Signal Processing: *Image Communication* 24 (2009) 263–276

Contents lists available at ScienceDirect

Signal Processing: *Image Communication*

journal homepage: www.elsevier.com/locate/image

Context-based entropy coding in AVS video coding standard

Li Zhang [a,*], Qiang Wang [b], Ning Zhang [c], Debin Zhao [b], Xiaolin Wu [c], Wen Gao [d]

[a] *Key Lab of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China*
[b] *Department of Computer Science and Engineering, Harbin Institute of Technology, Harbin, China*
[c] *Department of Electrical and Computer Engineering, McMaster University, Canada*
[d] *Institute of Digital Media, School of Electronic Engineering and Computer Science, Peking University, Beijing, China*

ARTICLE INFO

Article history:
Received 2 June 2008
Received in revised form
10 November 2008
Accepted 19 December 2008

Keywords:
AVS
DCT video coding
Entropy coding
Context modeling
Variable length coding
Arithmetic coding

ABSTRACT

In this paper, two context-based entropy coding schemes for AVS Part-2 video coding standard are presented. One is Context-based 2D Variable Length Coding (C2DVLC) as a low complexity entropy coding scheme for AVS Part-2 Jizhun profile. C2DVLC uses multiple 2D-VLC tables to exploit the statistical features of DCT coefficients for higher coding efficiency. Exponential–Golomb codes are applied in C2DVLC to code the pairs of the run-length of zero coefficients and the non-zero coefficients for lower storage requirement. The other is Context-based Binary Arithmetic Coding (CBAC) as an enhanced entropy coding scheme for AVS Part-2 Jiaqiang profile. CBAC utilizes all previously coded coefficient magnitudes in a DCT block for context modeling. This enables adaptive arithmetic coding to exploit the redundancy of the high-order Markov process in DCT domain with a few contexts. In addition, a context weighting technique is used to further improve CBAC's coding efficiency. Moreover, CBAC is designed to be compatible to C2DVLC in coding elements which simplifies the implementations. The experimental results demonstrate that both C2DVLC and CBAC can achieve comparable or even slightly higher coding performance when compared to Context-Adaptive Variable Length Coding (CAVLC) in H.264/AVC baseline profile and Context-Based Adaptive Binary Arithmetic Coding (CABAC) in H.264/AVC main profile respectively.

1. Introduction

AVS video coding standard [4] adopts the motion-compensated hybrid coding framework. In such a framework, intra and inter predictions [19] are first used to remove the spatial and temporal correlations, generating prediction errors. Then the prediction errors are transformed by DCT, quantized and finally entropy coded. Entropy coding is used for data compression and stream organization. In high-quality video coding, most of the bit budget is spent on transformed prediction errors, called DCT coefficients. Consequently, how efficiently DCT coefficients are entropy coded will significantly impact the coding efficiency of the whole video coding.

Context-based entropy coding uses context modeling to exploit statistical behaviors of sequentially observed symbols for higher coding efficiency. The context modeling is used to select a specific context from a given context set for a symbol which will be coded. Generally speaking, a specific context corresponds to a specific statistical behavior. Since the estimated conditional entropy based on the selected context for the conditional coding of a symbol should maximally approach the actual one, coding conditioned on contexts can adapt to symbols' local statistical variations and therefore can improve coding efficiency. One of the most famous context modeling methods is Rissanen's [11] context modeling for universal data coding. It can theoretically approach the bound of the

* Corresponding author.
E-mail address: zhanglili@jdl.ac.cn (L. Zhang).

0923-5965/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.image.2008.12.001

minimal code length based on the concept of stochastic complexity [15]. Besides, there are efficient context modeling methods [15–17] for image coding.

In video coding, the statistical behaviors of DCT coefficients are usually diverse. This diversity can be observed on different coding conditions, e.g., different video contents and different quantization step sizes. In particular, DCT coefficients at different frequency subbands also show different statistical behaviors. Context-based entropy coding is to exploit such diverse statistical behaviors to achieve higher coding efficiency.

Observed domain knowledge, which is an abstraction of statistical behaviors, is usually used to guide context modeling in context-based video entropy coding. The entropy coding in MPEG-2 [6] is an example. MPEG-2 can use two Variable Length Coding (VLC) tables for adaptively coding intra or non-intra DCT coefficients. Another kind of well-known domain knowledge in DCT coefficients is, along the zig-zag path of DCT blocks,[1] non-zero coefficients show a statistical decreasing tendency in magnitude and the run-length of successive zero coefficients shows a statistical increasing tendency. This domain knowledge has guided the context modeling in Context-Adaptive Variable Length Coding (CAVLC) [2] and Context-Based Adaptive Binary Arithmetic Coding (CABAC) [8] in H.264/AVC [5]. For example, CAVLC uses multiple contexts, each associated with a VLC table, to adapt to such statistical tendencies, so that local statistical variations of DCT coefficients even in one DCT block are exploited which brings further coding efficiency.

This paper presents two context-based entropy coding schemes for coding DCT coefficients in AVS Part-2 video coding standard. One is Context-based 2D Variable Length Coding (C2DVLC) [13] for AVS Part-2 Jizhun profile and the other is Context-based Binary Arithmetic Coding (CBAC) for AVS Part-2 Jiaqiang profile. C2DVLC and CBAC can be used for different application purposes. For example, C2DVLC has lower complexity than CBAC, which can be used in computational resource constrained applications. CBAC has higher coding efficiency than C2DVLC but with extra computational costs, which can be used in applications requiring higher coding efficiency. Both C2DVLC and CBAC adopt context modeling to achieve higher coding efficiency. The context modeling is also inspired by the above mentioned domain knowledge, but is designed to fully exploit the statistical features of 8×8 DCT coefficients (AVS Part-2 video coding adopts 8×8 DCT [18]). In particular, CBAC is designed to be compatible to C2DVLC in coding elements. In terms of coding efficiency, C2DVLC and CBAC achieve comparable or even slightly higher coding performance compared to CAVLC and CABAC, respectively.

The rest of this paper is organized as follows. In Section 2, the statistical features of DCT coefficients are analyzed first, and then the coding elements of DCT coefficient and the coding order used in C2DVLC and CBAC are introduced. Section 3 overviews C2DVLC and describes the used context modeling and Exponential–Golomb (E–G) [12] codes. Section 4 presents CBAC in detail, including the underlying ideas behind its design. The coding performance of C2DVLC and CBAC is evaluated in Section 5. Section 6 concludes this paper.

[1] A DCT block denotes a block of DCT coefficients, as the hybrid coding framework is block-based.

2. Statistical features of DCT coefficients, coding elements, and coding order

2.1. Statistical features of DCT coefficients

In a typical DCT block, e.g., in progressive videos, non-zero coefficients are always clustered around the top-left corner and roughly symmetrically positioned in the horizontal and vertical directions. This is because the statistical distributions of DCT coefficients at low-frequency DCT subbands have larger variances than those at high-frequency ones (their expectations are all zero), which is similar to the case in image coding as pointed out in Ref. [7]. Therefore, the symmetrical scan pattern, e.g., the zig-zag scan, is usually used to reorganize DCT coefficients. After reorganization, the coefficients in a DCT block are arranged into a one-dimensional list. Before entropy coding, especially in VLC, the list is further represented by two kinds of symbols: non-zero coefficient, denoted as *Level*, and the number of successive zero coefficients before a *Level*, denoted as *Run*. In a statistical view, DCT coefficients always exhibit the following statistical features, expressed via *Level* and *Run*.

(1) The magnitude of *Level* shows a statistical decreasing tendency while *Run* shows a statistical increasing tendency along the zig-zag scan path.
(2) *Level* and *Run* are correlated, e.g., a *Level* with a smaller magnitude is more likely to be preceded by a larger *Run*. Fig. 1 shows the probability distributions of *Run* under different *absLevel*s, where *absLevel* means absolute value of *Level*. Comparing Fig. 1(a) and (b), we can see that the correlations between *Level* and *Run*. For example, a smaller magnitude of *Level* is more likely to be preceded by a larger *Run*. Fig. 1 is based on the statistical data of 8×8 DCT blocks under AVS Part-2 video platform, where the '*City*' sequence in 720p format is used.
(3) (*Level*, *Run*) pairs have varying statistical distributions along the zig-zag scan path even in one DCT block. Fig. 2 shows the probability distributions of (*absLevel*, *Run*) pairs by the scanned positions. It can be seen that the distribution shape of the first scanned (*absLevel*, *Run*) pair along the zig-zag path is much sharper than that of the fifth scanned pair. This demonstrates the statistical non-stationary of (*Level*, *Run*) pairs.

The above statistical features are the observed domain knowledge of DCT coefficients. They should be used to guide the context modeling design, and meanwhile the desired context modeling should fully exploit these statistical features to achieve higher coding efficiency.

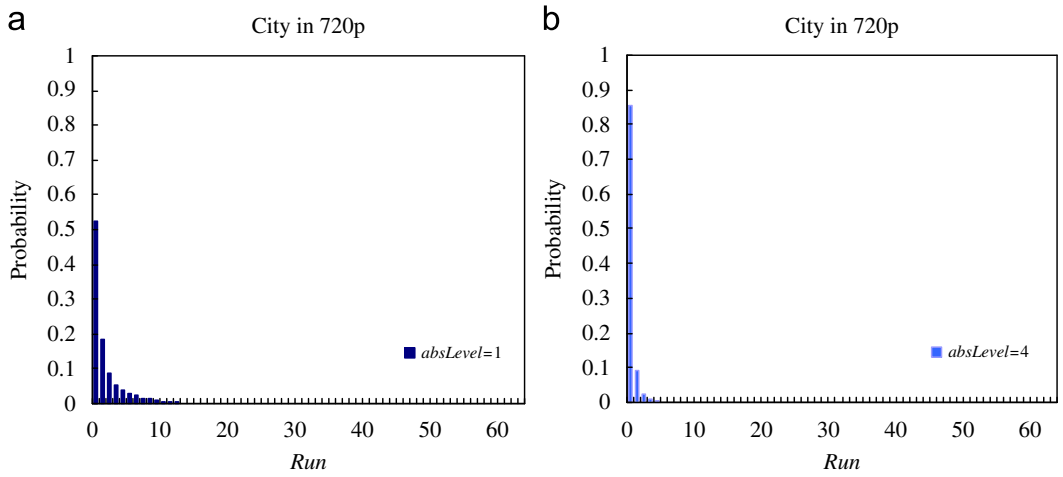


Fig. 1. Probability distributions of *Run* under (a) *absLevel* = 1 and (b) *absLevel* = 4 in City 720p video.

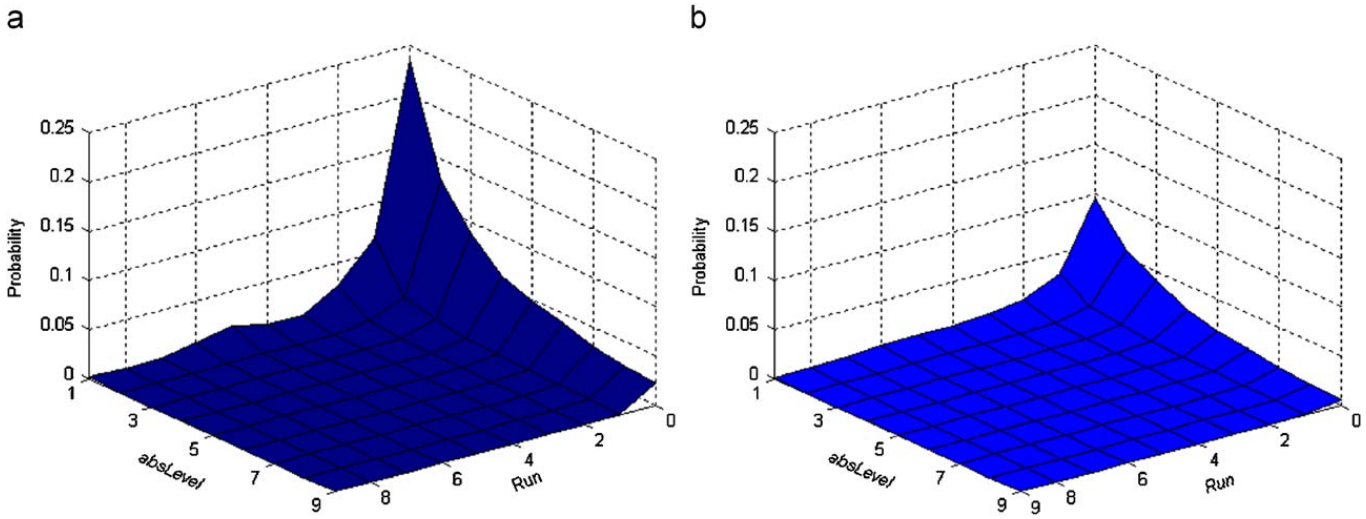


Fig. 2. Probability distributions of (*absLevel, Run*) pairs in City 720p video. (a) The first scanned pair along the zig-zag scan path and (b) the fifth scanned pair along the zig-zag scan path.

2.2. Coding elements

(*Level*, *Run*) is an efficient representation of DCT coefficients, which has been widely used. The coding elements used in C2DVLC and CBAC are also *Level* and *Run*. In particular, a special symbol, *EOB*, is used to signal the coding end of a DCT block. *EOB* is denoted as (0, 0) in this paper. So with respect to coding elements, C2DVLC and CBAC are compatible.

2.3. Coding order

As pointed out in Section 2.1, the magnitude of *Level* exhibits a descending tendency along the zig-zag scan path. Then the dependence among *Level*s can be utilized, which will be greatly useful for context modeling. As will be described in Sections 3 and 4, both C2DVLC and CBAC rely on *Level*'s changing tendency to identify large statistical variations and design contexts. It is easy to understand that *Level*'s changing tendency in different coding order exhibits different statistical behaviors. Thus, before we begin to discuss the design of context modeling, we need to explain why the coding order is the reverse zig-zag scan order.

Fig. 3 shows the probability distributions of the first scanned *absLevel* both in the zig-zag scan order and in the reverse zig-zag scan order in form of histograms. Fig. 3 is based on the statistical data of 8×8 DCT blocks under AVS Part-2 video platform, where the '*News*' sequence in CIF format is used under QP = 27. It can be easily observed that in the zig-zag scan order, the magnitude of the first *Level* has a more scattered distribution. On the contrary, in the reverse scan order, the probability of the first *Level*'s magnitude equal to 1 takes a significant large percentage (roughly more than 96%). The similar results can also be found in the other videos. This indicates that the first scanned *absLevel*'s distribution in the zig-zag scan order has a larger variance than that in the reverse zig-zag scan order.

Then, if coding DCT coefficients along the zig-zag scan order, we will meet the following problems. First, the

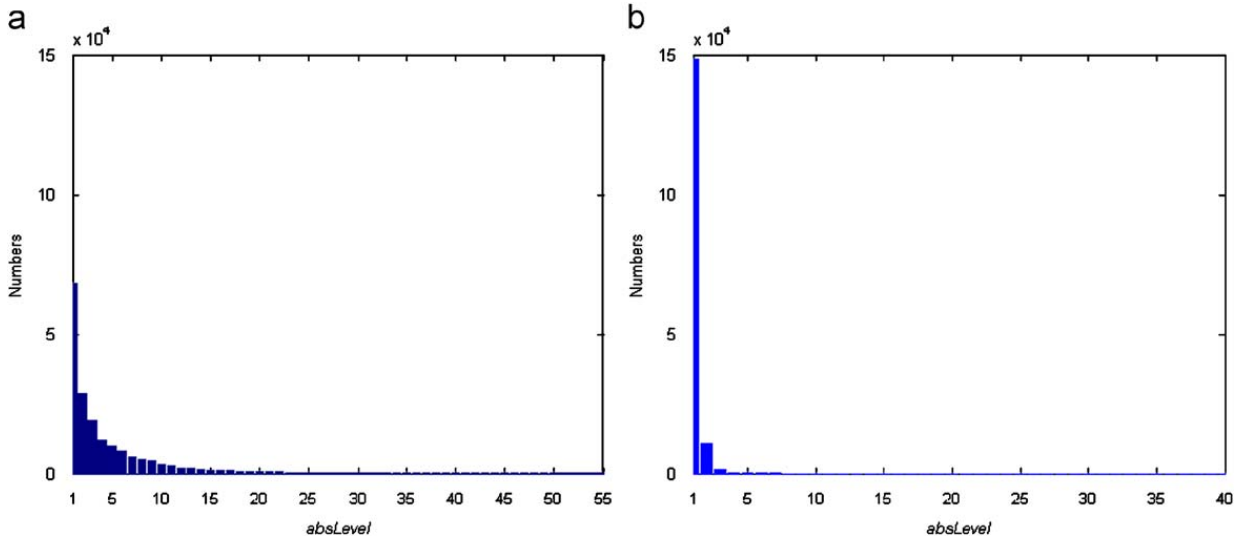


Fig. 3. Histograms of the first *absLevel* (a) in the zig-zag scan order and (b) in the reverse zig-zag scan order in *News* CIF video.

Table 1
Example of coding order of a DCT Block.

Scanning position	1	2	3	4	5	6	7	8	9	…	64
Reorganized DCT coefficients	9	−2	3	0	−2	0	0	−1	0	0	0
(*Level*, *Run*) pairs in scan order	(9, 0), (−2, 0), (3, 0), (−2, 1), (−1, 2), (0, 0)										
Coding order	(−1, 2), (−2, 1), (3, 0), (−2, 0), (9, 0), (0, 0)										

scattered distribution increases the uncertainty of the first *Level*. Thus we cannot make a good estimation of the probability of magnitude of the first *Level* and the bits used for coding the first *Level* will be more than that in the reverse zig-zag scan order. Second, when coding in the zig-zag scan order, the larger variance of the first *Level* leads to multiple choices of contexts, and there will not be enough samples to accurately estimate the context variables. Therefore, the probability estimations of the subsequent *Level*s will be unreliable. Third, several non-zero coefficients in the end of zig-zag scan order usually constitute a subsequence of ± 1. This will increase the difficulty of determining the coding end of a block and the contexts based on previously coded symbols will be mixed up without distinguishing the frequency positions of different ± 1. On the contrary, coding in the reverse scan order makes it easier to follow the tendency of *Level* variation. When coding in the reverse scan order, the degree of increasing tendency can be easily estimated based on the maximal magnitude of previously coded *Level*s. Due to the reasons listed above, C2DVLC and CBAC use the reverse scan order as coding order.

In the sequel, for a given DCT block with at least one non-zero coefficient, we use N to represent the number of non-zero coefficients and suppose $(L_{N-1}, R_{N-1}),\ldots,(L_1, R_1)$, (L_0, R_0) are the sequence of the (*Level*, *Run*) pairs formed after the zig-zag scan, where they are indexed along the reverse scan order. The coding is in the reverse scan order, which means (L_0, R_0) is coded first and (L_{N-1}, R_{N-1}) is coded last. Table 1 gives an example of reorganizing an 8×8 DCT block by scanning, produced (*Level*, *Run*) pairs as well as the coding order for these pairs. The symbol (0, 0) denotes *EOB*.

3. Context-based 2D variable length coding

In this section, C2DVLC for AVS Part-2 Jizhun profile is presented. First, we outline the basic coding procedure of C2DVLC. The algorithm block diagram of C2DVLC is depicted in Fig. 4. Then, we describe how the context modeling in C2DVLC is designed by exploiting the statistical features presented in Section 2.1. Last, we introduce the E–G codes used in C2DVLC.

3.1. Overview of C2DVLC

C2DVLC codes each (L_i, R_i) $(i = 0, 1,\ldots,N-1)$ one by one with multiple off-line trained 2D-VLC tables until all pairs are coded. At last, *EOB* is coded to indicate that there are no more non-zero coefficients in the block.

Fig. 4 illustrates the block diagram of C2DVLC encoder to highlight how a (*Level*, *Run*) pair represented by (L_i, R_i) is entropy coded. During the process of (L_i, R_i) encoding, three steps are performed in turn. First, the table index should be calculated. Here, the adopted multiple VLC tables are two-dimensional to utilize the correlation between *Level* and *Run*. For high coding efficiency, these VLC tables are designed for switch in order to adapt to the varying statistical distributions of (*Level*, *Run*) pairs.

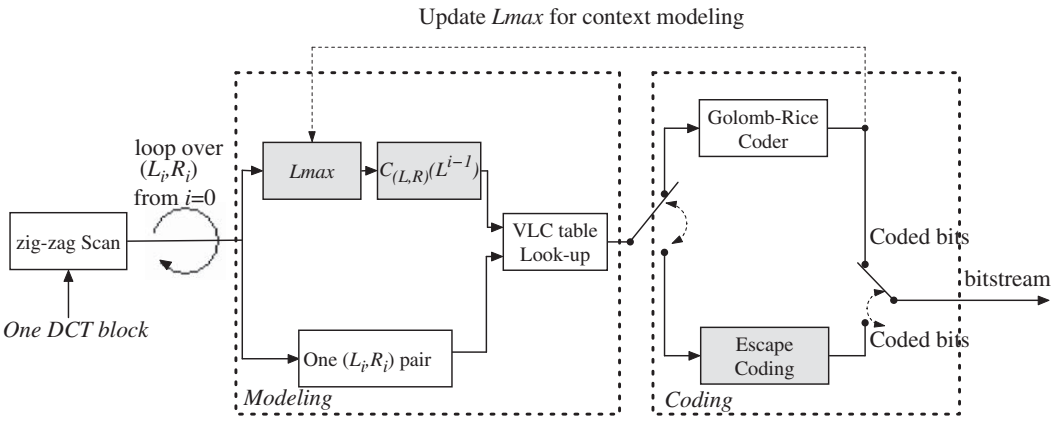


Fig. 4. C2DVLC encoder block diagram.

Table switch is based on the maximal magnitude of prior coded *Level*s before the current (L_i, R_i) pair, denoted as L_{max}. Therefore, the table index selected for current (L_i, R_i) pair can be represented by $C_{(L,\ R)}(L^{i-1})$ (L^{i-1} denotes the past symbols of L_i). For the first pair (L_0, R_0) to be coded in one block, the table index is fixed to be zero. Second, with the table index selected in the first step, the mapping between (L_i, R_i) and one *CodeNumber* has to be done through a table look-up operation. *CodeNumber*s in different VLC tables reflect different estimated conditional probabilities of the current (L_i, R_i) pair. Finally, the corresponding unique (E–G) codeword of *CodeNumber* is output, and one pair coding is done. E–G codes are used in C2DVLC for simplicity and efficiency. Note that if the current (L_i, R_i) pair is out of the VLC table, *Level* and *Run* will be coded separately by escape coding. At the same time, the update of L_{max} should be performed. For detail information about escape coding, please refer to Ref. [3].

3.2. Context modeling in C2DVLC

This subsection provides detailed information for the context modeling of (L_i, R_i) pairs, i.e., how to adaptively switch 2D-VLC tables in one DCT block.

Based on the analysis in Section 2.1, we can utilize the statistical increasing in magnitude of *Level* from L_0 to L_{N-1} for context modeling. Another effective description of *Level* increase is the update of variable L_{max} defined as that in Section 3.1. For the first pair (L_0, R_0) to be coded in one block, L_{max} is initialized to be zero.

Here, we first denote the past symbols of L_i by

$$L^{i-1}=\begin{cases} L_{i-1},L_{i-2},\ldots,L_0, & \text{if } 1\leqslant i<N \\ \varnothing, & \text{if } i=0 \end{cases}. \tag{1}$$

Then the context modeling for (L_i, R_i) can be defined as

$$C_{(L,R)}(L^{i-1})=j \quad \text{if } (Th[j+1]>L_{\max}\geqslant Th[j])) \tag{2}$$

with $C_{(L,\ R)}(L^{-1})=0$ and the threshold array is

$$Th[0\ \ldots 7]=\begin{cases} \langle 0,\ 1,\ 2,\ 3,\ 5,\ 8,\ 11,\ \infty\rangle & \text{intra\_luma} \\ \langle 0,\ 1,\ 2,\ 3,\ 4,\ 7,\ 10,\ \infty\rangle & \text{inter\_luma} \\ \langle 0,\ 1,\ 2,\ 3,\ 5,\ \infty,\ \infty,\ \infty\rangle & \text{chroma} \end{cases}. \tag{3}$$

Here the values of $C_{(L,\ R)}(\cdot)$ are the indices of the contexts, i.e., the 2D-VLC table indices. According to Eqs. (2) and (3), the value set of $C_{(L,\ R)}(\cdot)$ is {0, 1,…,6} for luminance component and {0, 1,…,4} for chrominance component. Each of them corresponds to one different context, which corresponds to an interval I_k derived from $Th[0\ldots7]$. The lower bound of I_k is equal to $Th[k]$ while the upper bound is $Th[k+1]$. For example, for the luminance part of intra mode, $I_0=[0]$, $I_3=[3,5)$, and $I_6=[11,\infty)$.

The table switch process can be seen as a recursive context transition process, which can be described as

$$C_{(L,R)}(L^{i-1})=\begin{cases} j, & \text{if } (abs(L_{i-1})\geqslant Th[C_{(L,R)}(L^{i-2})+1] \text{ and } (Th[j+1]>abs(L_{i-1})\geqslant Th[j])) \\ C_{(L,R)}(L^{i-2}), & \text{if } abs(L_{i-1})<Th[C_{(L,R)}(L^{i-2})+1] \end{cases}. \tag{4}$$

Eq. (4) actually describes that an appearance of a *Level* indicated by L_{i-1}, if its magnitude is equal or larger than the current used I_k's upper bound or I_{k+1}'s lower bound, i.e., $Th[k+1]$, triggers the context transition from one context $C_{(L,\ R)}(L^{i-2})$ to another $C_{(L,\ R)}(L^{i-1})$. That is, the prior coded $abs(L_{i-1})$ is larger than the upper bound of $C_{(L,\ R)}(L^{i-2})$'s corresponding interval. The new context is determined from the maximum previously coded magnitude among (L^{i-1}). Such context transition can exploit the increasing trend in magnitude from L_0 to L_{N-1} as well as the sequential dependency. Moreover, the use of the maximum values of previously coded *absLevel*s instead of the nearby coded *Level* can effectively deal with those coefficients which are not in a monotonously increasing direction. In this way, the drawback of the traditional VLC coding with one single VLC table in one DCT block, which cannot adapt to locally statistical variations leading to low efficiency, can be solved efficiently.

3.3. Exponential–glomob codes

For the final VLC coding, codewords are constructed based on E–G codes. In C2DVLC, *k*th order E-G codes

with k equal to 0, 1, 2, and 3 are used. The order for each table is determined by the distribution of (*Level*, *Run*) pairs under the corresponding context. Table 2 lists part of E–G codes when k's values are 0, 1, and 2. We can see E–G codes have a regular construction, which consists of a prefix and a suffix. Given a *CodeNumber* N and a specific order k, the prefix part consists of l zeros followed by one and the suffix part is the binarization representation of value $N-2^k(2^l-1)$. l is given by

$$l = \max\{0, \lceil \log_2((N+1)/2^{k+1} + 1/2)\rceil\}. \quad (5)$$

Due to the regular codeword structure, E–G codes can be real-time constructed in coding process without involving high computational complexity. Thus, the entries stored in 2D-VLC tables could be mapping relationships (*CodeNumber*s) from (*Level*, *Run*) pairs to E–G codewords instead of real codes. This is a valuable feature that resolves the problem of high memory requirement resulted from multiple 2D-VLC tables.

Table 2
Example of Exponential–Golomb codes of order 0, 1, and 2.

Code number ($N=0$–4)	Code words					
	$k=0$		$k=1$		$k=2$	
	Prefix	Suffix	Prefix	Suffix	Prefix	Suffix
0	1	–	1	0	1	00
1	01	0	1	1	1	01
2	01	1	01	00	1	10
3	001	00	01	01	1	11
4	001	01	01	10	01	000

4. Context-based binary arithmetic coding

In C2DVLC, the VLC tables are pre-defined by off-line training which cannot capture the local statistical variations in one context and a symbol with its probability greater than 0.5 cannot be efficiently coded due to the intrinsic limit of 1 bit/symbol of VLC codes. Arithmetic coding can naturally avoid these problems for higher coding efficiency. In this section, CBAC for AVS Part-2 Jiaqiang profile is presented. In Subsection 4.1, we introduce the basic coding structure of CBAC coding. The procedure of CBAC coding is depicted in Fig. 5. In the subsequent subsections, the individual key techniques in CBAC, including symbol binarization, context formation and quantization, binary arithmetic coder, are discussed in detail.

4.1. Overall coding structure of CBAC

In CBAC, coding a data symbol involves the following steps: (a) binarization, (b) context model selection and (c) arithmetic encoding. For a given non-binary valued syntax element, it is uniquely mapped to a binary sequence, a so-called bin string. Each of the given binary decision, which referred to as a bin in the sequel, enters the context modeling stage, where a context is selected and the corresponding choice of contexts may depend on previously encoded syntax elements or binarized bins. Then, after the assignment of a context, the bin value along with its associated model is passed to the regular coding engine or bypass coding, where the final stage of arithmetic encoding together with a subsequent context updating takes place.

Fig. 5 depicts the coding block diagram of (*Level*, *Run*) pairs in one DCT block. As C2DVLC does, CBAC codes each (L_i, R_i) $(i = 0, 1, \ldots, N-1)$ pair one by one along the reverse scan order until all pairs are coded. A so-called *EOB*

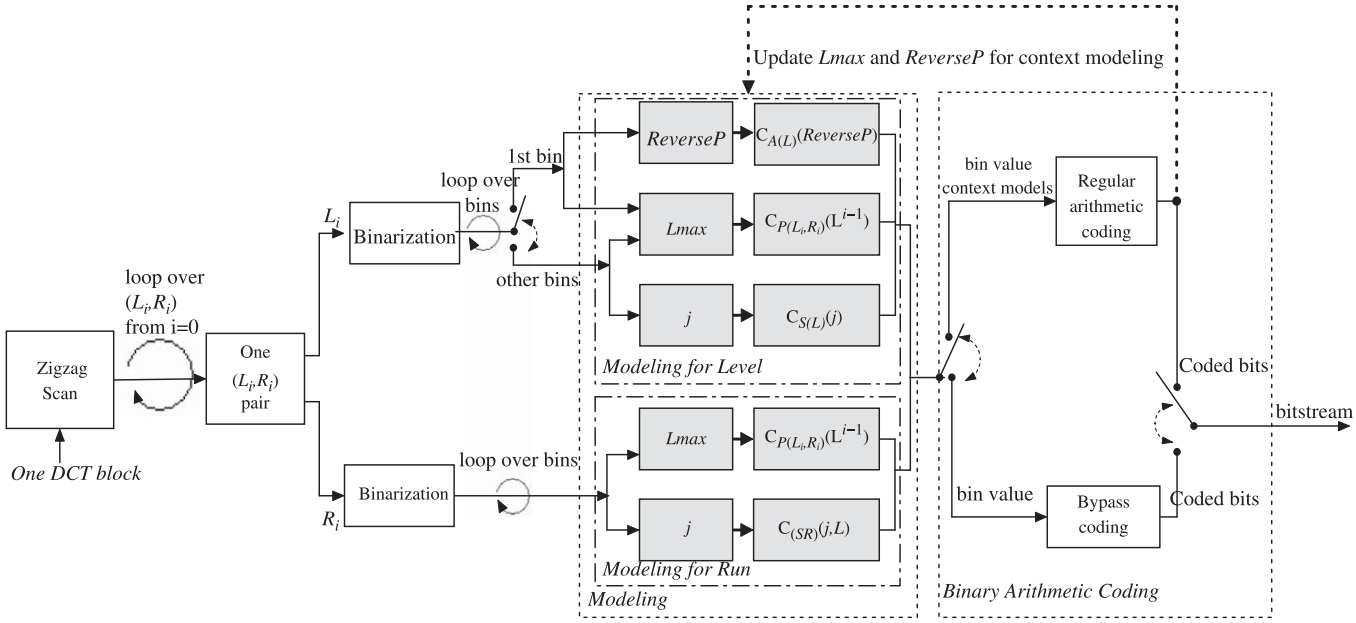


Fig. 5. Block diagram of CBAC encoder.

symbol is coded at last to signal the end of a DCT block. For each (L_i, R_i) pair, the *Level* precedes the associated *Run*. Firstly, both *Level* and *Run* are unary binarized into several bins. For the signed integer *Level*, it is presented by sign and unary bits of its magnitude (*absLevel*). Secondly, for each bin of *absLevel* and *Run*, a product context is applied, which consists of a *primary* context $C_{P(L,\ R)}(L^{i-1})$ and a *secondary* context indicated by $C_{S(L)}(j)$ for *Level* or $C_{S(L)}(j, L)$ for *Run*. *Primary* context relies on the past coded L^{i-1}, and the corresponding context index is determined by the variable L_{max} which denotes the maximal prior coded *absLevel*, as the same in C2DVLC. To keep the number of contexts used for coefficient coding reasonably small, the *primary* contexts are quantized into five categories. Under each *primary* context, seven nested *secondary* contexts are defined. They are classified according to the bin indices for *Level* or both the value of currently coded *absLevel* and bin indices for *Run*. In the following, the bin index of *absLevel* or *Run* is denoted as variable *j*. The *secondary* context index is first initialized with the value of zero at the beginning of (*Level*, *Run*) pair coding. Besides, for the first bin of *absLevel*, another so-called *accompanying* context $C_{A(L)}$(*ReverseP*) which utilizes the position of *absLevel* in coded order is designed for context weighting. It is quantized by the variable *ReverseP*. At last, the first bin of *absLevel* is sent to regular binary arithmetic coder with the technique of context weighting using *secondary* context and *accompanying* context. All other bins of *absLevel* and *Run* are regularly coded according to the *secondary* context index. Besides, the sign of *Level* is coded with bypass coding. After one (*Level*, *Run*) instance has been coded, all these contexts are updated.

In summary, CBAC contains the following features:

- To be compatible with C2DVLC in coding syntax elements: (*Level*, *Run*) pairs and *EOB*; coding order in the reverse scan.
- Unary binarization scheme.
- Context quantization according to all previously coded magnitudes of *Level*s.
- Context weighting technique.

4.2. Symbol binarization

The coding elements of CBAC is (*Level*, *Run*) pairs. It is easy to understand that the symbol values of *Level* and *Run* are integers in a large range in a DCT block. Coding these values directly by an *m*-ary (for $m>2$) arithmetic code will have a high computational complexity. Besides, the source with typically large alphabet size often suffers from "context dilution" effect when the high-order conditional probabilities have to be estimated on a relatively small set of coding samples. Here, binary arithmetic code is adopted in CBAC. Therefore, for the non-binary valued symbols, e.g. *Level* and *Run*, they should perform binarization before sending to arithmetic coder. The binarization process is as follows:

- The signed integer *Level* is represented by sign (0/1: +/−) and the unary bits of its magnitude (*absLevel*).
- The positive integer *Run* is simply represented by unary bits.

Table 3
Unary binarization.

N	Unary representation (Bin string)							
0	1							
1	0	1						
2	0	0	1					
…	–	–	–	–				
5	0	0	0	0	0	1		
…	–	–	–	–	–	–	–	
Bin index (*j*)	0	1	2	3	4	5	6	…

Table 3 lists part of the mapping from unsigned numbers to unary binary representation. We can see that the binarized representation has a regular code structure, which is a concatenation of a prefix code and a suffix code. Given a nonnegative number *N*, the prefix part consists of $n-1$ zeros and the suffix part is fixed to be a one.

4.3. Context formation and quantization

We model the symbol sequence as a high-order Markov process, and compress it by context-based arithmetic coding. A key issue in context modeling of an input symbol sequence is how to balance the desire of using a high-order context modeling technique against the context model cost. As we know from the view of information theory, the higher the conditional entropy, the more an observer can predict the state of a variable, knowing the state of the other variables. Therefore, if the context order is not sufficiently high, it will not be able to capture all the local statistical behaviors of the source sequence. But on the other hand, if the order of the model is too high, there will not be enough samples to accurately estimate the context parameters, causing context dilution problem. To solve this problem, CBAC adopts a novel context quantization technique that generates only 34 context states out of a very large causal context, as described below.

To reduce the context model cost, i.e. the number of coding states, the coding context is formed as a product of two classified contexts: one *primary* context and one *secondary* context.

4.3.1. Primary contexts

DCT cannot completely remove the statistical redundancy between coefficients in different subbands. It is observed that the coefficient magnitudes after DCT transform strongly correlate to the previously coded coefficients. To utilize this correlation, we can define a function to characterize the relationship among *Level*s. This function can be determined in an off-line design process, and it can be designed to be a linear or more sophisticated function. As we mentioned in Section 2, coding in the reverse scan order is easy to follow the increasing tendency based on *Level* information. Meanwhile, *Run* is correlated to *Level*. To utilize this domain knowledge, we define the function as the *primary* context based on the

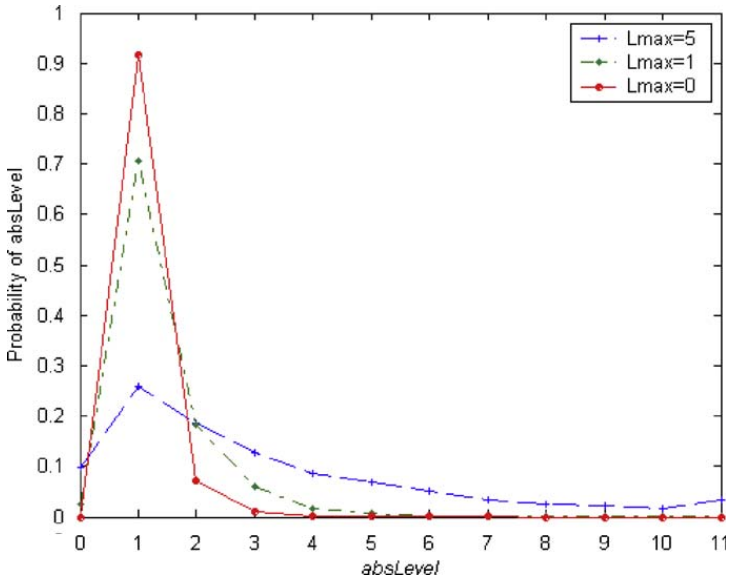


Fig. 6. Probability distributions of *absLevel* according to L_{max} in Harbour 720p video.

maximum of all previously coded magnitudes of coefficients in current block for the consideration of computational complexity and algorithm efficiency. We denote the maximum magnitude as L_{max}, then the *primary* context for current (*Level*, *Run*) pair can be defined as

$$C_{P(L,R)}(L^{i-1}) = L_{max}. \tag{6}$$

In essence, the variable L_{max} acts as a context quantizer that maps all histories of the current block up to the current pair to an integer value. L_{max} is initialized with the value of zero at the beginning of a DCT block, and will be updated on the fly during sequential coding of the (*Level*, *Run*) pairs. By conditioning the maximum magnitudes of all previously coded coefficients, the entropy coding using the estimated conditional probability $p((L_i, R_i)/L_{max})$ improves the coding efficiency over just using $p((L_i, R_i))$. The probability distribution of *absLevel* according to L_{max} is represented in Fig. 6, from which we can observe that the probability distribution of *absLevel* is variable according to L_{max}. Note that the case of *absLevel* equal to zero represents *EOB*, which indicates the end of the current block coding. This is the same as in C2DVLC.

In image or video coding, the occurrences of large amount of L_{max} will result in a problem that the dynamic range of context variable L_{max} can still be too large which will increase the time and space complexity. To tackle this problem, we need a way of merging different contexts in which $p((L_i, R_i)/L_{max})$ are close to reduce the number of contexts for conditional entropy coding. Thus, we quantize L_{max} into M levels to form *primary* contexts. In practice, $M = 5$ is found to be sufficient. Denote the L_{max} quantizer by Q, i.e., Q: $L_{max} \rightarrow \{0, 1, 2, 3, 4\}$. The quantization criterion is to minimize the conditional entropy of the (*Level*, *Run*) pairs. In an off-line design process, we get a set of ((*absLevel*, *Run*), L_{max}) instances from a training set, and use the standard dynamic programming technique to choose $0 = q_0 < q_1 < q_2 < q_3 < q_4 < q_5 = \infty$ to partition L_{max} into M ranges so that the needed average code length

$$\begin{aligned}&\sum_i \Bigg\{ p(q_i \leqslant L_{max} < q_{i+1}) \sum_{(L,R)} p((L,R)|q_i \leqslant L_{max} < q_{i+1}) \\ &\quad \times \log(p((L,R)|q_i \leqslant L_{max} < q_{i+1}))\Bigg\} \\ &= -\sum_i \sum_{(L,R)} p(L,R) \log p((L,R)|q_i \leqslant L_{max} < q_{i+1})\end{aligned} \tag{7}$$

for coding these pairs is minimized. This quantizer, whose parameters are:

$$q_1 = 1,\ q_2 = 2,\ q_3 = 3,\ q_4 = 5, \tag{8}$$

works almost as well as the optimal individual-image dependent L_{max} quantizer. The quantization function can also be defined as follows:

$$\chi_{(L_{\max})} = \begin{cases} L_{\max}, & L_{\max} \in [0,\ 2] \\ 3, & L_{\max} \in [3,\ 4] \\ 4, & \text{otherwise} \end{cases}. \tag{9}$$

And the *primary* context index $C_{P(L,\ R)}(L_{i-1})$ equals to $\chi_{(Lmax)}$. This 5-Level quantizer for the maximum magnitudes of previously coded coefficients actually can help small images to generate enough samples for context modeling to learn $p((L_i, R_i)/L_{max})$ quickly in adaptive entropy coding. Meanwhile, it can also save a lot of memory during entropy coding. In the previous example, the values of L_{max} and *primary* context index are listed in Table 4.

Table 4
Updating of context variable L_{max} of the example (*Level*, *Run*) sequence.

(*Level*, *Run*)	(−1,2)	(−2,1)	(3,0)	(−2,0)	(9,0)	(0,0)
L_{max}	0	1	2	3	3	9
Primary context index	0	1	2	3	3	4

4.3.2. Secondary context

Under each primary context, seven nested secondary contexts are used to code the bin value corresponding to binary decisions of *Level* and *Run* values. The seven secondary contexts are defined as shown in Table 5. The first three contexts in Table 5 are designed according to the bin index for coding bin values for *absLevel* while the following four contexts are based on both bin index and current *Level* for *Run* values.

For the bins of *absLevel*, the *secondary* context index is defined as

$$C_{S(L)}(j) = (j \leqslant 1)?\quad j : 2, \tag{10}$$

where the variable j is the bin index of *absLevel*. Since the binarized first bin value equals to 1 only when it is EOB, That is to say, the first bin value of *absLevel* (the corresponding bin index j is 0) carries the *EOB* information of the current DCT block, therefore it is reasonable to design one separated context for the first bin of *absLevel*. Furthermore, as shown in Fig. 6, the *absLevel* always has the largest possibility to equal to 1 at different frequency subbands. Hence we should emphasize this case, which is corresponding to the special context designed for the second bin of *absLevel*. For space efficiency, we use only one context to encode all remaining bins of *absLevel*. It should be pointed out that for the first to be coded *absLevel* which cannot be zero, the first binarized bin need not to be coded. Therefore, when L_{max} equals to be zero, the *secondary* context index $C_{S(L)}(j) = 0$ can never be used.

When coding a *Run*, in addition to L_{max} and the bin index information as that of *absLevel* does, the current coded *Level* information of the current (*Level*, *Run*) pair is also taken into consideration for *secondary* context modeling. This is because *Level* is coded first followed by the *Run*. The context index $C_{S(R)}(j, L)$ is determined by

$$C_{S(R)}(j,L) = \begin{cases} 3 + (j == 0?\, 0 : 1), & \text{if } (\text{abs}(L) == 1) \\ 5 + (j == 0?\, 0 : 1), & \text{otherwise} \end{cases}. \tag{11}$$

Fig. 7 depicts the probability distributions of *Run* values according to diverse *absLevel* under different *Primary* context index. In Fig. 7, the probability distributions of *Run* values are variable even with the same *Primary* context index. Hence, only use one condition of previously coded non-zero coefficients, i.e. L_{max}, cannot capture *Run*'s behavior perfectly. As mentioned in Section 2.1, there exists some correlation between the magnitudes of *Level* and *Run* in one (L_i, R_i) pair. The larger the magnitude of *Level* is, the higher the probability of smaller *Run* value will be. Therefore, for a given *Run*, the statistical distribution of bins needs to be determined by also considering the corresponding current coded *Level*, which can lead to better modeling and estimations of *Run* values. Besides, one context for the first bin of *Run* is specially formed to distinguish from other bins. Extensive experiments demonstrate that more contexts to distinguish different bins of *Level* or *Run* will not improve the coding efficiency too much.

For the sign of *Level*, the statistical analysis reveals that the distribution of transformed coefficients is approximately symmetric with respect to zero, i.e., the sign bit averagely consumes one bit. Thus, the sign of *Level* is simply dumped (coded using probability 0.5 without any context modeling) with bypass coding.

Table 5
Updating of context variable L_{max} of the example (*Level*, *Run*) sequence.

Bin of *Level*/*Run*	*Secondary* context index
First bin of *absLevel* (i.e., the *EOB* symbol).	0
Second bin of *absLevel*, if exist.	1
Remaining bins of *absLevel*, if exist.	2
First bin of *Run* if *absLevel* = 1.	3
Remaining bins of *Run* when *absLevel* = 1, if exist.	4
First bin of *Run* when *absLevel* > 1.	5
Remaining bins of *Run* when *absLevel* > 1, if exist	6

4.3.3. Context weighting

It is well known that adaptive entropy coding can benefit from both the position and the magnitude of *Level*. However, the contexts introduced so far are based on the magnitude of *Level*. In order to further improve compression performance, another context variable *ReverseP*, the position of the current *Level* in the reverse scanning order is introduced in CBAC. The variable *ReverseP* is initialized to zero in the transformed block. Based on *ReverseP*, a so-called *accompanying* context $C_{A(L)}(ReverseP)$ is introduced. This context relies on local information of the symbol

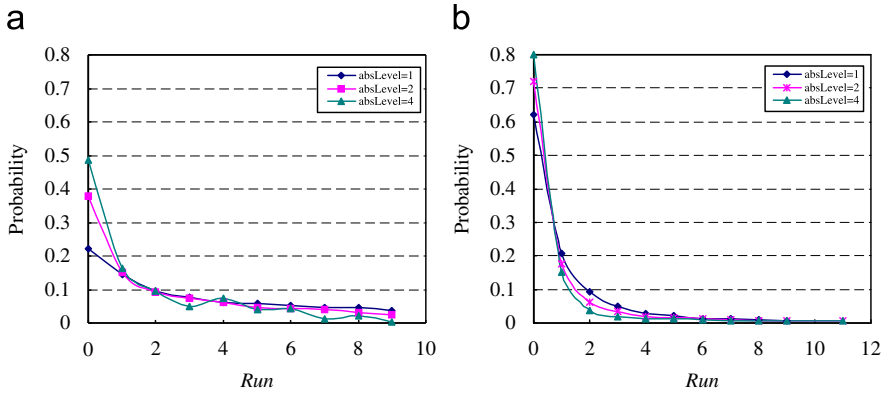


Fig. 7. Probability distribution of different *Run* values. (a) *Primary* context index equal to 0 and (b) *Primary* context index equal to 2.

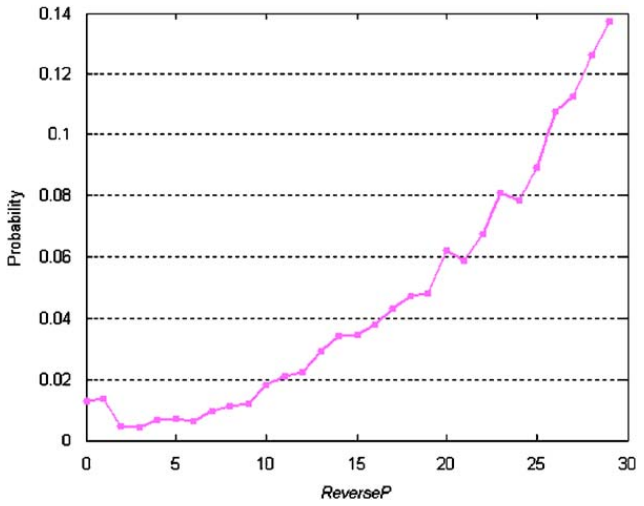


Fig. 8. Probabilities of the 1st bin of *absLevel* equal to one according to *ReverseP*.

which is the approximate frequency component. The probability distributions of *absLevel* can vary in terms of *ReverseP*. Fig. 8 represents the probability distribution of the first bin *of absLevel* equal to one (*EOB*) according to the local information. We can see that the probability of *EOB* is changing with the variable *ReverseP*. For an 8×8 block with at least one non-zero coefficients, the range of *ReverseP* is [0, 63], and it is uniformly quantized into 32 *accompanying* contexts, [0, 31]. The context index increments are determined as follows:

$$C_{A(L)}[ReverseP] = 16 \times (ReverseP \gg 5) + ((ReverseP \gg 1)\&0x0f). \quad (12)$$

In the binary arithmetic coding of the *Run* and *Level* values, each *accompanying* context created by *ReverseP* will be combined with the same seven *secondary* contexts as in the case of *primary* contexts created by L_{max}.

Now when coding each binary decision, we have two conditional probability estimates: one in the product context derived from L_{max} and the other from *ReverseP*. Then an interesting question is if we can make use of both position and *Level* without increasing the model cost, can we get a shorter code length? The answer is yes. The two kinds of contexts created above are defined as: $C_1 = L_{max}$ and $C_2 = ReverseP$. Let $p(x/c_1)$ and $p(x/c_2)$ be the estimated conditional probabilities and w be the weighting factor, then the weighted probability of the current DCT coefficient x is assigned as follows:

$$p(x|c_1 \cup c_2) = w \times p(x|c_1) + (1 - w) \times p(x|c_2). \quad (13)$$

Since $p(x|c_1)$ and $p(x|c_2)$ are probability measures on x, given c_1 and c_2, $p(x|c_1 \cup c_2)$, a weighted sum of $p(x|c_1)$ and $p(x|c_2)$, is also a probability measure on x. Thus, a weighted probability distribution of the two estimated distributions is used to drive the arithmetic coder. In our scheme, the equal weighting scheme is used ($w = 0.5$), which is found to produce better compression results.

The context weighting technique is effective when being applied to code the *EOB* symbol. The coding gain on other bins is usually less than 0.5%. So for low complexity we only use the context weighting technique on the *EOB* context, i.e., *secondary* context index equal to zero.

4.4. Binary arithmetic coding

In this section, we present the binary arithmetic coder on logarithm domain adopted in CBAC. Actually, the CBAC coding engine consists of two sub-engines, one for the 'regular' coding mode, which includes the utilization of adaptive contexts, and the other so-called 'bypass' coding engine for a fast encoding of symbols, for which an approximately uniform probability (the probabilities of symbol '0' and '1' are equal, i.e., 0.5) is assumed. The following presentation includes the basic parts of binary arithmetic coder: (1) interval subdivision, (2) renormalization process to keep finite precision during the whole coding process, (3) adaptively update probability estimation. For the detail information about the binary arithmetic coder, readers can refer to Ref. [14].

Binary arithmetic coding is based on the principle of recursive interval subdivision that involves the following elementary multiplication operation. Suppose that an estimate of the probability $p_{MPS} \in (0.5, 1)$ of the *most probable symbol* (*MPS*) is given and the given interval is represented by its lower bound denoted as L and its width (range) represented by R. Based on these settings, the given interval is subdivided into two subintervals: one interval of width

$$R_{MPS} = R \times p_{MPS} \quad (14)$$

which is associated with the *MPS*, and the dual interval of width $R_{LPS} = R - R_{MPS}$, which is assigned to the *least probable*

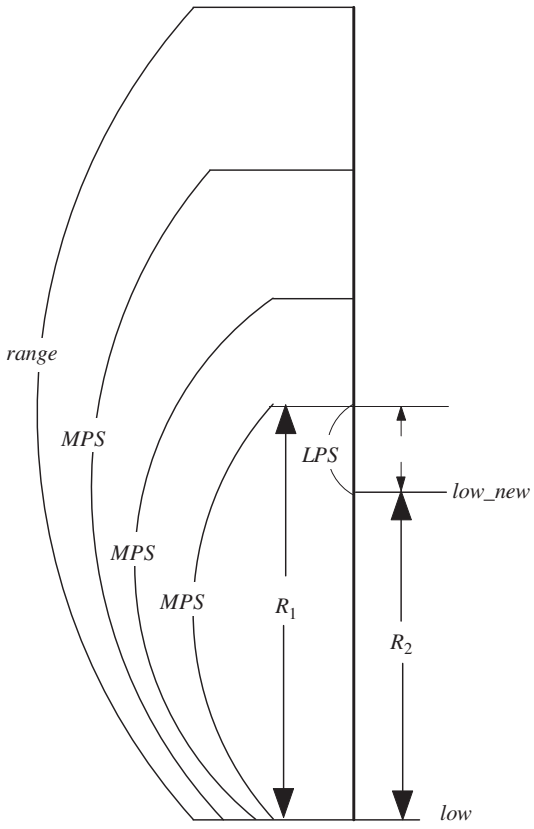


Fig. 9. One binary arithmetic coder cycle.

symbol (*LPS*) having a probability estimation of $1-p_{MPS}$. Fig. 9 depicts a complete cycle of coding process. The process is an iterative one which consists of consecutive *MPS* symbols and one *LPS* symbol. We keep 9 bit precision for range during whole coding process. In the binary arithmetic coder of CBAC, we substitute the multiplication in Eq. (14) with addition by using logarithm domain instead of original domain. When a *MPS* happens, the renewal of range is given as

$$LG\_R_{new} = LG\_R + LG\_p_{MPS}, \tag{15}$$

where LG_x indicates the logarithm value of variable x and R_{new} is the new range after encoding one *MPS*. For the case of meeting one *LPS*, we denote the two *MPS* range before and after encoding the *LPS* as R_1 and R_2 as shown in Fig. 9. Then, the range after the whole coding cycle in original domain should be

$$R_{LPS} = R_1 - R_2. \tag{16}$$

And the new lower bound of current range equals to the addition of *low* and R_2. Since R_1 and R_2 are both calculated on the logarithm domain, we have to get the value of R_1 and R_2 from LG_R_1 and LG_R_2, then

$$R_1 = 2^{LG\_R_1} = 2^{-s_1+t_1} \approx 2^{-s_1} \times (1 + t_1 - \Delta_1), \tag{17}$$

and

$$R_2 = 2^{LG\_R_2} = 2^{-s_2+t_2} \approx 2^{-s_2} \times (1 + t_2 - \Delta_2). \tag{18}$$

From (16)–(18), we can get the following, ignoring the approximation error Δ_1 and Δ_2:

$$R_{LPS} = 2^{-s_2} \times t_3, \tag{19}$$

and

$$t_3 \approx \begin{cases} t_1 - t_2 & \text{if } (s_2 == s_1) \\ (t1 \ll 1) - t_2 & \text{if } (s_2 = s_1 - 1) \end{cases}. \tag{20}$$

After the value of R_{LPS} is obtained, the renewed lower bound is updated. Then the renormalization process is carried out to guarantee that the most significant bit of the updated *range* value is always '1'. Until now, one coding cycle is finished. After one bin is encoded by arithmetic coder, the estimated probability of the chosen context should also be updated. Actually, in CBAC, the probability of each context model is initialized to be 0.5 for both *MPS* and *LPS* at the start of coding. With the coding of some bins, the adaptive probability estimation of *MPS* on logarithm domain is performed. The probability estimation is fulfilled using only additions/subtractions and shifts as in the following formulas:

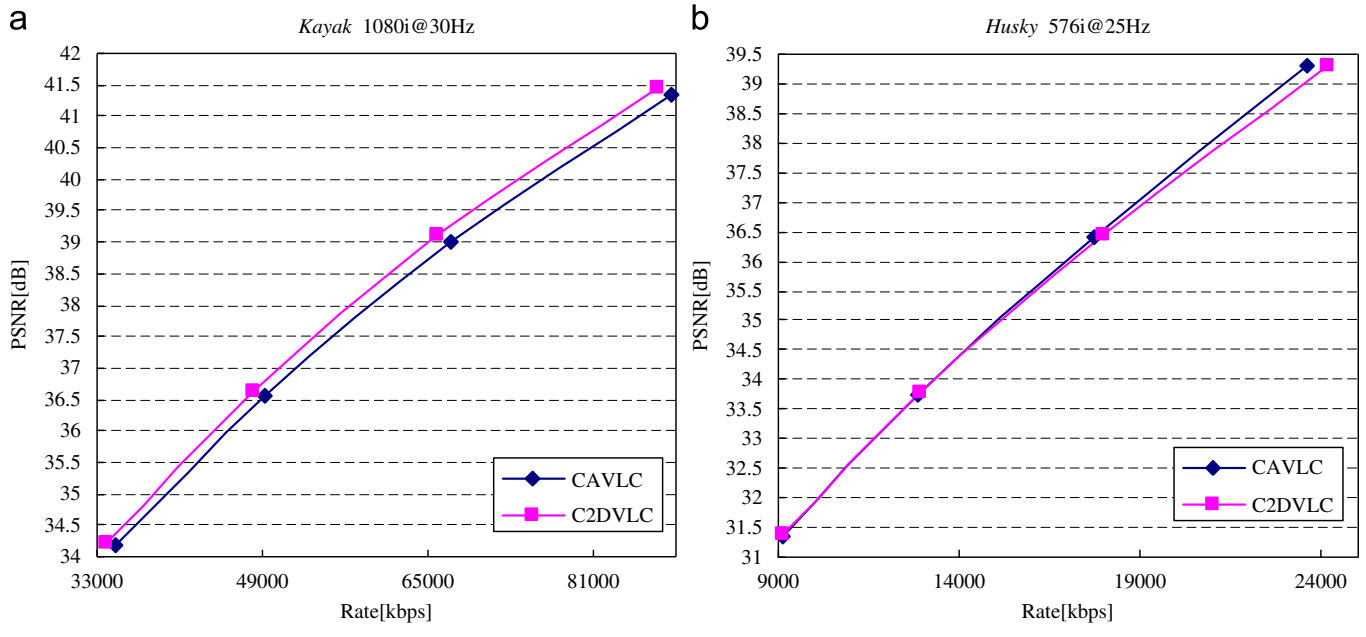
$$\begin{cases} LG\_P_{MPS} \leftarrow LG\_P_{MPS} + LG\_f & \text{if } (LPS \text{ happens}) \\ LG\_P_{MPS} \leftarrow LG\_P_{MPS} - (LG\_P_{MPS} \gg cw) & \text{if } (MPS \text{ happens})' \end{cases} \tag{21}$$

where f is equal to $(1-2^{-cw})$. Here, cw is the size of sliding widow to control the speed of probability adaptation. The smaller cw is, the faster the probability adaptation will be.

In summary, the arithmetic coder in CBAC replaces the traditional multiplications for range update and probability estimation update with additions by combining original domain and logarithmic domain. In order to keep the cost of alternation between the two domains low, the approximation is employed. Moreover, the renormalization takes place only when one *LPS* happens so that much time can be saved for renormalization process.

5. Experimental results

This section reports the coding performance of C2DVLC and CBAC. In addition, the coding gain of context weighting technique is also shown in Subsection 5.2. The test sequences include two typical progressive HD



C2DVLC

Fig. 10. Rate-distortion curves of C2DVLC and CAVLC at (a) *Kayak* and (b) *Husky* videos.

Table 6
Average Coding efficiency gain of C2D-VLC compared with CAVLC.

Sequences	Kayak	Mobile calendar	Husky	Basket ball	Football	Harbour	Night	Average
ΔPSNR (dB)	0.25	0.02	−0.08	−0.08	0.19	0.21	0.08	0.08
ΔRate (%)	−3.2	−0.09	1.0	1.3	−3.8	−5.0	−2.3	−1.7

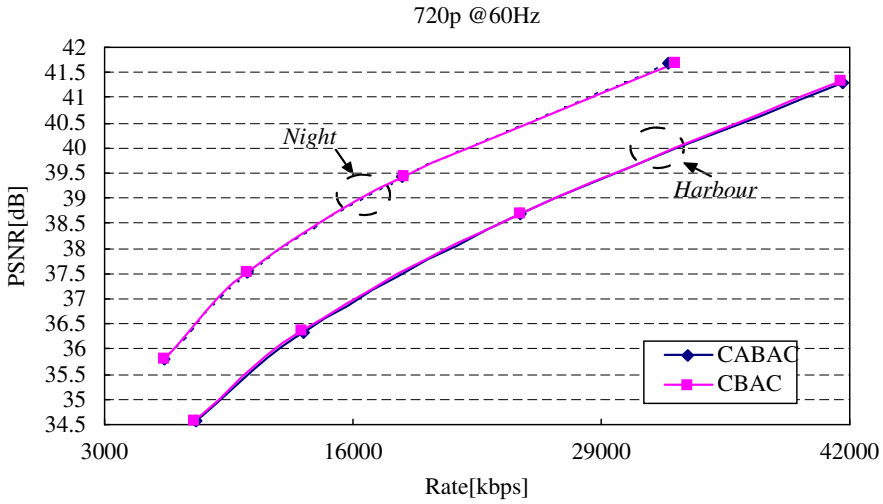


Fig. 11. Rate-distortion curves of CBAC and CABAC at '*Harbour*' and '*Night*' sequences.

Table 7
Average coding gain of CBAC compared with CABAC.

Sequences	Kayak	Mobile calendar	Husky	Basketball	Football	Average
(*a*) *At middle bit-rates*						
ΔPSNR (dB)	−0.12	−0.05	0.01	0.06	0.12	0.005
ΔRate (%)	1.7	−0.05	0.0	−0.9	−2.5	−0.2
(*b*) *At high bit-rates*						
ΔPSNR (dB)	0.22	0.16	0.20	0.20	0.17	0.19
ΔRate (%)	−2.1	−2.4	−1.2	−1.8	−2.1	−1.7

sequences, *Harbour* and *Night*, in 1280×720@60 Hz and two interlaced HD sequences, *Kayak* and *Fireworks*, in 1920×1088@30 Hz. Besides, four interlaced SD sequences, *Football* in 704×480, *Husky* in 704×576, *Basketball* and *Mobilecalendar* in 720×576 all at 25 Hz. Rm62c platform [9], which is developed by AVS working group as AVS Part-2 reference software, is used. In all simulations, rate-distortion optimization was switched on. One hundred frames of each test sequences are coded. The coding performance was evaluated using the IBBPBBP frame structure. Here, the motion search was conducted in a range of $[-32 \ldots 32] \times [-32 \ldots 32]$ samples for two reference frames. All encoding mode decisions including the motion search, the macroblock mode decision, and for interlaced sequences picture-based frame/field decision were performed with rate-distortion optimization on. For all the simulations, bit-rates were adjusted by using fixed values of QPs. The value of QP for B pictures was set to be the same with those for I and P pictures.

Table 8
Bit savings provided by context weighting.

Sequences	Without weighting (bpf)	Weighting (bpf)	Bit savings (%)
Bus	4996	4503	10.62
Foreman	2824	2589	9.5
Mobile	7536	6369	15.86

5.1. Coding performance of C2DVLC

C2DVLC is compared to CAVLC in H.264/AVC in the testing. CAVLC is realized into Rm62c platform, and encode the 8×8 DCT coefficients by splitting an 8×8 block into four 4×4 blocks which is the same as that in H.264/AVC high profile. In this simulation, Quantization values are set as 24, 28, 32, and 36. For these QPs, the

Table 9
Average coding efficiency gain of CBAC compared with C2D-VLC.

Sequences	Kayak	Mobile calendar	Husky	Basket ball	Football	Harbour	Night	Average
ΔPSNR (dB)	0.59	0.42	0.54	0.48	0.51	0.33	0.25	0.45
ΔRate (%)	−7.5	−7.1	−6.4	−7.5	−10.1	−7.5	−6.7	−7.5

PSNR is in a range of 30–40 dB. Fig. 10 depicts the rate-distortion curves of the luminance component for '*kayak*' and '*Husky*' sequences. Furthermore, we employed Bjontegarrd delta PSNR as described in Ref. [1] to calculate the average coding gains. The coding gains of C2DVLC relative to CAVLC are shown in Table 6. The average PSNR gain is 0.08 dB. It can be observed that C2DVLC can provide comparable rate-distortion performance for all tested sequences with CAVLC. Especially, for '*kayak*' sequence at 1920×1088@30 Hz, the average PSNR gain is up to 0.25 dB.

5.2. Coding performance of CBAC

In this subsection, CBAC is compared to CABAC in H.264/AVC to evaluate its coding performance. CABAC, including its context modeling designed for 8×8 DCT coefficients and the adaptive binary arithmetic coding, is transplanted from JM10.2 [10] to Rm62c. JM10.2 is the reference software for H.264/AVC which is released by JVT. In CABAC, the fixed context initialization method is applied. While in CBAC, the initial probabilities of 0 and 1 are set to be equal in each context, which are 0.5.

Fig. 11 depicts the rate-distortion curves of CBAC and CABAC at '*Harbour*' and '*Night*' sequences in 720p@60 Hz. From this figure, we can see that CBAC achieves the similar coding efficiency as CABAC. Table 7 (a) shows the average coding gains of CBAC relative to CABAC under QP 24, 28, 32, and 36. For these QPs, the PSNR is in a range of 30–40 dB, corresponding to middle bit-rates. The results listed in Table 7(b) are obtained under four smaller QPs, 10, 12, 14, and 16, which correspond to the PSNR values in the range from 40 to 50 dB. From these two tables, it can be easily observed that CBAC can achieve comparable coding efficiency with CABAC. Overall, the average PSNR gains of 0.005 and 0.19 dB can be obtained by CBAC at middle to high bit-rates separately. With bit-rate increased, the coding gain becomes larger.

To verify the efficiency of the context weighting technique, three common testing sequences with CIF format, *Bus*, *Foreman*, and *Mobile*, are used. Table 8 lists the bit savings of the first bin of *absLevel*. In the testing, QP is set to be 27 with rate-distortion optimization off. The average bits used for coding the first bin of *absLevel* per frame without the context weighting technique is shown in the second column while the third column shows the average bits per frame (bpf) using the context weighting technique. In the last column, the average bit savings is listed. It can be observed the context weighting technique can save up to 15.86% coding bits.

5.3. CBAC compared to C2DVLC

This subsection demonstrates that CBAC can significantly outperform C2DVLC in terms of coding efficiency. Table 9 lists the coding gains (both in PSNR increase and bit-rate savings) of CBAC compared with C2DVLC. It can be observed that CBAC shows superior coding performance in comparison to C2DVLC. For the tested sequences, average PSNR gains from 0.25 to 0.59 dB, equally about 6.4–10.1% bit-rate savings, are obtained.

6. Conclusions

This paper presents the two context-based entropy coding schemes, C2DVLC and CBAC, adopted in AVS Part-2 Jizhun profile and Jiaqiang profile respectively. C2DVLC jointly encodes *Level* and *Run* and utilizes multiple 2D-VLC tables to exploit the statistical features of DCT coefficients for higher coding efficiency. E–G codes are applied in C2DVLC to code (*Level*, *Run*) pairs for low storage requirement. CBAC improves the coding efficiency with acceptable extra computational complexity. The further coding efficiency comes from three aspects. Firstly, the novel context quantization method allows characterization of high-order Markov processes without suffering from context dilution problem. Secondly, the context weighting technique can merge multiple context models into one. Thirdly, the binary arithmetic coder also plays an important role for efficiency improvement. Moreover, CBAC is designed to be compatible to C2DVLC in coding elements which simplifies the implementations. The experimental results demonstrate that both C2DVLC and CBAC can achieve comparable or even slightly higher coding performance when compared to CAVLC and CABAC in H.264/AVC, respectively.

Acknowledgements

This work was supported in part by the National Science Foundation (60833013 and 60803068) and Major State Basic Research Development Program of China (973 Program, 2009CB320903).

References

[1] G. Bjontegaard, Calculation of average PSNR differences between RD-Curves, In: Doc. ITU-T VCEG (VCEG-M33), Austin, Texas, USA, March 2001.
[2] G. Bjontegaard, K. Lillevold, Context-adaptive VLC coding of coefficients, In: Doc. Joint Video Team of ISO/IEC MPEG & ITU-T VCEG (JVT-C028), Fairfax, VA, May 2002.

[3] J. Dong, J. Lou, L. Yu, Improved entropy coding method, In: Doc. AVS Working Group (M1214), Beijing, China, December 2003.
[4] L. Fan, S. Ma, F. Wu, Overview of AVS video standard, In: Proceedings of the ICME04, Taiwan, China, 2004, pp. 423–426.
[5] ITU-T, Advanced video coding for generic audiovisual services, ITU-T Recommendation H.264, version 1, 2003.
[6] ITU-T and ISO/IEC JTC1, Generic coding of moving pictures and associated audio information–Part 2: Video, ITU-T Recommendation H.262–ISO/IEC 13818-2 (MPEG-2), November 1994.
[7] E.Y. Lam, J.W. Goodman, A mathematical analysis of the DCT coefficient distributions of images, IEEE Transactions on Image Process 9 (10) (October 2000) 1661–1666.
[8] D. Marpe, H. Schwarz, T. Wiegand, Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard, IEEE Transactions on Circuits and Systems for Video Technology 13 (7) (July 2003) 620–636.
[9] [Online] Available: 〈http://159.22.42.57/incoming/dropbox/video_software/Rm62c.zip〉.
[10] [Online] Available: 〈http://iphome.hhi.de/suehring/tml/download/jm10.2.zip〉.
[11] J. Rissanen, A universal data compression system, IEEE Transactions on Information Theory 29 (5) (September 1983) 656–664.
[12] J. Teuhola, A compression method for clustered bit-vectors, Information Processing Letters 7 (6) (October 1978) 308–311.
[13] Qiang Wang, Debin Zhao, Wen Gao, Context-based 2D VLC entropy coder in AVS video coding standard, Journal of Computer Science and Technology 21 (3) (May 2006) 315–322.
[14] Wei Yu, Ping Yang, Yun He, Arithmetic codec on logarithm domain, In: Proc. Picture Coding Symposium 2006, Beijing, China, April 2006.
[15] M.J. Weinberger, J. Rissanen, R.B. Arps, Applications of universal context modeling to lossless compression of gray-scale images, IEEE Transactions on Image Process 5 (4) (April 1996) 575–586.
[16] X. Wu, Lossless compression of continuous-tone images via context selection, quantization, and modeling, IEEE Transactions on Image Process 6 (5) (May 1997) 656–664.
[17] M. Xu, X. Wu, P. Franti, Context quantization by kernel fisher discriminant, IEEE Transactions on Image Process 15 (1) (January 2006) 169–177.
[18] Cixun Zhang, Lu Yu, Jian Lou, Wai-kuen Cham, Jie Dong, The technique of prescaled integer transform: concept, design and applications, IEEE Transactions on Circuits and Systems for Video Technology 18 (1) (January 2008) 84–97.
[19] Xiangyang Ji, Debin Zhao, Feng Wu, Yang Lu, Wen Gao, B-picture coding in AVS video compression standard, Signal Processing: Image Communication 23 (January 2008) 31–41.