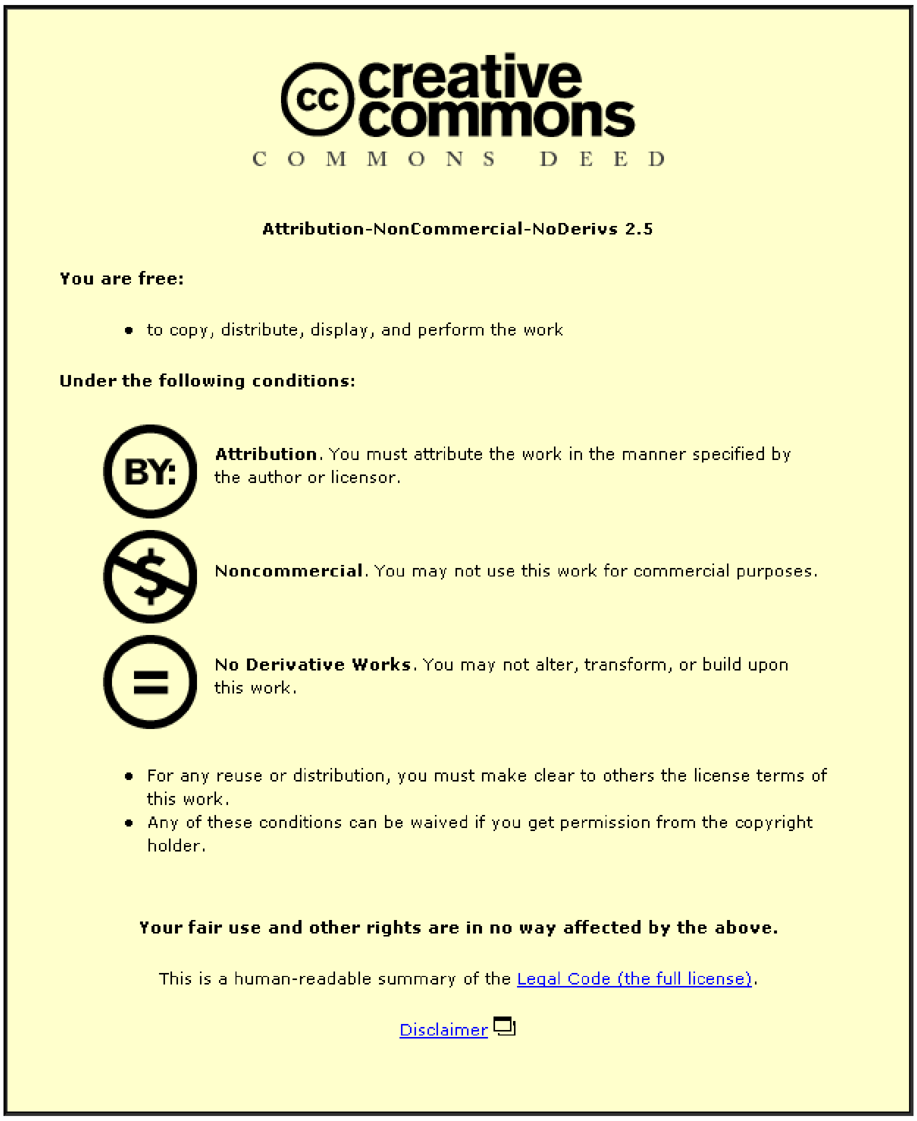


Shape adaptive integer transform for coding arbitrarily shaped objects in H.264/AVC

Xiongwen Li, Eran Edirisinghe, Helmut Bez
Dept. of Computer Science, Loughborough University, Loughborough, UK LE11 3TU

ABSTRACT

The use of shape-adaptive transforms is a popular approach for coding arbitrarily shaped objects in image/video coding due to their adaptability at object edges and low complexity. In this respect shape adaptive DCT (SA-DCT) and shape adaptive DWT (SA-DWT) have been proposed in previous literature. The Integer Transform (IT), a derivative of the 4x4 DCT, has been adopted in the latest H.264/AVC standard for coding image blocks in residual data (texture). The associated integer arithmetic guarantees fast and accurate coding/decoding. In this paper, we propose a novel Shape Adaptive Integer Transform (SA-IT) which can be effectively used in future for enabling arbitrary shaped object coding in H.264. Though Integer Transforms are a derivative of 4x4 DCTs, in H.264, to maintain integer arithmetic capability, the post-and pre-scaling factors of transform process are integrated into the forward and inverse quantiser stages respectively for reducing the total number of multiplications and avoiding the loss of accuracy. Thus SA-IT considerably differs from SA-DCT and calls for novel design and implementation considerations based on combining those merits of both SA-DCT and IT. We provide theoretical proofs and support them with experimental justifications.

Keywords: H.264, video coding, shape adaptive, integer transform, shape adaptive integer transform, SA-IT, SA-DCT, Arbitrarily shaped object, DCT

1. INTRODUCTION

H.264/AVC is the latest video coding standard that was developed jointly by the Moving Picture Experts Group (MPEG) and the Video Coding Experts Group (VCEG). It has published both as Part 10 of MPEG-4 and ITU-T Recommendation, H.264[1, 2, 3]. Its original aim was to provide similar functionality to earlier standards such as H.263+ and MPEG-4 Visual (Simple Profile) but with significantly better compression performance and improved support for reliable transmission. One noteworthy functionality absence in H.264 is the capability of coding arbitrary shaped objects. Provided, means exists for coding and transmitting shape information (e.g., the use of the auxiliary stream), one would additionally require SA-IT to fully enable the above functionality.

Unlike the traditional 8x8 DCT[6] used in previous standards, the 4x4 IT in H.264.AVC is carried out using integer arithmetic. Thus not only it avoids mismatches in the inverse transform, but also significantly minimizes computational complexity due to the exclusive use of simple addition and shift operations within 16-bit arithmetic. However, the present form of IT used within H.264/AVC has a limitation in encoding arbitrarily shaped objects as pixels outside the object would also be considered within the transformations. This would result in a waste of compute power and processing time.

In 1995, Sikora and Makai proposed a shape-adaptive DCT algorithm (SA-DCT)[4] to replace the popular 8x8 block based DCT coding, in arbitrary shaped object coding associated with MPEG-4 Visual Texture Coding standard. The SA-DCT is based on pre-defined sets of one-dimensional DCT basis functions. It allows an arbitrary region of a block to be efficiently transformed and compressed[3]. Two most important benefits of SA-DCT are its capability to adapt to the arbitrary shapes at video object boundaries and its low complexity. In[7], the authors solved some outstanding issues related to conventional SA-DCT proposed in[4] by orthonormalizing the basis functions of the SA-DCT. Nevertheless, all such previous work adopted a floating point arithmetic design and implementation, which is not suitable to be used in conjunction with IT used in H.264/AVC.

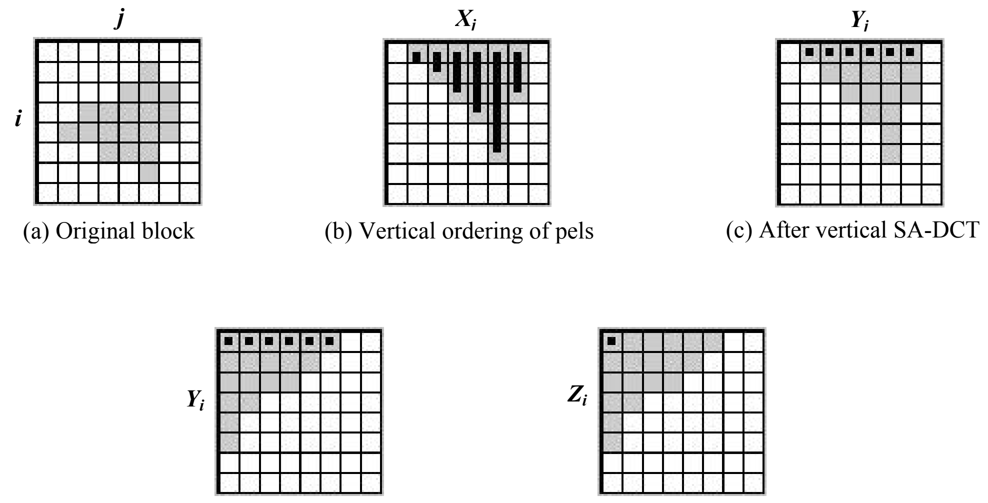
As regards to the limitation of coding arbitrarily shaped objects in H.264/AVC, it would be a significant advantage if specific regions of a frame, rather than the complete frame can be represented and coded at high quality as compared to regions of non-interest. To resolve the above mentioned restriction, it is the objective of this paper to propose a new method for coding arbitrarily shaped objects in H.264/AVC. This new approach will be referred to as SA-IT within the context of this paper. In our present design, we separate the core part of the integer transform and incorporate post-and pre-scaling factors into the forward and inverse quantiser stages respectively. For clarity of presentation, in section 2 we first provide a brief review on the SA-DCT algorithm proposed in[4, 7]. Section-3 details the IT used within H.264/AVC while section 4 describes the proposed SA-IT scheme. Section 5 presents some experiments that were designed to test the capability of the proposed ideas. Finally section 6 concludes with an insight into future directions of research.

2. SHAPE-ADAPTIVE DCT

The basic idea of the SA-DCT[4] is to transform an arbitrarily shaped image object by cascading column and row transforms of a given 8x8 block. Fig. 1 gives an example of the application of SA-DCT algorithm on an 8x8 image block that fully encloses an arbitrarily shaped object. Fig. 1(a) illustrates the division of the pixels within the said block into two groups, namely; foreground (shaded grey) and background (white). The foreground pixels are to be encoded with SA-DCT by first applying a 1-D DCT vertically and then applying a 1D transform horizontally on the resulting vertically transformed foreground object. This is done as follows: firstly, the length *N(j)* (*1* £ *N(j)* £ *8*) of every column *j* (*1* £ *j* £ *8*) of the foreground pixels are calculated. Then, each column is shifted up and finally aligned with the upper border of the block (see Fig. 1(b)). For a column of foreground pixels of length *N(j)*, the associated DCT transform matrix $A_{N(j)}$ is given by:

$$A_{N(j)}(p,k) = c_0 \cdot \cos\left[\frac{(2k+1)\cdot p \cdot \pi}{2N(j)}\right] \quad p,k = 0,1,\ldots,N(j)-1 \tag{1}$$

Here $c_0 = \sqrt{1/2}$ if $p = 0$, and $c_0 = 1$ otherwise.



(a) Original block (b) Vertical ordering of pels (c) After vertical SA-DCT

(d) Horizontal ordering of pels (e) SA-DCT coefficients

Figure 1: Example of forward SA-DCT processing in an arbitrarily shape image block

$Y_{j,}$, the vertical DCT-coefficients of column *j* resulting from the foreground pixels, X_j , can thus be obtained by using the following formula[7]:

$$Y_j = \sqrt{\frac{2}{N(j)}} \cdot A_{N(j)} \cdot X_j \quad , \quad 1 \le j \le 8, \quad 1 \le N(j) \le 8 \tag{2}$$

After vertical 1-D DCT transformation, the DC coefficients (denoted by ■ mark in fig.1(c)) for each column are found along the upper edge of the 8x8 block. Next, the rows are shifted to align at the left border of the 8x8 block (see fig.1(d)) and a horizontal (i.e. on individual rows) 1-D DCT transform is performed for each row of coefficients Y_i using the equations, (1) and (2). Finally, the result of DCT coefficients within the 8x8 boundary block is shown in Fig. 1(e). Note that the final DC coefficient (denoted by ■) for the whole boundary block is located in the upper left border of the block. The remaining coefficients are concentrated around the DC coefficient depending on the actual shape of the arbitrarily shaped object. The Inverse Shape-Adaptive DCT can be obtained with the use of the following equation:

$$X_j^* = \sqrt{2/N} \cdot A_{N(j)}^T \cdot Y_j^* \tag{3}$$

in both horizontal and vertical directions. Here Y_j^* denotes transformed coefficients, X_j^* denotes the inverse-transformed data and j refers to the length of the data set.

3. INTEGER TRANSFORM IN H.264

Similar to previous video coding standards, H.264/AVC employs transform coding of the prediction residual. Nevertheless, the transformation in H.264/AVC is applied to 4x4 blocks, and instead of the popular DCT, a separable integer transform (IT) with similar properties as a 4x4 DCT is used[2, 5]. There are several key features associated with the application of IT in H.264. Firstly, all operations can be carried out using integer arithmetic implemented as simple addition and shift operations. This results in a speed up of operation and accuracy in decoding. In addition to the above, the non integer calculations that are an inherent part of the transform operation are factored out and integrated into the quantization stage. This can be explained below.

The 2D DCT transform of a 4x4 matrix X can be written as:

$Y = AXA^T$, where $A=\begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix}$, where A is the 4x4 2D-DCT transformation matrix, $a=\frac{1}{2}$, $b=\sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right)$ and $c=\sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right)$.

Further factorizing and assuming $\frac{c}{b} = 0.5$ and $b = \sqrt{2/5}$ (In order to remain the orthogonal transform), we can rearrange the above equation as:

$$Y = \left(C_f X C_f^T\right) \otimes E_f = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} & & & \\ & & X & \\ & & & \\ & & & \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix}\right) \otimes \begin{bmatrix} a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \\ a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \end{bmatrix} \tag{4}$$

where: $a = 1/2$, $b = \sqrt{2/5}$.

In equation 4, $C_f X C_f^T$ is a 'core' 2D transform where C_f and C_f^T are the integer transform matrix and its transpose separately. X is the 4x4 pixel block data, E_f is a matrix of scaling factors (i.e. a post-scaling matrix) that resulted from the factorization discussed above and the symbol $\otimes$ indicates that each element of $C_f X C_f^T$ is multiplied by the scaling factor in the corresponding position in the matrix E_f. The post-scaling matrix E_f can be incorporated into the forward quantiser, thus guaranteeing that all calculations related the 'core' transform are integer. This integer transform is an approximation to the conventional 4x4 DCT but because of the change to factors b and c/b, the result of the IT will not be identical to the 4x4 DCT. Similarly, the inverse transform is given by the following equation:

$$X = C_i^T \left(Y \otimes E_i\right) C_i = \begin{bmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{bmatrix} \cdot \left(\begin{bmatrix} & & & \\ & Y & & \\ & & & \\ & & & \end{bmatrix} \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix}\right) \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1/2 & -1/2 & -1 \\ 1 & -1 & -1 & 1 \\ 1/2 & -1 & 1 & -1/2 \end{bmatrix} \tag{5}$$

In the above formula, the Y (the forward transformed coefficients) is pre-scaled by multiplying each coefficient by the appropriate weight factor from matrix E_i. The factors $\pm 1/2$ in the matrices C_i and C_i^T can be implemented by a simple right-shift operation without a significant loss of accuracy since the coefficients Y are pre-scaled. The forward and inverse transforms are orthogonal, i.e. $T^{-1}\left(T\left(X\right)\right)=X$. Both E_f and E_i are integrated into the forward and inverse quantisers separately in H.264. More details on the specific aspects of the transform and quatisation in H.264 can be found in[5].

4. SHAPE-ADAPTIVE INTEGER TRANSFORM (SA-IT)

In this section, we provide the theoretical concepts and mathematical proofs associated with the proposed SA-IT algorithm. The approach used is similar to that described for SA-DCT in section 2. However due to the requirement of having to maintain all calculations within the core transform (see section 3) as integer arithmetic, some important additional design considerations and matrix factorizations are utilized within the current context.

4.1 Forward transform

Vertical:

In equation (2), if $B_{N(j)}=\sqrt{2/N(j)}\cdot A_{N(j)}$, $\Rightarrow\ Y_j=B_{N(j)}\cdot X_j$ (6)

Note that $B_{N(j)}$ is a $N(j)$x$N(j)$ matrix and X_j , Y_j respectively represent vectors representing the column 'j' of the original 4x4 pixel block and the corresponding column's transform. Depending on the arbitrary shape of the object, the length of the above two vectors, $N(j)$, can range from 1-4. Thus depending on the length of $N(j)$, $B_{N(j)}$ can be factorised to $C_{N(j)}\otimes E_{N(j)}$ where $C_{N(j)}$ consists of integers and $E_{N(j)}$ consists of fractional elements :

When,

$$N(j)=1,\quad \Rightarrow B_1=C_1\otimes E_1=\left[1\right]\otimes\left[1\right]$$

$$N(j)=2,\quad \Rightarrow B_2=C_2\otimes E_2=\begin{bmatrix}1&1\\1&-1\end{bmatrix}\otimes\begin{bmatrix}a&a\\a&a\end{bmatrix}$$

$$N(j)=3,\quad \Rightarrow B_3=C_3\otimes E_3=\begin{bmatrix}1&1&1\\1&0&-1\\1&-2&1\end{bmatrix}\otimes\begin{bmatrix}b&b&b\\a&a&a\\c&c&c\end{bmatrix}$$

$$N(j)=4,\quad \Rightarrow B_4=C_4\otimes E_4=\begin{bmatrix}1&1&1&1\\2&1&-1&-2\\1&-1&-1&1\\1&-2&2&-1\end{bmatrix}\otimes\begin{bmatrix}a^2&a^2&a^2&a^2\\d&d&d&d\\a^2&a^2&a^2&a^2\\d&d&d&d\end{bmatrix}$$

where, $a=\sqrt{\dfrac{1}{2}},\ \ b=\sqrt{\dfrac{1}{3}},\ \ c=\sqrt{\dfrac{1}{6}},\ \ d=\sqrt{\dfrac{1}{10}}$.

The constants in matrix B_4 have been approximated and modified following a strategy similar to that used in IT[3] (see section 3). Thus equation 6 can be re-written as:

$$Y_j=\left(C_{N(j)}\otimes E_{N(j)}\right)\cdot X_j=\sum_{k=1}^{N(j)}\left(C_{N(j)(k)}\cdot X_{j(k)}\right)\otimes E_{N(j)(k)}\qquad 1\le j\le 4,\quad 1\le N\left(j\right)\le 4 \tag{7}$$

Note that $C_{N(j)}$ is the integer part of the $N(j)$x$N(j)$ integer transform matrix of column j of X_j and $C_{N(j)(k)}$ is the *kth* column of $C_{N(j)}$. Similarly, $E_{N(j)}$ and $E_{N(j)(k)}$ can be defined similarly but represent scaling factors, that arise when the integer transform matrix is factorised. Note that $X_{j(k)}$ represents the k^{th} internal pixel of X_j. The symbol ⊗ represents scalar multiplication as described in section 3.

Horizontal:

After carrying out the 1D-IT along columns as described in section 4.1 above, the transformed coefficient columns are left aligned as described in section 2. Subsequently following a strategy similar to that used in obtaining equation (7), the following equation could be obtained for 1D-IT along coefficient rows:

$$Z_i = \sum_{k=1}^{M(i)} \left(C_{M(i)(k)} \cdot Y_{i(k)} \right) \otimes E'_{M(i)(k)} \quad 1 \le M(i) \le 4,\ 1 \le i \le 4 \tag{8}$$

In the above equation, *M(i)* refers to the length of row *i* (i.e. the number of coefficients of row *i*). $C_{M(i)(k)}$, $Y_{i(k)}$ and $E'_{M(i)(k)}$ are defined accordingly as the equation (7) in section 4.1. Note that $E_{N(j)}$ consists of a total of 6 factors in equation (7), while $E'_{M(i)}$ is made up of 18 factors (Table 1) in equation (8).

4.2 Inverse transform
In terms of the equations (8) and (7) separately, the inverse transforms in both horizontal and vertical directions and their factorizations can be summarized by the following equations:

Horizontal:

$$Y_i = \sum_{k=1}^{M(i)} C^T_{M(i)(k)} \cdot \left(Z_{i(k)} \otimes E'^{\mathrm{T}}_{M(i)(k)} \right) \quad 1 \le M(i) \le 4,\ 1 \le i \le 4 \tag{9}$$

Vertical:

$$X_j = \sum_{k=1}^{N(j)} C^T_{N(j)(k)} \cdot Y_{j(k)} \qquad 1 \le N(j) \le 4,\ 1 \le j \le 4 \tag{10}$$

Here, $Z_{i(k)}$ and $Y_{j(k)}$ are respectively the k^{th} forward horizontal transform coefficient of row *i* of the SA-IT transformed block and the k^{th} inverse horizontal transform vector of column *j*. $C^T_{M(i)(k)}$, $C^T_{N(j)(k)}$ and $E'^T_{M(i)(k)}$ represent the transposes of $C_{M(i)(k)}$, $C_{N(j)(k)}$ and $E'_{M(i)(k)}$ respectively.

4.3 Quantisation
Forward Quantisation:

In equation (8), the output of forward transform consists of two parts, an integer part, $W_{M(i)(k)} = C_{M(i)(k)} \cdot Y_{i(k)}$, and a non-integer part (i.e. it consists of the post-scaling factors) $E'_{M(i)(k)}$. To maintain integer arithmetic within the core forward transform, in H.264/AVC, the above post scaling factors $E'_{M(i)(k)}$ are incorporated into the forward quantization process as follows. Thus, the integers, transformed coefficients, $W_{M(i)(k)}$ are quantized and scaled by a single operation as follows:

$$Q_i = round\left(\sum_{k=1}^{M(i)} W_{M(i)(k)} \otimes E'_{M(i)(k)} / Qstep \right) \tag{11}$$

where Q_{step} is a quantizer step size and there are a total of 52 values of Q_{step} are supported by the H.264/AVC standard, indexed by a Quantisztion Parameter, QP Table[1, 3]. Q_{step} doubles in size for every increment of 6 in QP. The rounding operation here rounds towards smaller integers.

Following the approach used within H.264/AVC reference model software[8], we apply the factor $E'_{M(i)(k)}/Qstep$ in equation 11, as a multiplication by a factor *MF* and a right-shift, thus avoiding any division operations. i.e., the equation 11 can be re-written as follows:

$$Q_i = round\left(\sum_{k=1}^{M(i)} W_{M(i)(k)} \otimes \frac{MF_{M(i)(k)}}{2^{qbits}}\right)$$

where

$$\frac{MF_{M(i)(k)}}{2^{qbits}} = \frac{E'_{M(i)(k)}}{Qstep}$$

and

$$qbits = 15 + floor(QP/6) \tag{12}$$

In integer arithmetic, equation (12) can be implemented as follows:

$$|Q_i| = \sum_{k=1}^{M(i)}\left(\left|W_{M(i)(k)}\right| \otimes MF_{M(i)(k)} + F_{M(i)}f\right) ?\ qbits = \sum_{k=1}^{M(i)} R_{M(i)(k)}$$

$$sign(Q_i) = sign\left(\sum_{k=1}^{M(i)} W_{M(i)(k)}\right) \tag{13}$$

where the symbol >> indicates a binary shift right. $F_{M(i)}$ is a column matrix of size $M(i)$ x 1 , in which all elements are 1. The so-called dead-zone control parameter f is set at $2^{qbits}/2$, in which *qbits* is derived by a quantization parameter (see equation), QP, an integer selected by the user. It determines the value of Q_{step} via a predefined lookup table[1, 3]. Further, to optimize the operational speed, the integer multiplication factors, $MF_{M(i)(k)}$ are obtained via Table 1 look-up. Note that in equation (13), $R_{M(i)(k)}$ is the sub-quantisation coefficient(s) of each $W_{M(i)(k)}$. Q_i represents the quantized coefficients of row *i*. For further details of the above quantization procedure we refer readers to[3, 5].

Factors	**QP**					
	0	**1**	**2**	**3**	**4**	**5**
a	37449	33825	29127	26886	23301	20971
b	30393	27962	23301	21845	19418	17050
c	21845	20560	16644	15887	13443	12052
d	17476	16131	13107	12336	10485	9532
a^2	26214	23831	20164	18724	16384	14563
ac	15887	14563	11650	10922	9709	8738
ad	13107	11650	9532	8738	7489	6553
a^3	18724	17476	14563	13797	11915	10485
bc	12945	11650	9709	8962	7767	7281
bd	9986	8738	7767	6990	6355	5377
cd	6990	6990	5825	4993	4369	3883
a^2c	10922	10922	8738	7943	6721	6241
a^2d	8738	8738	6553	6553	5242	4766
1	52428	47662	40329	37449	32768	29127
b^2	17924	16644	13706	12945	11096	9709
c^2	9709	8322	7281	6472	5825	4854
d^2	5242	5242	4194	4194	3495	2995
a^4	13107	11915	10082	9362	8192	7281

Table 1: Multiplication factor MF

The first six values of MF (for each coefficient position) used by the proposed SA-IT are given in Table 1. The factors MF remain unchanged but the divisor 2^{qbits} increases by a factor of two for each increment of six in QP when $QP > 5$.

Inverse Quantisation:

In inverse quantization (equation 14), the pre-scaling factor $E'^{T}_{M(i)(k)}$ (see equation (9)) is multiplied by a constant scaling factor of 64 to avoid rounding errors[3]:

$$W_i' = \sum_{k=1}^{M(i)} \left(R_{M(i)(k)} \otimes V_{M(i)(k)} \right) \cdot 2^{floor(QP/6)}$$

where

$$V_{M(i)(k)} = round\left(E'^{T}_{M(i)(k)} \cdot Qstep \cdot 64 \right) \tag{14}$$

The integer scaling factor, $V_{M(i)(k)}$ are defined in our proposal for $0 £ QP £ 5$ are shown in Table 2. W_i' are the de-quantized coefficients of row *i,* which is are subsequently transformed to the pixel domain by the core inverse transform in equations (9) and (10). The values at the output of the inverse transform are divided by 64 to remove the scaling introduced in equation (14). This is achieved using only an addition and a right-shift.

Factors	**QP**					
	0	**1**	**2**	**3**	**4**	**5**
a	28	31	36	39	45	50
b	23	25	30	32	36	41
c	16	17	21	22	26	29
d	12	13	16	17	20	22
a^2	20	22	26	28	32	36
ac	11	12	15	16	18	20
ad	8	9	11	12	14	16
a^3	14	15	18	19	22	25
bc	9	10	12	13	15	16
bd	7	8	9	10	11	13
cd	5	5	6	7	8	9
a^2c	8	8	10	11	13	14
a^2d	6	6	8	8	10	11
1	40	44	52	56	64	72
b^2	13	14	17	18	21	24
c^2	6	7	8	9	10	12
d^2	4	4	5	5	6	7
a^4	10	11	13	14	16	18

Table 2: Scaling factor V

5. EXPERIMENTAL RESULTS

An experiment was designed to evaluate the effectiveness of the proposed SA-IT algorithm in coding arbitrary shaped objects. The first frame of the video sequence Foreman (QCIF, grayscale Fig. 3) was used for experiments, with the assumption that the shape of the foreground object (i.e. of the Foreman) was known as an alpha-plane.

Firstly, the frame is segmented into two slice groups, namely, foreground (object, i.e. the man) and background, using the alpha-plane information. Subsequently the extracted arbitrary shaped foreground object is enclosed within a tightest fitting rectangle, which is later extended to enable the accommodation of full 4x4 blocks. Finally the enclosed 4x4 blocks are divided into boundary (a part of pixels belongs to the object), foreground (all pixels are inside of the object) and background (all pixels are outside of the object) blocks, and the proposed SA-IT is applied to all boundary blocks. Note that the normal IT can be applied to all foreground object blocks while coding background object blocks are ignored as they do not belong to the object being coded.

In Fig. 2, we illustrate the detailed (including intermediate stages) coding results for a selected arbitrary shaped boundary block. The reconstruction results of both foreground and background are given in Fig. 4(a) and Fig. 4(b) respectively.

$j=$	1	2	3	4
$i=1$				10
2			4	12
3		10		4
4			15	7

(a) Input block *X*

	10	4	10
		15	12
			4
			7

(b) Shifting of pels.

	10	19	33
		-11	14
			1
			-13

	1	a	a^2
		a	d
			a^2
			d

(c) Output of vertical transform: *Y* (left) and *E* (right)

	$10b+19ba+33ba^2$	$10a-33a^2$	$10c-38ca+33ca^2$
		$-11a^2+14ad$	$-11a^2-14ad$
			a^2
			$-13d$

(d) Output of horizontal transform: $Z = W$ Ä E'

	12	-2	-1
		-1	-5
			0
			-2

(e) Quantized values: Q

	604	96	-34
		-136	-248
			0
			-80

(f) Inverse quantized: W'

	666	288	618
		1056	746
			202
			330

(g) Output of inverse transform: X'

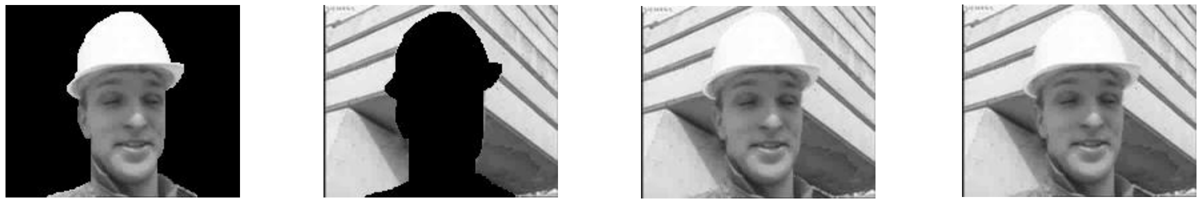
			9
		4	11
	10		3
		16	5

(h) Reconstructed block

Figure 2: An example of SA-IT of a arbitrary shaped block

Fig. 4(a) and 4(b) illustrate the separation of the original Forman image of Fig. 3, into foreground and background objects. Fig. 4(c) and 4(d) illustrate the reconstructed images using the proposed SA-IT scheme and the standard IT scheme. Identical quantizer parameter (QP = 28) have been used to code all blocks in Fig. 4(c). No difference is visual quality is observed between the two images. A comparison of PSNR values of the reconstructed images when using different QP values is shown in Table 3. The results justify that the PSNR values are comparable.

Fig. 3 Original image



(a) Reconstructed foreground (b) Reconstructed background (c) Reconstructed by SA-IT (d) Reconstructed by IT

Fig. 4 A comparison of proposed SA-IT and traditional IT

QP	PSNR(SA-IT)	PSNR(IT)
4	50.88dB	50.88dB
10	48.78dB	48.91dB
15	45.46dB	45.68dB
28	35.79dB	36.17dB

Table 3: Comparison of PSNR values

As mentioned earlier, a functionality of significant practical importance of the proposed SA-IT scheme is its ability to code foreground and background areas at different bit rate / quality. For example, less important background regions within a frame may be encoded with lower quality and more important foreground regions/objects can be coded at higher quality. We demonstrate this enhance functionality in our second experiment described below.

Fig. 5 illustrates the comparison of visual image quality between two images compressed at the same rate (i.e. compression ratio). For the image compressed with the traditional IT scheme we have used a QP of 37 to achieve the fixed bit rate. In contrast when using the SA-IT scheme, for the foreground we have used a QP of 10 and for the background a QP of 48. Note the clear improvement of the foreground object quality in (b) as compared to that of (a) and the clear degradation of quality in the background region of (b) as compared to that of (a). A similar coding strategy has many applications in practice.

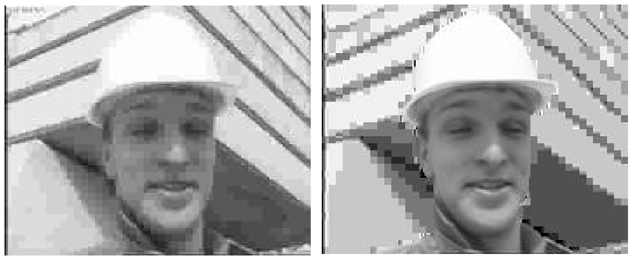


Fig. 5 Comparison of image quality based on 64% of compression ratio. (a) Image in Fig. 3 compressed using the Int-DCT scheme with a QP = 37. (b) Image in Fig. 3 compressed foreground and background using the SAI-DCT with QP = 10 and QP=48 separately.

Com. Rate	IT		Proposed SA-IT			
	QP	PSNR	QP(BG)	QP(FG)	PSNR	PSNR(FG)
53%	28	36.17	33	4	33.64	55.51
64%	37	29.76	48	10	22.35	53.54

Table 4: PSNR comparisons and QP considerations at different compression rates

Further results related to the above experiment are tabulated in Table 4. BG and FG refer respectively to the background and foreground. Results illustrate the additional coding flexibility offered by the proposed scheme.

6. CONCLUSIONS

In this paper, we have proposed a Shape-Adaptive Integer Transform algorithm for coding arbitrary shaped objects in H.264/AVC. The standard stages of IT based CODEC used in H.264/AVC have been modified and extended to enable arbitrary shaped object coding. The SA-IT approach adopted here is similar to the previous work of SA-DCT and SA-DWT, yet resolves additional mathematical challenges due to the requirement that integer arithmetic should be maintained at all stages. Experimental results have been provided to justify the accuracy and functionality of the proposed method. We have shown that the proposed scheme is useful in region-of-interest based coding applications that may require important foreground objects to be coded at a comparatively higher quality as compared to the background regions. Though the implementations carried out within our present research context is a proof of concept, we are currently in the process of incorporating the proposed methodology within a standard H.264/AVC implementation.

Further research challenges exists in designing data-structures and algorithms for maintaining arbitrary shaped slice groups, adaptive setting of quantization parameters and shape coding within auxiliary information streams.

REFERENCES

1. Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, "Draft ITU-T recommendation and final draft international standard of joint video specification ITU-T Rec. H.264/ISO/IEC 14 496-10 AVC", JVTG050, 2003.
2. T. Wiegand, G. Sullivan, G. Bjontegaard and A. Luthra, "Overview of the H.264 /AVC Video Coding Standard", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, No. 7, pp. 560- 576, 2003.
3. Iain E. G. Richardson, *H.264 and MPEG-4 Video Compression Video Coding for Next-generation Multimedia*, John Wiley & Sons Ltd, England, 2003.
4. T. Sikora and B. Makai, "Shape-Adaptive DCT for Coding of Arbitrarily Shaped Image Segments", Signal Processing: Image Comm., Vol. 7, pp. 381-395, 1995.
5. H. S. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, "Low-Complexity Transform and Quantization in H.264/AVC", IEEE Transactions on Circuits and Systems for Video Technology, Vol 13, pp. 598-603, 2003.
6. K. R. Rao and P. Yip, "Discrete Cosine Transform", Academic Press, 1990.
7. A. Kaup and S. Panis, "On the Performance of the Shape Adaptive DCT in Object-Based Coding of Motion Compensated Difference Images", Picture Coding Symposium, ITG-Fachbericht 143, pp. 653-657, Sep. 1997.
8. H.264 Reference Software Version JM9.2 http://bs.hhi.de/~suehring/tml/, March 2005.