H.264 / MPEG-4 Part 10 White Paper

Variable-Length Coding

1. Introduction

The Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG are finalising a new standard for the coding (compression) of natural video images. The new standard [1] will be known as H.264 and also MPEG-4 Part 10, "Advanced Video Coding". The standard specifies two types of entropy coding: Context-based Adaptive Binary Arithmetic Coding (CABAC) and Variable-Length Coding (VLC). The Variable-Length Coding scheme, part of the Baseline Profile of H.264, is described in this document.

Please note that the H.264 draft standard is not yet finalised and so readers are encouraged to refer to the latest version of the standard.

2. Coded elements

Parameters that require to be encoded and transmitted include the following (Table 2-1).

Table 2-1 Parameters to be encoded

Parameters	Description
Sequence-, picture- and slice-layer syntax elements	
Macroblock type mb_type	Prediction method for each coded macroblock
Coded block pattern	Indicates which blocks within a macroblock contain coded coefficients
Quantizer parameter	Transmitted as a delta value from the previous value of QP
Reference frame index	Identify reference frame(s) for inter prediction
Motion vector	Transmitted as a difference (mvd) from predicted motion vector
Residual data	Coefficient data for each 4x4 or 2x2 block

Above the slice layer, syntax elements are encoded as fixed- or variable-length binary codes. At the slice layer and below, elements are coded using either variable-length codes (VLCs) [2] or context-adaptive arithmetic coding (CABAC) [3] depending on the entropy encoding mode.

3. Variable length coding (VLC)

When entropy_coding_mode is set to 0, residual block data is coded using a context-adaptive variable length coding (CAVLC) scheme and other variable-length coded units are coded using Exp-Golomb codes.

3.1 Exp-Golomb entropy coding

Exp-Golomb codes (Exponential Golomb codes) are variable length codes with a regular construction. Table 3-1 lists the first 9 codewords; it is clear from this table that the codewords progress in a logical order. Each codeword is constructed as follows:

[M zeros][1][INFO]

where INFO is an M-bit field carrying information. The first codeword has no leading zero or trailing INFO; codewords 1 and 2 have a single-bit INFO field; codewords 3-6 have a 2-bit INFO field; and so on. The length of each codeword is (2M+1) bits.

Each Exp-Golomb codeword can be constructed by the encoder based on its index **code_num**:

$M = ?\log_2(code\_num+1)?$
$INFO = code\_num + 1 - 2^M$

A codeword can be decoded as follows:
1. Read in M leading zeros followed by 1.
2. Read M-bit INFO field.
3. code_num = 2^M + INFO – 1

(For codeword 0, INFO and M are zero).

Table 3-1 Exp-Golomb codewords

code_num	Codeword
0	1
1	010
2	011
3	00100
4	00101
5	00110
6	00111
7	0001000
8	0001001
…	…

A parameter **v** to be encoded is mapped to code_num in one of 3 ways:

ue(v) : Unsigned direct mapping, code_num = v. Used for macroblock type, reference frame index and others.

se(v) : Signed mapping, used for motion vector difference, delta QP and others. v is mapped to code_num as follows (Table 3-2).
code_num = 2|v| (v < 0)
code_num = 2|v| - 1 (v ? 0)

Table 3-2 Signed mapping se(v)

v	code_num
0	0
1	1
-1	2
2	3
-2	4
3	5
…	…

me(v) : Mapped symbols; parameter v is mapped to code_num according to a table specified in the standard. This mapping is used for the coded_block_pattern parameter. Table 3-3 lists a small part of the table for Inter predicted macroblocks: coded_block_pattern indicates which 8x8 blocks in a macroblock contain non-zero coefficients.

Table 3-3 Part of coded_block_pattern table

coded_block_pattern (Inter prediction)	code_num
0 (no non-zero blocks)	0
16 (chroma DC block non-zero)	1
1 (top-left 8x8 luma block non-zero)	2
2 (top-right 8x8 luma block non-zero)	3

4 (lower-left 8x8 luma block non-zero)	4
8 (lower-right 8x8 luma block non-zero)	5
32 (chroma DC and AC blocks non-zero)	6
3 (top-left and top-right 8x8 luma blocks non-zero)	7
…	…

Each of these mappings (ue, se and me) is designed to produce short codewords for frequently-occurring values and longer codewords for less common parameter values. For example, macroblock type Pred_L0_16x16 (i.e. 16x16 prediction from a previous picture) is assigned code_num 0 because it occurs frequently whereas macroblock type Pred_8x8 (8x8 prediction from a previous picture) is assigned code_num 3 because it occurs less frequently. The commonly-occurring motion vector difference (MVD) value of 0 maps to code_num 0 whereas the less-common MVD = -3 maps to code_num 6.

3.2 Context-based adaptive variable length coding (CAVLC)

This is the method used to encode residual, zig-zag ordered 4x4 (and 2x2) blocks of transform coefficients. CAVLC is designed to take advantage of several characteristics of quantized 4x4 blocks:

1. After prediction, transformation and quantization, blocks are typically sparse (containing mostly zeros). CAVLC uses run-level coding to compactly represent strings of zeros.
2. The highest non-zero coefficients after the zig-zag scan are often sequences of +/-1. CAVLC signals the number of high-frequency +/-1 coefficients ("Trailing 1s" or "T1s") in a compact way.
3. The number of non-zero coefficients in neighbouring blocks is correlated. The number of coefficients is encoded using a look-up table; the choice of look-up table depends on the number of non-zero coefficients in neighbouring blocks.
4. The level (magnitude) of non-zero coefficients tends to be higher at the start of the reordered array (near the DC coefficient) and lower towards the higher frequencies. CAVLC takes advantage of this by adapting the choice of VLC look-up table for the "level" parameter depending on recently-coded level magnitudes.

CAVLC encoding of a block of transform coefficients proceeds as follows.

1. Encode the number of coefficients and trailing ones (coeff_token).
The first VLC, coeff_token, encodes both the total number of non-zero coefficients (TotalCoeffs) and the number of trailing +/-1 values (T1). TotalCoeffs can be anything from 0 (no coefficients in the 4x4 block) [1] to 16 (16 non-zero coefficients). T1 can be anything from 0 to 3; if there are more than 3 trailing +/-1s, only the last 3 are treated as "special cases" and any others are coded as normal coefficients.

There are 4 choices of look-up table to use for encoding coeff_token, described as Num-VLC0, Num-VLC1, Num-VLC2 and Num-FLC (3 variable-length code tables and a fixed-length code). The choice of table depends on the number of non-zero coefficients in upper and left-hand previously coded blocks N_u and N_L. A parameter N is calculated as follows:

If blocks U and L are available (i.e. in the same coded slice), $N = (N_u + N_L)/2$
If only block U is available, $N=N_U$; if only block L is available, $N=N_L$; if neither is available, N=0.

Figure 3-1 Neighbouring blocks N_U and N_L

[1] Note: coded_block_pattern (described earlier) indicates which **8x8** blocks in the macroblock contain non-zero coefficients; however, within a coded 8x8 block, there may be **4x4** sub-blocks that do not contain any coefficients, hence TotalCoeff may be 0 in any 4x4 sub-block. In fact, this value of TotalCoeff occurs most often and is assigned the shortest VLC.

N selects the look-up table (Table 3-4) and in this way the choice of VLC adapts depending on the number of coded coefficients in neighbouring blocks (**context adaptive**). Num-VLC0 is "biased" towards small numbers of coefficients; low values of TotalCoeffs (0 and 1) are assigned particularly short codes and high values of TotalCoeff particularly long codes. Num-VLC1 is biased towards medium numbers of coefficients (TotalCoeff values around 2-4 are assigned relatively short codes), Num-VLC2 is biased towards higher numbers of coefficients and FLC assigns a fixed 6-bit code to every value of TotalCoeff.

Table 3-4 Choice of look-up table for coeff_token

N	Table for coeff_token
0, 1	Num-VLC0
2, 3	Num-VLC1
4, 5, 6, 7	Num-VLC2
8 or above	FLC

2. Encode the sign of each T1.
For each T1 (trailing +/-1) signalled by coeff_token, a single bit encodes the sign (0=+, 1=-). These are encoded **in reverse order**, starting with the highest-frequency T1.

3. Encode the levels of the remaining non-zero coefficients.
The **level** (sign and magnitude) of each remaining non-zero coefficient in the block is encoded **in reverse order**, starting with the highest frequency and working back towards the DC coefficient. The choice of VLC table to encode each level adapts depending on the magnitude of each successive coded level (**context adaptive**). There are 7 VLC tables to choose from, Level_VLC0 to Level_VLC6. Level_VLC0 is biased towards lower magnitudes; Level_VLC1 is biased towards slightly higher magnitudes and so on. The choice of table is adapted in the following way:

(a) Initialise the table to Level_VLC0 (unless there are more than 10 non-zero coefficients and less than 3 trailing ones, in which case start with Level_VLC1).
(b) Encode the highest-frequency non zero coefficient.
(c) If the magnitude of this coefficient is larger than a pre-defined threshold, move up to the next VLC table.

In this way, the choice of level is matched to the magnitude of the recently-encoded coefficients. The thresholds are listed in Table 3-5; the first threshold is zero which means that the table is always incremented after the first coefficient level has been encoded.

Table 3-5 Thresholds for determining whether to increment Level table number

Current VLC table	Threshold to increment table
VLC0	0
VLC1	3
VLC2	6
VLC3	12
VLC4	24
VLC5	48
VLC6	N/A (highest table)

4. Encode the total number of zeros before the last coefficient.
TotalZeros is the sum of all zeros preceding the highest non-zero coefficient in the reordered array. This is coded with a VLC. The reason for sending a separate VLC to indicate TotalZeros is that many blocks contain a number of non-zero coefficients at the start of the array and (as will be seen later) this approach means that zero-runs at the start of the array need not be encoded.

5. Encode each run of zeros.
The number of zeros preceding each non-zero coefficient (run_before) is encoded **in reverse order**. A run_before parameter is encoded for each non-zero coefficient, starting with the highest frequency, with two exceptions:
(a) If there are no more zeros left to encode (i.e. ? [run_before] = TotalZeros), it is not necessary to encode any more run_before values.
(b) It is not necessary to encode run_before for the final (lowest frequency) non-zero coefficient.

The VLC for each run of zeros is chosen depending on (a) the number of zeros that have not yet been encoded (ZerosLeft) and (b) run_before. For example, if there are only 2 zeros left to encode, run_before can only take 3 values (0,1 or 2) and so the VLC need not be more than 2 bits long; if there are 6 zeros still to encode then run_before can take 7 values (0 to 6) and the VLC table needs to be correspondingly larger.

CAVLC Examples

In all the following examples, we assume that table Num-VLC0 is used to encode coeff_token.

Example 1

4x4 block:

0	3	-1	0
0	-1	1	0
1	0	0	0
0	0	0	0

Reordered block:
0,3,0,1,-1,-1,0,1,0…

TotalCoeffs = 5 (indexed from highest frequency [4] to lowest frequency [0])
TotalZeros = 3
T1s = 3 (in fact there are 4 trailing ones but only 3 can be encoded as a “special case”)

Encoding:

Element	Value	Code
coeff_token	TotalCoeffs=5, T1s=3	0000100
T1 sign (4)	+	0
T1 sign (3)	-	1
T1 sign (2)	-	1
Level (1)	+1 (use Level_VLC0)	1
Level (0)	+3 (use Level_VLC1)	0010
TotalZeros	3	111
run_before(4)	ZerosLeft=3; run_before=1	10
run_before(3)	ZerosLeft=2; run_before=0	1
run_before(2)	ZerosLeft=2; run_before=0	1
run_before(1)	ZerosLeft=2; run_before=1	01
run_before(0)	ZerosLeft=1; run_before=1	No code required; last coefficient.

The transmitted bitstream for this block is 000010001110010111101101 .

Decoding:

The output array is "built up" from the decoded values as shown below. Values added to the output array at each stage are underlined.

Code	Element	Value	Output array
0000100	coeff_token	TotalCoeffs=5, T1s=3	Empty
0	T1 sign	+	1
1	T1 sign	-	-1, 1
1	T1 sign	-	-1, -1, 1
1	Level	+1	1, -1, -1, 1
0010	Level	+3	3, 1, -1, -1, 1
111	TotalZeros	3	3, 1, -1, -1, 1
10	run_before	1	3, 1, -1, -1, 0, 1
1	run_before	0	3, 1, -1, -1, 0, 1
1	run_before	0	3, 1, -1, -1, 0, 1
01	run_before	1	3, 0, 1, -1, -1, 0, 1

The decoder has inserted two zeros; however, TotalZeros is equal to 3 and so another 1 zero is inserted before the lowest coefficient, making the final output array:
0, 3, 0, 1, -1, -1, 0, 1

Example 2

4x4 block:

-2	4	0	-1
3	0	0	0
-3	0	0	0
0	0	0	0

Reordered block:
-2, 4, 3, -3, 0, 0, -1, …

TotalCoeffs = 5 (indexed from highest frequency [4] to lowest frequency [0])
TotalZeros = 2
T1s = 1

Encoding:

Element	Value	Code
coeff_token	TotalCoeffs=5, T1s=1	0000000110
T1 sign (4)	-	1
Level (3)	Sent as –2 (**see note 1**) (use Level_VLC0)	0001
Level (2)	3 (use Level_VLC1)	0010
Level (1)	4 (use Level_VLC1)	00010
Level (0)	-2 (use Level_VLC2)	111
TotalZeros	2	0011
run_before(4)	ZerosLeft=2; run_before=2	00
run_before(3..0)	0	No code required

The transmitted bitstream for this block is 000000011010001001000010111001100.

Note 1: Level (3), with a value of -3, is encoded as a special case. If there are less than 3 T1s, then the first **non**-T1 level will **not** have a value of +/-1 (otherwise it would have been encoded as a T1). To

save bits, this level is incremented if negative (decremented if positive) so that +/-2 maps to +/-1, +/-3 maps to +/-2, and so on. In this way, shorter VLCs are used.

Note 2: After encoding level (3), the level_VLC table is incremented because the magnitude of this level is greater than the first threshold (which is 0). After encoding level (1), with a magnitude of 4, the table number is incremented again because level (1) is greater than the second threshold (which is 3). Note that the final level (-2) uses a different code from the first encoded level (also –2).

Decoding:

Code	Element	Value	Output array
0000000110	coeff_token	TotalCoeffs=5, T1s=1	Empty
1	T1 sign	-	-1
0001	Level	-2 decoded as -3	-3, -1
0010	Level	+3	+3, -3, -1
00010	Level	+4	+4, 3, -3, -1
111	Level	-2	-2, 4, 3, -3, -1
0011	TotalZeros	2	-2, 4, 3, -3, -1
00	run_before	2	-2, 4, 3, -3, 0, 0, -1

All zeros have now been decoded and so the output array is:
-2, 4, 3, -3, 0, 0, -1

(This example illustrates how bits are saved by encoding TotalZeros: only a single run needs to be coded even though there are 5 non-zero coefficients).

4. References

1 ITU-T Rec. H.264 / ISO/IEC 11496-10, “Advanced Video Coding”, Final Committee Draft, Document JVT-E022, September 2002
2 JVT Document JVT-C028, G.Bjontegaard and K. Lillevold, “Context-Adaptive VLC Coding of Coefficients”, Fairfax, VA, May 2002
3 JVT Document JVT-L13, D. Marpe, G. Blattermann and T. Wiegand, “Adaptive Codes for H.26L”, Eibsee, January 2001