & 34 \\ 217 & 17 & 87 & 61 & 137 & 38 \\ 242 & 19 & 97 & 68 & 153 & 43 \\ 281 & 22 & 22 & 79 & 177 & 50 \\ 306 & 24 & 24 & 86 & 194 & 54 \\ 344 & 27 & 27 & 96 & 218 & 61 \end{bmatrix} \quad r = \begin{bmatrix} 0 & 3 & 4 & 3 & 0 & 3 & 4 & 3 \\ 3 & 1 & 5 & 1 & 3 & 1 & 5 & 1 \\ 4 & 5 & 2 & 5 & 4 & 5 & 2 & 5 \\ 3 & 1 & 5 & 1 & 3 & 1 & 5 & 1 \\ 0 & 3 & 4 & 3 & 0 & 3 & 4 & 3 \\ 3 & 1 & 5 & 1 & 3 & 1 & 5 & 1 \\ 4 & 5 & 2 & 5 & 4 & 5 & 2 & 5 \\ 3 & 1 & 5 & 1 & 3 & 1 & 5 & 1 \end{bmatrix}
$$

For T8×8-3, the module of every row is approximated and the normalization can be implemented on the encoder. On the encoder set, a 6×6 scale matrix A_3 can be used to reach normalization and quantization, but on the decoder set only a 6×1 scale matrix B_3 is used instead that would reduce hardware memory requirement.

$$
\mathrm{A}_3 = \begin{bmatrix} 26306 & 24283 & 26851 & 25274 & 26577 & 25535 \\ 23436 & 21634 & 23922 & 22517 & 23677 & 22749 \\ 20879 & 19273 & 21312 & 20060 & 21094 & 20267 \\ 18601 & 17171 & 18987 & 17871 & 18793 & 18056 \\ 16572 & 15297 & 16915 & 15922 & 16742 & 16086 \\ 14764 & 13628 & 15070 & 14185 & 14916 & 14331 \end{bmatrix} \qquad B_3 = \begin{bmatrix} 17 \\ 19 \\ 21 \\ 24 \\ 27 \\ 30 \end{bmatrix}
$$

For software implementation, T8×8-2 can be implemented by using 6 shifts and 32 adds through butterfly decomposition. T8×8-3 can be implemented by using 36 adds and 10 shifts. Figure 3 shows the butterfly decomposition for T8×8-2.

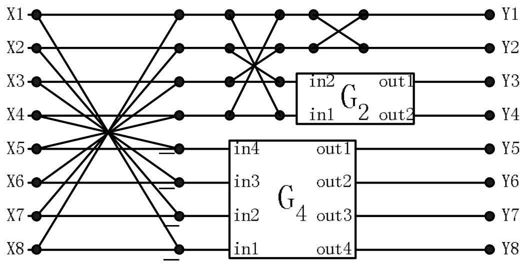

Figure 3. Butterfly decomposition of T8×8-2

Here,

$$G_2 = \begin{bmatrix} 1 & 2 \\ -2 & 1 \end{bmatrix} \qquad G_4 = \begin{bmatrix} 5 & 4 & 3 & 1 \\ 3 & 1 & -5 & -4 \\ 4 & -5 & -1 & 3 \\ 1 & -3 & 4 & -5 \end{bmatrix}$$

and G_4 can be further decomposed as shown in Figure 4:

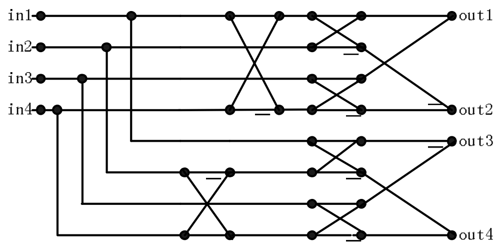

Figure 4. Butterfly decomposition of G_4

4. Experimental Results

To test the performance of proposed 8×8 transform, the JVT reference software-JM5.0c is selected as test platform, in which an ABT scheme is implemented.

First the performance comparison between 4×4 transforms based coder and 8×8 transform coder is studied. The test conditions are listed in Table 4. Table 5 shows the complexity ration between 8×8 transform and 4×4 transform platform. Figure 5 shows the PSNR curve of two coders. From Table 5 and the curve, we can see that 8×8 transform based coder can reach similar or even better performance compared with 4×4 based coder. But the encoder and decoder complexity of 8×8 coder is reduced heavily, because variable block size is restricted to vary from 16x16 to 8×8.

Figure 6 show the performance for different integer transform, including the 4×4 transform used in H.264 and two proposed 8×8 transforms. From the curve, we can see the proposed 8×8 transform can achieve similar or even better performance.

Table 4, Test conditions for 8×8/4×4 transform/prediction coder based on JM5.0c

	8×8 coder	4×4 coder
Transform	T8×8-1	H.264 4×4
Entropy Coding	CABAC	CABAC
Variable Block Size MC	16x16, 16x8, 8x16, 8×8	16x16, 16x8, 8x16, 8×8, 8x4, 4x8, 4×4
RDO	On	On
Multiple Reference	2 reference frames	2 reference frames
Search Range	±64	±64

Table 5, Complexity comparison between 8×8/4×4 transform platform

	Harbour (8×8/4×4)	Night (8×8/4×4)
Time-Ratio (encoder)	67.68%	67.68%
Time-Ratio (decoder)	89.31%	96.51%

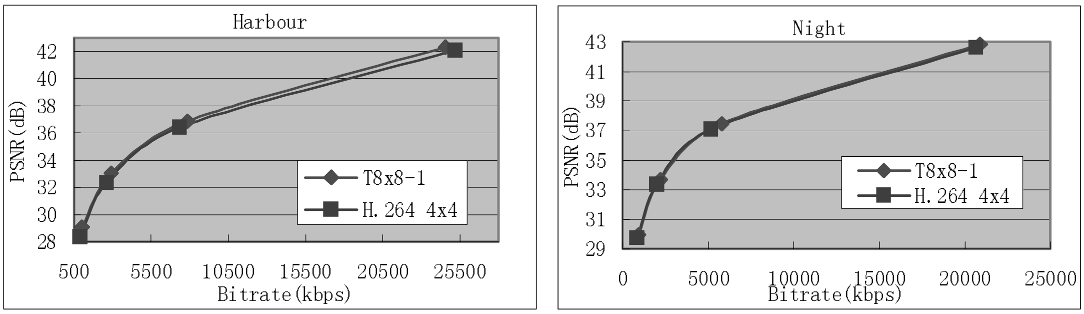

Figure 5. PSNR performance comparison between 8×8/4×4 transform platform

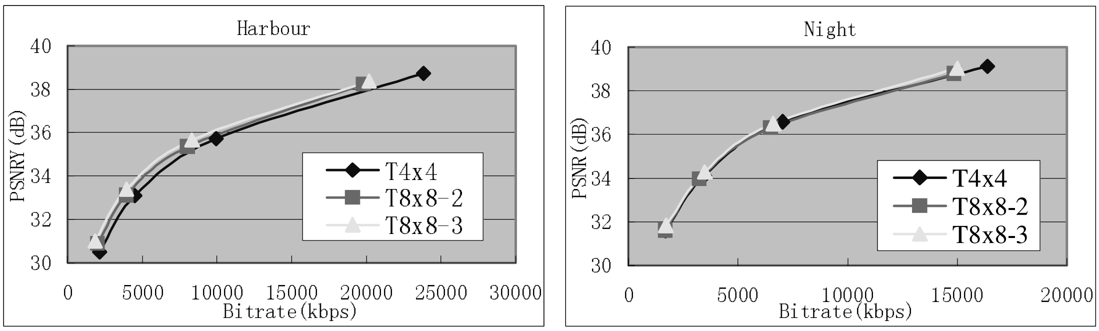

Figure 6. PSNR performance comparison between proposed ICT

5. Conclusion

In this paper, low complexity integer transform and its corresponding quantization scheme used in H.264 have been studied, and based on this research, a kind of low complexity 8×8 integer transforms and corresponding quantization scheme are implemented for HD coding. The proposed 8×8 transform based coder can achieve similar or even better performance compared with 4×4 transform based coder. But the encoder/decoder complexity of proposed 8×8 transform based coder is much lower than the 4×4 coder.

6. Reference

1. Trac D. Tran, “The BinDCT: Fast Multiplierless Approximation of the DCT,” IEEE Signal Processing Letters, Vol. 7, No.6 2000. pp. 141-144.
2. Ying-jui Chen, Soontorn Oraintara, Trac D. Tran, Kevin Amaratunga, Truong Q. Nguyen, “Multiplierless Approximation of Transforms with Adder Constraint,” IEEE Signal Processing Letters, Vol. 9, No. 11, November 2002. pp. 344-347.
3. Ying-jui Chen, Soontorn Oraintara, Truong Q. Nguyen, “Video Compression Using Integer DCT,” ICIP 2000.
4. Mathias Wien, Shijun Sun, “ICT Comparison for Adaptive Block Transform,” VCEG-L12. 2003, Jan 01.
5. A. Hallapuro, M. Karczewicz, and H. Malvar, “Low Complexity Transform and Quantization – Part I: Basic Implementation,” ISO/IEC JTC1/SC29/WG11 and ITU-T SG16 Q.6 Document JVT-B038, January 2002.
6. A. Hallapuro, M. Karczewicz, and H. Malvar, “Low Complexity Transform and Quantization – Part II: Extensions,” ISO/IEC JTC1/SC29/WG11 and ITU-T SG16 Q.6 Document JVT-B039, January 2002.
7. M. Wien, “Clean-up and improved design consistency for ABT”, Doc. JVT-E025.
8. J.A. Michell, G. A. Ruiz, A. M. Burón, “Parallel-pipelined Architecture for 2-D ICT VLSI Implementation,” ICIP 2003.
9. S. Gordon, D. Marpe, T. Wiegand, “Simplified Use of 8×8 Transforms – Proposal”, Doc. JVT-J029, Waikaloa, Dec. 2003.
10. S. Gordon, D. Marpe, T. Wiegand, “Simplified Use of 8×8 Transforms – Updated Proposal & Results”, Doc JVT-K028, 11th Meeting: Munich, Germany, 15-19 March, 2004.
11. Gisle Bjontegaard, “Coding improvement by using 4×4 blocks for motion vectors and transform”, Doc Q15-C-23, Eibsee, Bavaria, Germany, 2-5 December, 1997.