INTERNATIONAL ORGANISATION FOR STANDARDISATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC1/SC29/WG11
CODING OF MOVING PICTURES AND AUDIO

ISO/IEC JTC1/SC29/WG11
MPEG2001/M7512
July 2001

Source: **Gary Sullivan and Thomas Wiegand, on behalf of ITU-T VCEG (Q.6/SG16)**
Status: **Proposal**
Title: **Detailed Algorithm Technical Description for ITU-T VCEG Draft H.26L Algorithm in Response to Video and DCinema CfPs**
Author: **ITU-T VCEG (Q.6/16)**

ITU - Telecommunications Standardization Sector
STUDY GROUP 16
Video Coding Experts Group (VCEG)

Document VCEG-Nxx
Filename: VCEG-Nxx.doc
Generated: 7/10/01

Question: Q.6/SG16(VCEG)

Source: Gisle Bjontegaard (editor)
Telenor Broadband Services
P.O.Box 6914 St.Olavs plass
N-0130 Oslo, Norway
Tel: +47 23 13 83 81
Fax: +47 22 77 79 80
Email: gisle.bjontegaard@telenor.com

With TML-8 Changes edited by Thomas Wiegand

Title: H.26L Test Model Long Term Number 8 (TML-8) draft0.

Purpose: Test model

This d0 version of TML-8 includes the following changes from TML-6 (adopted in Austin and Porto Seguro):

- Run coding of coded MBs M29,M57
- 1/8 pixel prediction accuracy M45
- DQUANT on MB level M31
- Vectors pointing outside picture M34
- Dropping RDquant M72
- Drop of isolated chroma AC coefficients M32
- New sections7,8 for data partitioning and NAL(input not received yet) M52
- Some bug fixes
- New section for entropy coding including the description of UVLC and CABAC. (D. Marpe)
- Section “Motion Estimation and Mode Decision” with new High-complexity mode (H. Schwarz)

1 Scope

This document is a description of a reference coding method to be used for the development of a new compression method ITU-T recommendation- H.26L. The basic configuration of the algorithm is similar to H.263.

Some of the differences from H.263 are:

- Only one regular VLC is used for symbol coding
- 1/4 pixel positions are used for motion prediction
- A number of different blocksizes are used for motion prediction
- Residual coding is based on 4x4 blocks and a integer transform is used
- Multiple reference frames may be used for prediction and this is considered to replace any use of B-frames

The first part of the document describes the coding method by mainly defining the decoder actions. Towards the end, 'test model issues' relevant for the encoder to be used as reference for the development of the standard will be covered. This split should make it easy at a later stage to split the document into what is relevant for the standard and a test model. However, at the moment we find it preferable to have one combined document.

2 Organization of source- and compressed data

2.1 Picture formats

At the moment only the QCIF and CIF formats are included in the model [Not true of the software anymore – the document needs to be updated about this (but not relevant to testing at QCIF and CIF resolutions)].

2.2 Subdivision of a picture into macroblocks

A CIF picture is divided into 18x22 = 396 macroblocks. Similarly a QCIF picture is divided into 99 macroblocks as indicated in Figure 1. At the moment there are no other layers in the described model. [Slice layers are supported in the software, but not fully reflected in this document.]

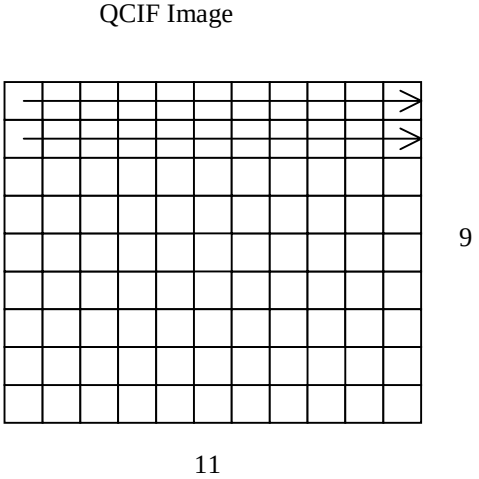


Figure 1 Subdivision of a QCIF picture into 16x16 macroblocks

2.3 Order of the bitstream within a macroblock

Figure 2 and Figure 3 indicates how a macroblock is divided and the order of the different syntax elements resulting from coding a macroblock.

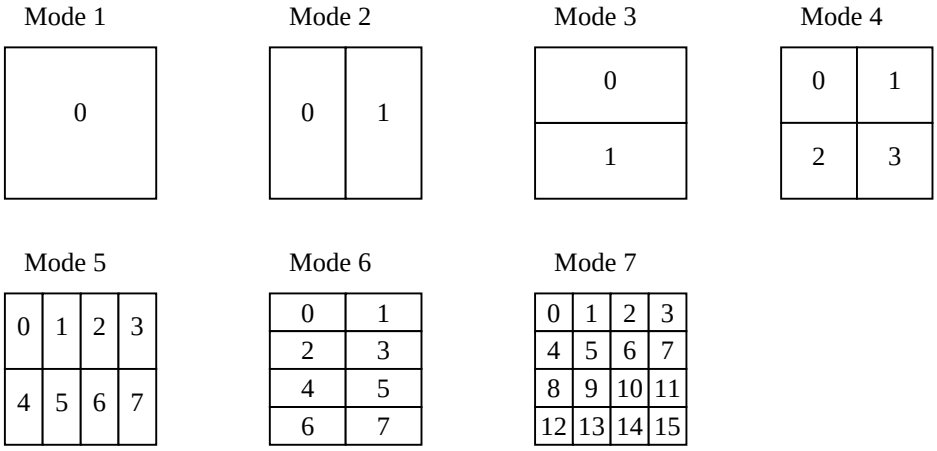


Figure 2 Numbering of the vectors for the different blocks depending on the inter mode. For each block the horizontal component comes first followed by the vertical component

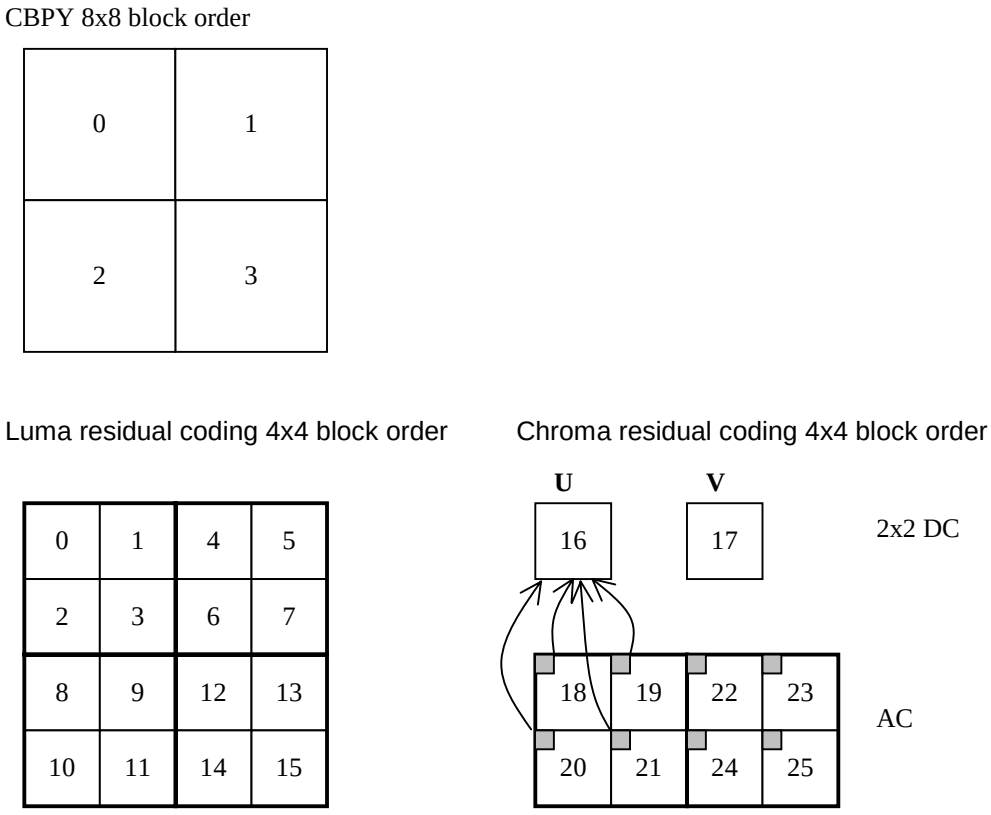


Figure 3 Ordering of blocks for CBPY and residual coding of 4x4 blocks

2.4 Syntax

2.4.1 Syntax diagram

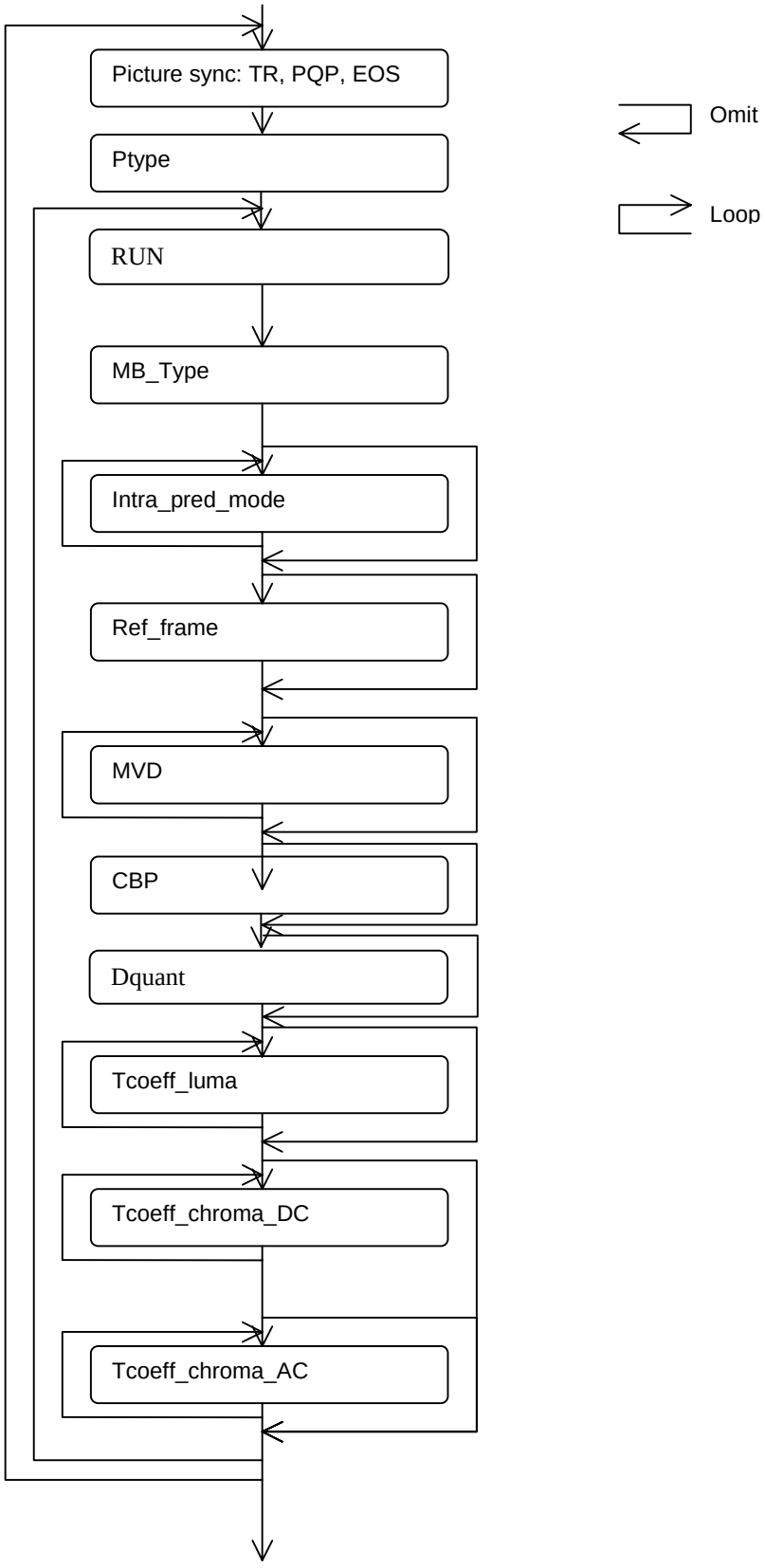


Figure 4 Syntax diagram for all the elements in the video bitstream

3 Description of syntax elements

3.1 Picture sync

The first codeword in a picture is 31 bits long (L = 31). It works as a picture sync, but it also contains an INFO part that has 15 bits. These bits are used for:

TR(8 bits) Temporal reference. The value of TR is formed by incrementing its value in the temporally-previous reference picture header by one plus the number of skipped or non-reference pictures at the picture clock frequency since the previously transmitted one.

PQP(5 bits) Picture QP: Information about the quantizer QUANT to be used for luma for the picture. (See under Quantization concerning QUANT for chroma). The 5 bit representation is the natural binary representations of the values of QP which range from 0 to 31. QP is a pointer to the actual quantization parameter QUANT to be used. (See below under quantization). The range of quantization value is still about the same as for H.263, 1-31. An approximate relation between the QUANT in H.263 and QP is: $QUANT_{H.263}(QP) \approx QP_0(QP) = 2^{QP/6}$. $QP_0()$ will be used later for scaling purposes when selecting prediction modes.

Formats 0 indicates QCIF, 1 indicates CIF

EOS 0 for picture header. 1 indicates End Of Sequence

The sketch below indicates where the different bits are located within INFO.

8b: TR	5b: PQP	1b: Formats	1b: EOS

3.2 Picture type (Ptype)

Code_number =0: Inter picture with prediction from the most recent decoded picture only.

Code_number =1: Inter picture with possibility of prediction from more than one previous decoded picture. For this mode information reference picture for prediction must be signalled for each macroblock.

Code_number =2: Intra picture.

Code_number =3: B picture with prediction from the most recent previous decoded and subsequent decoded pictures only.

Code_number =4: B picture with possibility of prediction from more than one previous decoded picture and subsequent decoded picture. When using this mode, information reference frame for prediction must be signaled for each macroblock.

3.3 RUN

A macroblock is called skipped if no information is sent. In that case the reconstruction of an inter macroblock is made by copying the colocated picture material from the laset decoded frame. (See separate definition for B-pictures). RUN indicates the number of skipped macroblocks in an inter- or B-picture before a coded picture. If the last MB of the frame is not coded an additional codeword is added which points to a MB outside the picture.

3.4 Macro block type (MB_Type)

Refer to **Error! Reference source not found.**. There are different MB-Type tables for Intra and Inter frames.

3.4.1 Intra

Intra 4x4 Intra coding as defined in sections 3.5.1 to 3.5.8.

Imode, nc, AC — See definition in section 3.5.9.3. These modes refer to 16x16 intra coding.

3.4.2 Inter

Skip — No further information about the macroblock is transmitted. A copy of the colocated macroblock in the most recent decoded picture is used as reconstruction for the present macroblock.

NxM (eg. 8x4) — The macroblock is predicted from a past picture with block size NxM. For each NxM block motion vector data is provided. Depending on N and M there may be 1 to 16 sets of motion vector data for a macroblock.

Intra 4x4 — 4x4 intra coding.

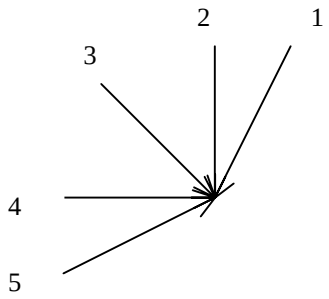
Code numbers from 9 and upwards represent 16x16 intra coding.

3.5 Intra prediction mode (Intra_pred_mode)

Even in Intra mode, prediction is always used for each sub block in a macroblock. A 4x4 block is to be coded (pixels labelled a to p below). The pixels A to I from neighbouring blocks are already decoded and may be used for prediction.

```
I A B C D
E a b c d
F e f g h
G i j k l
H m n o p
```

There are 6 intra prediction modes labelled 0 to 5. Mode 0 is 'DC-prediction' (see below). The other modes represent directions of predictions as indicated below.



3.5.1 Mode 0: DC prediction

Generally all pixels are predicted by (A+B+C+D+E+F+G+H)//8. If four of the pixels are outside the picture, the average of the remaining four is used for prediction. If all 8 pixels are outside the picture the prediction for all pixels in the block is 128. A block may therefore always be predicted in this mode.

3.5.2 Mode 1:

This mode is used only if all A,B,C,D are inside the picture.

a is predicted by:	(A+B)/2
e is predicted by	B
b,i are predicted by	(B+C)/2
f,m are predicted by	C
c,j are predicted by	(C+D)/2
d,g,h,k,l,n,o,p are predicted by	D

3.5.3 Mode 2: Vertical prediction

If A,B,C,D are inside the picture, a,e,i,m are predicted by A, b,f,j,n by B etc.

3.5.4 Mode 3: Diagonal prediction

This mode is used only if all A,B,C,D,E,F,G,H,I are inside the picture. This is a 'diagonal' prediction.

m is predicted by:	(H+2G+F)//4
i,n are predicted by	(G+2F+E)//4
e,j,o are predicted by	(F+2E+I)//4
a,f,k,p are predicted by	(E+2I+A)//4
b,g,l are predicted by	(I+2A+B)//4
c,h are predicted by	(A+2B+C)//4
d is predicted by	(B+2C+D)//4

3.5.5 Mode 4:Horizontal prediction

If E,F,G,H are inside the picture, a,b,c,d are predicted by E, e,f,g,h by F etc.

3.5.6 Mode 5:

This mode is used only if all E,F,G,H are inside the picture.

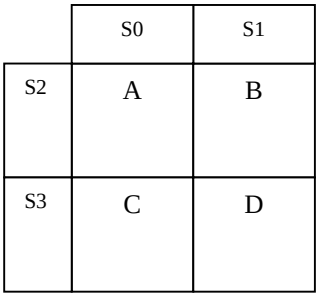
a is predicted by:	(E+F)/2
b is predicted by	F
c,e are predicted by	(F+G)/2
f,d are predicted by	G
i,g are predicted by	(G+H)/2
h,j,k,l,m,n,o,p are predicted by	H

3.5.7 Prediction of chroma blocks

For chroma prediction there is only one mode. No information is therefore needed to be transmitted. The prediction is indicated in the figure below. The 8x8 chroma block consists of 4 4x4 blocks A,B,C,D. S0,1,2,3 are the sums of 4 neighbouring pixels.

If S0, S1, S2, S3 are all inside the frame:
A = (S0 + S2 + 4)/8
B = (S1 + 2)/4
C = (S3 + 2)/4
D = (S1 + S3 + 4)/8

If only S0 and S1 are inside the frame:
A = (S0 + 2)/4
B = (S1 + 2)/4
C = (S0 + 2)/4
D = (S1 + 2)/4



If only S2 and S3 are inside the frame:
A = (S2 + 2)/4
B = (S2 + 2)/4
C = (S3 + 2)/4
D = (S3 + 2)/4
If S0, S1, S2, S3 are all outside the frame: A = B = C = D = 128
(Note: This prediction should be considered changed)

3.5.8 Coding of Intra prediction modes

Since each of the 4x4 luma blocks shall be assigned a prediction mode, this will require a considerable number of bits if coded directly. We have therefore tried to find more efficient ways of coding mode information. First of all we observe that the chosen prediction of a block is highly correlated with the prediction modes of adjacent blocks. This is illustrated in Figure 5 a. When the prediction modes of A and B are known (including the case that A or B or both are outside the picture) an ordering of the most probable, next most probable etc. of C is given. This ordering is listed in Table 1. For each prediction mode of A and B a list of 5 numbers is given. Example: Prediction mode for A and B is 2. The string 2 1 0 3 4 5 indicates that mode 2 is also the most probable mode for block C. Mode 1 is the next most probable one etc. In the bitstream there will for instance be information that Prob0 = 1 (see **Error! Reference source not found.**) indicating that the next most probable mode shall be used for block C. In our example this means Intra prediction mode 1. Use of '–' in the table indicates that this instance can not occur because A or B or both are outside the picture.
For more efficient coding, information on intra prediction of two 4x4 luma blocks are coded in one codeword (Prob0 and Prob1 in **Error! Reference source not found.**). The order of the resulting 8 codewords is indicated in Figure 5 b.

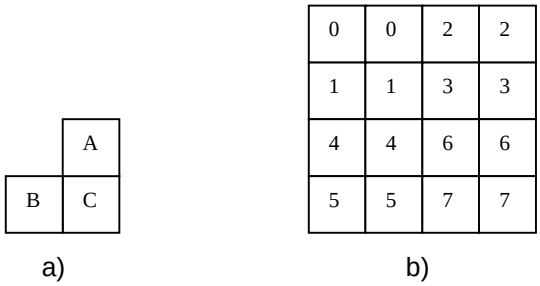


Figure 5 a) Prediction mode of block C shall be established. A and B are adjacent blocks. b) order of intra prediction information in the bitstream

Table 1 Prediction mode as a function of ordering signalled in the bitstream (see text)

B\A	outside	0	1	2	3	4	5
outside	0-----	021---	102---	201---	012---	012---	012---
0	045---	041352	104325	230415	304215	043152	043512
1	045---	014325	102435	203145	032145	041325	014352
2	045---	012345	102345	210345	302145	042135	013245
3	045---	304152	310425	231054	304215	403512	305412
4	405---	403512	401532	240351	430512	403512	405312
5	504---	540312	015432	201453	530412	450312	504132

3.5.9 Intra mode based on 16x16 macroblocks (16x16 intra mode)

This intra mode is particularly suitable for regions with little details, also referred to as 'flat' regions.

3.5.9.1 Prediction modes

Assume that the block to be predicted has pixel locations 0 to 15 horizontally and 0 to 15 vertically. We use the notation P(i,j) where i,j = 0..15. P(i,-1), i=0..15 are the neighboring pixels above the block and P(-1,j), j=0..15 are the neighboring pixels to the left of the block. Pred(i,j) i,j = 0..15 is the prediction for the whole luma macroblock. We have 4 different prediction modes:

IMODE = 0 (vertical)

Pred(i,j) = P(i,-1), i,j=0..15

IMODE = 1 (horizontal)

Pred(i,j) = P(-1,j), i,j=0..15

IMODE = 2 (DC prediction)

Pred(i,j) = $(\sum_{i=0}^{15}(P(-1,i)+P(i,-1)))/32$ i,j=0..15

IMODE = 3 (Plane prediction)

Pred(i,j) = (a + bx(i-7) + cx(j-7) +16)/32

Where:

a = 16x(P(-1,15) + P(15,-1))

b = 5x(*H*/4)/16

c = 5x(*V*/4)/16

$$H = \sum_{i=1}^{8} ix(P(7+i,-1)+P(7-i,-1))$$

$$V = \sum_{j=1}^{8} jx(P(-1,7+j)+P(-1,7-j))$$

3.5.9.2 Residual coding

The residual coding is based on 4x4 transform. But similar to coding of chroma coefficients, another 4x4 transform to the 16 DC coefficients in the macroblock are added. In that way we end up with an overall DC for the whole MB which works well in flat areas.

Since we use the same integer transform to DC coefficients, we have to perform additional normalization to those coefficients, which implies a division by 676. To avoid the division we performed normalization by $49/2^{15}$ on the encoder side and $48/2^{15}$ on the decoder side, which gives sufficient accuracy.

Only single scan is used for 16x16 intra coding.

To produce the bitstream, we first scan through the 16 'DC transform' coefficients. There is no 'CBP' information to indicate no coefficients on this level. If AC = 1 (see below) ac coefficients of the 16 4x4 blocks are scanned. There are 15 coefficients in each block since the DC coefficients are included in the level above.

3.5.9.3 Signalling of mode information for 16x16 intra coding

See **Error! Reference source not found.**. Three parameters have to be signaled. They are all included in MB-type.

Imode: 0,1,2,3

AC: 0 means there are no ac coefficients in the 16x16 block. 1 means that there is at least one ac coefficient and all 16 blocks are scanned.

nc: CBP for chroma (see 3.8)

3.6 Reference frame (Ref_frame)

If PTYPE indicates possibility of prediction from more than one previous decoded picture, the exact frame to be used must be signalled. This is done according to the following table.

Code_number	Reference frame
0	The last decoded frame (1 frame back)
1	2 frames back
2	3 frames back
..	..

3.7 Motion Vector Data (MVD)

If so indicated by MB_type, vector data for 1-16 blocks are transmitted. For every block a prediction is formed for the horizontal and vertical components of the motion vector. MVD signals the difference between the vector component to be used and this prediction. The order in which vector data is sent is indicated in Figure 2. Motion vectors are allowed to point to pixels outside the reference frame. If a pixel outside the reference frame is referred to in the prediction process the nearest pixel belonging to the frame (an edge or corner pixel) shall be used. This process is performed on a pixel bases..

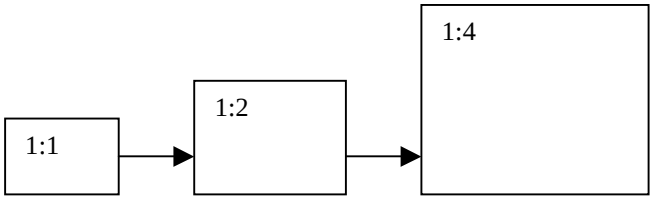
3.7.1 Fractional pixel accuracy

3.7.1.1 1/4 pixel accuracy

Note: It has been adopted to use direct interpolation to produce 1/4 pixel values (see VCEG-L20) at the decoder. However, this also has implications on the encoder which must use extra effort to avoid encoder/decoder mismatch. This issue will be further clarified in an ad.hoc. group and a solution presented at the next meeting. The text will then be updated.

In the baseline coding mode, 1/4 pixel accuracy is used to define motion vectors. To avoid encoder/decoder mismatch, the interpolation procedure has to be defined in detail.

The figure below indicates that interpolation for luminance may be defined as taking place in two steps.



Interpolation: 6H, 6V bilinear

Step I: Generation of ½ pixel positions

The 'x' - es below represent integer pixel positions. First the '+' - es (horizontal ½ pixel positions) are produced by using a 6 tap filter: (1,-5,20,20,-5,1)/32. The result is rounded to the nearest integer and clipped to the range 0 to 255. Then all the '*' – es are produced by using the same 6 tap. The result is rounded to the nearest integer and clipped to the range 0 to 255. If the filtering process refer to a pixel outside the picture, it is replaced by the nearest picture edge pixel.

```
x + x + x + x + x
* * * * * * * * *
x + x + x + x + x
* * * * * * * * *
x + x + x + x + x
* * * * * * * * *
```

Step II: Generation of ¼ pixel positions

The 'x' - es below represent ½ pixel positions (from above). First the '+' - es are produced by linear interpolation (average of the neighbouring ½ pixel positions). The result is truncated to the nearest integer. Then all the '*' – es are produced by vertical linear interpolation with truncation to the nearest integer.

```
x + x + x + x + x
* * * * * * * * *
x + x + x + x + x
* * * * * * * * *
x + x + x + x + x
* * * * * * * * *
```

Interpolation position with more low pass filtering

The figure below indicate a grid with interpolated positions where capital letters represent integer positions, numbers represent ½ pixel positions and lower case letters represent 1/4 pixel positions.

```
A a 1 b B
c d e f g
2 h 3 i 4
j k l m n
C o 5 p D
```

Relative to integer positions, the position m is chosen to represent a position with more low pass filtering. According to the definition above, m ≈ ('3' + '4' + '5' + 'D')/4. (≈ is used due to possible rounding effects). Instead we define:
m = ('A' + 'B' + 'C' + 'D' + 2)/4

3.7.1.2 1/8 pixel accuracy

For a higher complexity or higher coding efficiency profile (not yet defined) 1/8 pixel accuracy prediction using 8-tap filters may be used. The interpolation process is assumed to be performed in a separable manner - first in the horizontal direction and then in the vertical direction. Hence the filter operation is fully specified by the one-dimensional filter taps. The filter taps are given below:

Position								
Integer				1				
1/8	(-3	12	-37	485	71	-21	6	-1)/512
2/8	(-3	12	-37	229	71	-21	6	-1)/256
3/8	(-6	24	-76	387	229	-60	18	-4)/512
4/8	(-3	12	-39	158	158	-39	12	-3)/256
5/8	(-4	18	-60	229	387	-76	24	-6)/512
6/8	(-1	6	-21	71	229	-37	12	-3)/256
7/8	(-1	6	-21	71	485	-37	12	-3)/512

3.7.2 Prediction of vector components

In all macroblocks with square motion vector blocks (16x16, 8x8, 4x4) "median prediction" (see 3.7.2.1) is used. In case the macroblock may be classified to have directional segmentation the prediction is defined in 3.7.2.2.

3.7.2.1 Median prediction

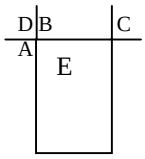
In the figure below the vector component E of the indicated block shall be predicted. The prediction is normally formed as the median of A, B and C. However, the prediction may be modified as described below. Notice that it is still referred to as "median prediction"

A The component applying to the pixel to the left of the upper left pixel in E

B The component applying to the pixel just above the upper left pixel in E

C The component applying to the pixel above and to the right of the upper right pixel in E

D The component applying to the pixel above and to the left of the upper left pixel in E



A, B, C, D and E may represent motion vectors from different reference pictures. As an example we may be seeking prediction for a motion vector for E from the last decoded picture. A, B, C and D may represent vectors from 2, 3, 4 and 5 pictures back. The following substitutions may be made prior to median filtering.

- If A and D are outside the picture, their values are assumed to be zero and they are considered to have "different reference picture than E".
- If D, B, C are outside the picture, the prediction is equal to A (equivalent to replacing B and C with A before median filtering).

- If C is outside the picture or still not available due to the order of vector data (see Figure 2), C is replaced by D.

If any of the blocks A, B, C, D are intra coded they count as having "different reference frame. If one and only one of the vector components used in the median calculation (A, B, C) refer to the same reference picture as the vector component E, this one vector component is used to predict E.

3.7.2.2 Directional segmentation prediction

If the macroblock where the block to be predicted belongs to has a directional segmentation (vector block size of 8x16, 16x8, 8x4 or 4x8) the prediction is generated as follows (refer to figure below and the definitions of A, B, C, E above):

Vector block size 8x16:

Left block:	A is used as prediction if it has the same reference picture as E, otherwise "Median prediction" is used
Right block:	C is used as prediction if it has the same reference picture as E, otherwise "Median prediction" is used

Vector block size 16x8:

Upper block:	B is used as prediction if it has the same reference picture as E, otherwise "Median prediction" is used
Lower block:	A is used as prediction if it has the same reference picture as E, otherwise "Median prediction" is used

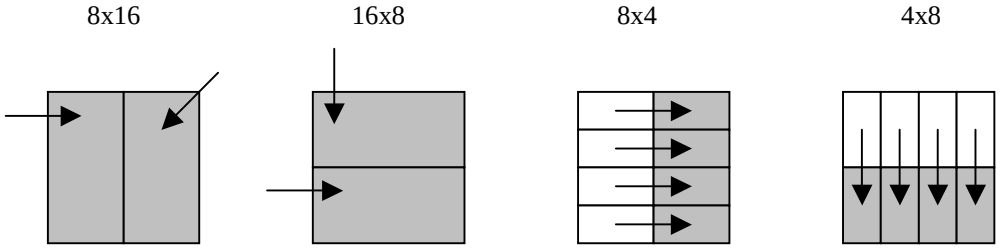
Vector block size 8x4:

For white blocks:	"Median prediction" is used
For shaded blocks:	A is used as prediction

Vector block size 4x8:

For white blocks:	"Median prediction" is used
For shaded blocks:	B is used as prediction

If the indicated prediction block is outside the picture, the same substitution rules are applied as in the case of median prediction.



3.7.3 Chroma vectors

Chroma vectors are derived from the luma vectors. Since chroma has half resolution compared to luma, the chroma vectors are obtained by a division of two:

Croma_vector = Luma_vector/2 - which means that the chroma vectors have a resolution of 1/8 pixel.

Due to the half resolution, a chroma vector applies to 1/4 as many pixels as the luma vector. For example if the luma vector applies to 8x16 luma pixels, the corresponding chroma vector applies to 4x8 chroma pixels and if the luma vector applies to 4x4 luma pixels, the corresponding chroma vector applies to 2x2 chroma pixel.

For fractional pixel interpolation for chroma prediction, bilinear interpolation is used. The result is rounded to the nearest integer.

3.8 Coded Block Pattern (CBP)

The CBP contains information of which 8x8 blocks - luma and chroma - contain transform coefficients. Notice that an 8x8 block contains 4 4x4 blocks meaning that the statement '8x8 block contains coefficients' means that 'one or more of the 4 4x4 blocks contain coefficients'. The 4 least significant bits of CBP contain information on which of 4 8x8 luma blocks in a macroblock contains nonzero coefficients. Let us call these 4 bits CBPY. The ordering of 8x8 blocks is indicated in Figure 3. A 0 in position n of CBP (binary representation) means that the corresponding 8x8 block has no coefficients whereas a 1 means that the 8x8 block has one or more non-zero coefficients.

For chroma we define 3 possibilities:

nc=0: no chroma coefficients at all.

nc=1 There are nonzero 2x2 transform coefficients. All chroma AC coefficients = 0. Therefore we do not send any EOB for chroma AC coefficients.

nc=2 There may be 2x2 nonzero coefficients and there is at least one nonzero chroma AC coefficient present. In this case we need to send 10 EOBs (2 for DC coefficients and 2x4=8 for the 8 4x4 blocks) for chroma in a macroblock.

The total CBP for a macroblock is: **CBP = CBPY + 16xnc**

The CBP is signalled with a different codeword for Inter macroblocks and Intra macroblocks since the statistics of CBP values are different in the two cases.

3.9 Dquant

Dquant contains the possibility of changing QUANT on the macroblock level. It does not need to be present for macroblocks without nonzero transform coefficients. More specifically Dquant is present for non-skipped macroblocks:

- If CBP indicates that there are nonzero transform coefficients in the MB

or

- If the MB is 16x16 based intra coded

Tha value of Dquant shall be interpreted in the same way as Motion Vecor Data (**Error! Reference source not found.**). Its value may range from -16 to +16 which enables the QP to be changed to any value in the range 0-31.

$QUANT_{new} = modulo_{32}(QUANT_{old} + Dquant + 32)$ (also known as "arithmetic wrap")

4 Transform and inverse transform

This section defines all elements related to transform coding and decoding. It is therefore relevant to all the syntax elements 'Tcoeff' in the syntax diagram.

4.1 4x4 block size

Instead of DCT, an integer transform with basically the same coding property as a 4x4 DCT is used. The transformation of the pixels a,b,c,d into 4 transform coefficients A,B,C,D is defined by:

```
A = 13a + 13b + 13c + 13d
B = 17a +  7b -  7c - 17d
C = 13a - 13b – 13c + 13d
D =  7a - 17b + 17c -  7d
```

The inverse transformation of transform coefficients A,B,C,D into 4 pixels a',b',c',d' is defined by:

```
a' = 13A + 17B + 13C +  7D
b' = 13A +  7B - 13C – 17D
c' = 13A –  7B – 13C + 17D
```

d' = 13A – 17B + 13C - 7D

The relation between a and a' is: a' = 676a. This is because the expressions defined above contain no normalisation. Normalisation will be performed in the quantization/dequantisation process and a final shift after inverse quantization.

The transform/inverse is performed both vertically and horizontally in the same manner as in H.263. By the above exact definition of inverse transform, the same operations will be performed on coder and decoder side which means that we have no 'inverse transform mismatch'.

4.2 2x2 transform/inverse transform of chroma DC coefficients

With the low resolution of chroma it seems to be preferable to have larger blocksize than 4x4. Specifically the 8x8 DC coefficient seems very useful for better definition of low resolution chroma. The 2 dimensional 2x2 transform procedure is illustrated below. DC0,1,2,3 are the DC coefficients of 2x2 chroma blocks.

DC0	DC1	**Two dimensional 2x2 transform** ⇒	DDC(0,0)	DDC(1,0)
DC2	DC3		DDC(0,1)	DDC(1,1)

Definition of transform:

DCC(0,0) = (DC0+DC1+DC2+DC3)/2

DCC(1,0) = (DC0-DC1+DC2-DC3)/2

DCC(0,1) = (DC0+DC1-DC2-DC3)/2

DCC(1,1) = (DC0-DC1-DC2+DC3)/2

Definition of inverse transform:

DC0 = (DCC(0,0)+ DCC(1,0)+ DCC(0,1)+ DCC(1,1))/2

DC1 = (DCC(0,0)- DCC(1,0)+ DCC(0,1)- DCC(1,1))/2

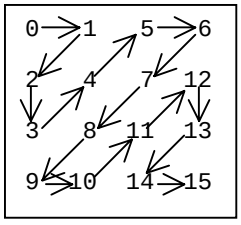
DC2 = (DCC(0,0)+ DCC(1,0)- DCC(0,1)- DCC(1,1))/2

DC3 = (DCC(0,0)- DCC(1,0)- DCC(0,1)+ DCC(1,1))/2

4.3 Scanning and quantization

4.3.1 Simple scan

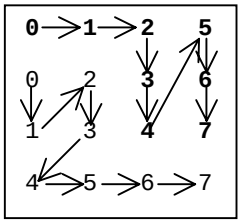
Except for Intra coding of luma with QP<24, simple scan is used. This is basically zig-zag scanning similar to the one used in H.263. The scanning pattern is:



4.3.2 Double scan

When using the VLC defined above, we use a one bit code for EOB. For Inter blocks and Intra with high QP the probability of EOB is typically 50% which is well matched with the VLC. In other words this means that we on average have one non-zero coefficient per 4x4 block in addition to the EOB code (remember

that a lot of 4x4 blocks only have EOB). On the other hand, for Intra coding we typically have more than one non-zero coefficient per 4x4 block. This means that the 1 bit EOB becomes inefficient. To improve on this the 4x4 block is subdivided into two parts that are scanned separately and with one EOB each. The two scanning parts are shown below – one of them in bold.



4.3.3 Quantization

The quantization/dequantization process shall perform 'normal' quantization/dequantization as well as take care of the fact that transform operations are kept very simple and therefore do not contain normalization of transform coefficients. 32 different QP values are used. They are arranged so that there is an increase of step size of about 12% from one QP to the next. Increase of QP by 6 means that the step size is about doubled. There is no 'dead zone' in the quantization process and the total range of step size from smallest to largest is about the same as for H.263.

The QP signalled in the bitstream applies for luma quantization/dequantization. This could be called QP_{luma}. For chroma quantization/dequantization a different value - QP_{chroma} - is used. The relation between the two is:

QP_{luma}	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
QP_{chroma}	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	17	18	19	20	20	21	22	22	23	23	24	24	25	25

When QP is used in the following we mean QP_{luma} or QP_{chroma} depending on what is appropriate.

Two arrays of numbers are used for quantization/dequantization.

A(QP=0,..,31)
620,553,492,439,391,348,310,276,246,219,195,174,155,138,123,110,98,87,78,69,62,55,49,44,39,35,
31,27,24,22,19,17

B(QP=0,..,31)

3881,4351,4890,5481,6154,6914,7761,8718,9781,10987,12339,13828,15523,17435,19561,21873,24552,27656,308
47,34870,38807,43747,49103,54683,61694,68745,77615,89113,100253,109366,126635,141533

The relation between A() and B() is: $A(QP)xB(QP)x676^2 = 2^{40}$.

It is assumed that a coefficient K is quantized in the following way:

$LEVEL = (KxA(QP) + fx2^{20})/2^{20}$ |f| is in the range (0-0.5) and f has the same sign as K.

Dequantization:

K' = LEVELxB(QP)

After inverse transform this results in pixel values that are 2^{20} too high. A shift of 20 bits (with rounding) is therefore needed on the reconstruction side. The definition of transform and quantization is designed so that no overflow will occur with use of 32 bit arithmetic.

4.3.4 Scanning and quantization of 2x2 chroma DC coefficients

DDC() is quantized (and dequantized) separately resulting in LEVEL, RUN and EOB. The scanning order is: DCC(0,0), DCC(1,0), DCC(0,1), DCC(1,1). Inverse 2x2 transform as defined above is then performed after dequantization resulting in dequantized 4x4 DC coefficients: DC0' DC1' DC2' DC3'.

Chroma AC coefficients (4x4 based) are then quantized similarly to before. Notice that there are only 15 AC coefficients. The maximum size of RUN is therefore 14. However, for simplicity we use the same

relation between LEVEL, RUN and Code no. As defined for 'Simple scan' in **Error! Reference source not found.**.

4.4 Use of 2-dimensional model for coefficient coding.

In the 3D model for coefficient coding (see H.263) there is no good use of a short codeword of 1 bit. On the other hand, with the use of 2D VLC plus End Of Block (EOB) (as used in H.261, H.262) and with the small block size, 1 bit for EOB is usually well matched to the VLC.

Furthermore, with the fewer non-zero coefficients per block, the advantage of using 3D VLC is reduced.

As a result we use a 2D model plus End Of Block (EOB) in the present model. This means that an event to be coded (RUN, LEVEL) consists of:

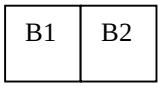
RUN which is the number of zero coefficients since the last nonzero coefficient.

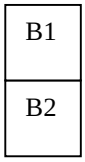
LEVEL the size of the nonzero coefficient

EOB signals that there are no more nonzero coefficients in the block

4.5 Deblocking filter

The deblocking filter process takes place both across vertical block edges and horizontal block edges. The filtering is performed first across vertical edges. The filtered pixels are then used when performing filtering across horizontal block edges. As a result block1 and block2 (see below) may be located in one of the positions:

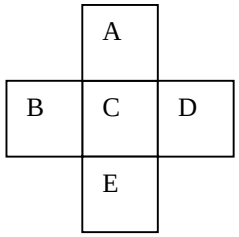




The filtering may be based on 4x4 block boundaries or macroblock boundaries if one or both adjacent macroblocks are intra coded (see 4.5.3)

4.5.1 Block strength

Here we consider filteringbased on a 4x4 block structure. Each 4x4 block in a reconstructed picture is assigned a **Strength** value ranging from 0 to 3. This value is assigned based on different criteria. The maximum value is chosen. Assume that one criterion results in **value** to be assigned. The new assignment is: **Strength = max(Strength,value),** so that we always keep the maximum value. In the figure below A, B, C, D and E are 4x4 blocks. Blocks may be luma or chroma. Different strenght is used for luma and chroma. This difference is introduced by $\mathbf{St_0}$ which is 0 for chroma and 1 for luma.



The procedure for assignment is as follows:

- Initially each 4x4 block of a picture is assigned **Strength = 0**
- If C is intra coded: $\mathbf{Strength_C = St_0}$ +2**,** $\mathbf{Strength_A = max(Strength_A, St_0}$ +1**),** $\mathbf{Strength_B = max(Strength_B, St_0}$ +1**),** $\mathbf{Strength_D = max(Strength_D, St_0}$ +1**),** $\mathbf{Strength_E = max(Strength_E, St_0}$ +1**)**
- If C has nonzero transform coefficients: $\mathbf{Strength_C = max(Strength_C,2)}$**,** $\mathbf{Strength_A = max(Strength_A,1)}$**,** $\mathbf{Strength_B = max(Strength_B,1)}$**,** $\mathbf{Strength_D = max(Strength_D,1)}$**,** $\mathbf{Strength_E = max(Strength_E,1)}$
- For luma blocks: If the absolute difference between one of the vector components of block A and block C is at least one integer pixel (four ¼ pixels) then: $\mathbf{Strength_A = max(Strength_A,1)}$**,** $\mathbf{Strength_C =}$

max(Strength$_C$,1). The same procedure is applied to vector differences between blocks (B and C), (C and D) and (C and E)

- For chroma blocks: Since chroma has half the resolution of luma, there may be different vectors also inside the chroma block C. This is illustrated by the horizontal broken line i block C. The procedure for a chroma block is that if the absolute difference between one of the vector components on each side of the boundary between A and C or across the dotted line inside C is at least one integer pixel (four ¼ pixels) then: **Strength$_A$ = max(Strength$_A$,1), Strength$_C$ = max(Strength$_C$,1)**. A similar procedure is applied to vector differences between blocks (B and C), (C and D) and (C and E) (*Note: This description for chroma may seem strange. It is written to reflect what happens in the software. A simpler rule should be sought and reflected in a later version of TML as well as in the software*)

4.5.2 Filtering of pixels on both sides of a 4x4 block boundary

In the following description it is assumed that a vertical 4x4 grid boundary is to be filtered (that is, horizontal filtering takes place). Horizontal grid boundaries are filtered utilizing the same algorithm, but 'left' pixels should be substituted for 'above' pixels and 'right' pixels for 'below' pixels. Let us denote the set of eight pixels across a 4x4 block boundary as l_4, l_3, l_2, l_1, r_1, r_2, r_3, r_4.
Up to two pixels can be updated as a result of the filtering process on both sides of the boundary (that is at most l_2, l_1, r_1, r_2). The number of updated pixels on both sides of the boundary depends on corresponding right and left activity parameters a_r and a_l. Both activity parameters are limited by the so called overall activity parameter *n*:

$$n = \begin{cases} 1 & \alpha + \alpha/2 \le 2\Delta \\ 2 & \alpha \le 2\Delta < \alpha + \alpha/2 \\ 3 & 2\Delta < \alpha \end{cases}$$

where $\Delta=|r_1-l_1|$ is a positive integer representing gray level difference across the boundary, and α is a QP dependent threshold specified in Table 1.

QP	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
α	0	0	0	0	0	0	0	0	3	3	3	7	7	7	12	17	17	22	28	34	40	47	60	67	82	89	112	137	153	187	213	249
β	0	0	0	0	0	0	0	0	3	3	3	4	4	4	6	6	6	7	8	8	9	9	10	10	11	11	12	12	13	13	14	14

Table 1. QP dependent threshold parameters α and β.

The right filter activity parameter a_r is equal to the smallest integer i>0 not larger than *n* for which the inequality $|r_1-r_{i+1}|*i > \beta$ holds. The QP dependent threshold β is given in Table 1. If no $i < n$ satisfies the condition, a_r is set to *n*. The left filter activity parameter a_l is obtained using the same set of rules substituting pixels on the right of the boundary r_i with pixels l_i on the left of the boundary. Once a_r and a_l are calculated, the pixels to be updated and their filtering support windows are determined by using Table 2. The same table applies for both right and left pixel although only right pixels are listed in the table. Information on how to filter the left pixels is obtained by swapping a_r and a_l and substituting r_1, r_2 with l_1, l_2.

a_r (only if a_l>1)	**pixel r_1**	**pixel r_2**
1	no filtering	no filtering
2	filtered	no filtering
3	filtered	filtered

Table 2. Pixels to be updated as a function of a_r.

When filtering within P frames, the left (right) activity parameter is upper bounded by 2 if *strength* of the left (right) 4x4 block is no larger than 1. Selected pixels are filtered as follows:

$$L_1 = (21*l_2 + 22*l_1 + 21*r_1 + 32)/64$$
$$L_2 = (21*l_3 + 22*l_2 + 21*L_1 + 32)/64$$
$$R_1 = (21*l_1 + 22*r_1 + 21*r_2 + 32)/64$$
$$R_2 = (21*R_1 + 22*r_2 + 21*r_3 + 32)/64$$

When either of pixels r_1, l_1 are updated by filtering then the difference between filtered and nonfiltered value is clipped to the range [-C, C] where C = (Clip(QP,strength$_{left}$) + Clip(QP,strength$_{right}$) + a_r + a_l) / 2. When pixel r_2 is updated by filtering then the difference is clipped to the range [-C_{right}, C_{right}] with C_{right} = Clip(QP,strength$_{right}$). Similarly for pixel l_2.

Definition of the table Clip(QP,strength):

QP	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	29	30	31
Clip(qp,1)	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	3	3
Clip(qp,2)	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	2	2	2	2	3	3	3	3	4	5	5
Clip(qp,3)	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	2	2	2	2	3	3	3	4	4	5	5	5	7	8	9	

The filter is applied on a 4x4 grid for both luminance and chrominace, first for all the vertical then for all the horizontal block boundaries.

4.5.3 Filtering across macroblock boundaries connected to intra coding

In case of intra coding and areas in the picture with little texture the subjective quality may be improved by stronger filtering across macroblock boundaries. This stronger filtering across a macroblock boundary is performed if:

One or both of the adjacent macroblocks are intra coded

and

a_r and a_l are equal to 3

and

Δ is in the range [2-qp/4]

In case these conditions are fulfilled, the pixels l_3 to r_3 when filtering luminance frames and l_2 to r_2 when filtering chrominance frames are modified in the following way:

$$L_1 = (25*l_3 + 26*l_2 + 26*l_1 + 26*r_1 + 25*r_2 + 64)/128$$
$$L_2 = (25*l_4 + 26*l_3 + 26*l_2 + 26*L_1 + 25*r_1 + 64)/128$$
$$L_3 = (\quad 26*l_4 + 51*l_3 + 26*L_2 + 25*L_1 + 64)/128$$
$$R_1 = (25*l_2 + 26*l_1 + 26*r_1 + 26*r_2 + 25*r_3 + 64)/128$$
$$R_2 = (25*l_1 + 26*R_1 + 26*r_2 + 26*r_3 + 25*r_4 + 64)/128$$
$$R_3 = (25*R_1 + 26*R_2 + 51*r_3 + 26*r_4 \quad + 64)/128$$

Notice that when filtering across the next 4x4 block boundary, pixel r_3 may get overwritten. To compensate the effect R_3 is stored and used as l_2 when filtering across the next 4x4 block boundary.

Otherwise filtering is done as described in the section above.

5 Entropy Coding

5.1 Universal Variable Length Coding (UVLC)

In the default entropy coding mode, a universal VLC is used to code all syntax. The table of codewords may be written in the following compressed form.

$$
\begin{array}{c}
1 \\
0\ x_0\ 1 \\
0\ x_1\ 0\ x_0\ 1 \\
0\ x_2\ 0\ x_1\ 0\ x_0\ 1 \\
0\ x_3\ 0\ x_2\ 0\ x_1\ 0\ x_0\ 1 \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots
\end{array}
$$

where x_n take values 0 or 1. We will sometimes refer to a codeword with its length in bits (L) and INFO = x_n .. x_1 x_0 . Notice that the number of bits in INFO is L/2 (division by truncation). The codewords are numbered from 0 and upwards. The definition of the numbering is:

Code_number = $2^{L/2}$ + INFO -1 (L/2 use division with truncation. INFO = 0 when L = 1)

Some of the first code numbers and codewords are written explicitly in the table below. As an example, for the code number 5, L = 5 and INFO = 10 (binary) = 2 (decimal)

Code number	Codewords in explicit form
0	1
1	0 0 1
2	0 1 1
3	0 0 0 0 1
4	0 0 0 1 1
5	0 1 0 0 1
6	0 1 0 1 1
7	0 0 0 0 0 0 1
8	0 0 0 0 0 1 1
9	0 0 0 1 0 0 1
10	0 0 0 1 0 1 1
11	0 1 0 0 0 0 1
.....	. . . . .

When L and INFO is known, the regular structure of the table makes it easy to create a codeword bit by bit. Similarly, a decoder may easily read bit by bit until the last "1" which gives the end of the codeword. L and INFO is then readily available. For each parameter to be coded, there is a conversion rule from the parameter value to the code number (or L and INFO). **Error! Reference source not found.** lists the connection between code number and most of the parameters used in the present coding method.

Table 2: Connection between codeword number and parameter values.

Code_number	RUN	MB_Type		Intra_pred_mode[1]		MVD	CBP		Tcoeff_chroma_DC[2]		Tcoeff_chroma_AC[2] Tcoeff_luma[2] Simple scan		Tcoeff_luma[2] Double scan	
		Intra	Inter	Prob0	Prob1		Intra	Inter	Level	Run	Level	Run	Level	Run
0	0	Intra4x4	16x16	0	0	0	47	0	EOB	-	EOB	-	EOB	-
1	1	0,0,0[3]	16x8	1	0	1	31	16	1	0	1	0	1	0
2	2	1,0,0	8x16	0	1	-1	15	1	-1	0	-1	0	-1	0
3	3	2,0,0	8x8	0	2	2	0	2	2	0	1	1	1	1
4	4	3,0,0	8x4	1	1	-2	23	4	-2	0	-1	1	-1	1
5	5	0,1,0	4x8	2	0	3	27	8	1	1	1	2	2	0
6	6	1,1,0	4x4	3	0	-3	29	32	-1	1	-1	2	-2	0
7	7	2,1,0	Intra4x4	2	1	4	30	3	3	0	2	0	1	2
8	8	3,1,0	0,0,0[3]	1	2	-4	7	5	-3	0	-2	0	-1	2
9	9	0,2,0	1,0,0	0	3	5	11	10	2	1	1	3	3	0
10	10	1,2,0	2,0,0	0	4	-5	13	12	-2	1	-1	3	-3	0
11	11	2,2,0	3,0,0	1	3	6	14	15	1	2	1	4	4	0
12	12	3,2,0	0,1,0	2	2	-6	39	47	-1	2	-1	4	-4	0
13	13	0,0,1	1,1,0	3	1	7	43	7	1	3	1	5	5	0
14	14	1,0,1	2,1,0	4	0	-7	45	11	-1	3	-1	5	-5	0
15	15	2,0,1	3,1,0	5	0	8	46	13	4	0	3	0	1	3
16	16	3,0,1	0,2,0	4	1	-8	16	14	-4	0	-3	0	-1	3
17	17	0,1,1	1,2,0	3	2	9	3	6	3	1	2	1	1	4
18	18	1,1,1	2,2,0	2	3	-9	51	9	-3	1	-2	1	-1	4
19	19	2,1,1	3,2,0	1	4	10	10	31	2	2	2	2	2	1
20	20	3,1,1	0,0,1	0	5	-10	12	35	-2	2	-2	2	-2	1
21	21	0,2,1	1,0,1	1	5	11	19	37	2	3	1	6	3	1
22	22	1,2,1	2,0,1	2	4	-11	21	42	-2	3	-1	6	-3	1
23	23	2,2,1	3,0,1	3	3	12	26	44	5	0	1	7	6	0
24	24	3,2,1	0,1,1	4	2	-12	28	33	-5	0	-1	7	-6	0
25	25		1,1,1	5	1	13	35	34	4	1	1	8	7	0
26	26		2,1,1	5	2	-13	37	36	-4	1	-1	8	-7	0
27	27		3,1,1	4	3	14	42	40	3	2	1	9	8	0
28	28		0,2,1	3	4	-14	44	39	-3	2	-1	9	-8	0
29	29		1,2,1	2	5	15	1	43	3	3	4	0	9	0
30	30		2,2,1	3	5	-15	2	45	-3	3	-4	0	-9	0
31	31		3,2,1	4	4	16	4	46	6	0	5	0	10	0
32	32			5	3	-16	8	17	-6	0	-5	0	-10	0
33	33			5	4	17	17	18	5	1	3	1	4	1
34	34			4	5	-17	18	20	-5	1	-3	1	-4	1
35	35			5	5	18	20	24	4	2	3	2	2	2
36	36					-18	24	19	-4	2	-3	2	-2	2
37	37					19	6	21	4	3	2	3	2	3
38	38					-19	9	26	-4	3	-2	3	-2	3
39	39					20	22	28	7	0	2	4	2	4
40	40					-20	25	23	-7	0	-2	4	-2	4
41	41					21	32	27	6	1	2	5	2	5
42	42					-21	33	29	-6	1	-2	5	-2	5
43	43					22	34	30	5	2	2	6	2	6
44	44					-22	36	22	-5	2	-2	6	-2	6
45	45					23	40	25	5	3	2	7	2	7
46	46					-23	38	38	-5	3	-2	7	-2	7
47	47					24	41	41	8	0	2	8	11	0
	..					..			..	..	..	..	..	..

[1] Prob0 and Prob1 defines the Intra prediction modes of two blocks relative to the prediction of prediction modes (see details in the section for Intra coding).

[2] For the entries above the horizontal line, the table is needed for relation between code number and Level/Run/EOB. For the remaining Level/Run combination there is a simple rule. The Level/Run combinations are assigned a code number according to the following priority: 1) sign of Level (+ -) 2) Run (ascending) 3) absolute value of Level (ascending).

[3.] 16x16 based intra mode. The 3 numbers refer to values for (Imode,AC,nc) - see 3.5.9.3.

5.2 Context-based Adaptive Binary Arithmetic Coding (CABAC)

5.2.1 Overview

The entropy coding method of context-based adaptive binary arithmetic coding (CABAC) has three distinct elements compared to the default entropy coding method using a fixed, universal table of variable length codes (UVLC):

1. *Context modeling* provides estimates of conditional probabilities of the coding symbols. Utilizing suitable context models, given inter-symbol redundancy can be exploited by switching between different probability models according to already coded symbols in the neighborhood of the current symbol to encode.
2. *Arithmetic codes* permit non-integer number of bits to be assigned to each symbol of the alphabet. Thus the symbols can be coded almost at their entropy rate. This is extremely beneficial for symbol probabilities much greater than 0.5, which often occur with efficient context modeling. In this case, a variable length code has to spend at least one bit in contrast to arithmetic codes, which may use a fraction of one bit.
3. *Adaptive* arithmetic codes permit the entropy coder to adapt itself to non-stationary symbol statistics. For instance, the statistics of motion vector magnitudes vary over space and time as well as for different sequences and bit-rates. Hence, an adaptive model taking into account the cumulative probabilities of already coded motion vectors leads to a better fit of the arithmetic codes to the current symbol statistics.

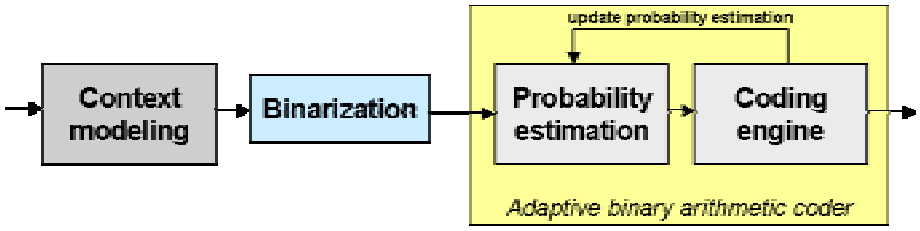


Figure 6: **Generic block diagram of CABAC entropy coding scheme**

Next we give a short overview of the main coding elements of the CABAC entropy coding scheme as depicted in Figure 6. Suppose a symbol related to an arbitrary syntax element is given, then, in a first step, a suitable model is chosen according to a set of past observations. This process of constructing a model conditioned on neighboring symbols is commonly referred to as *context modeling* and is the first step in the entropy coding scheme. The particular context models that are designed for each given syntax model are described in detail in Section 5.2.2 and Section 5.2.3. If a given symbol is non-binary valued, it will be mapped onto a sequence of binary decisions, so-called *bins*, in a second step. The actual *binarization* is done according to a given binary tree, as specified in Section 5.2.4. Finally, each binary decision is encoded with the *adaptive binary arithmetic coding* (AC) engine using the *probability estimates*, which have been provided either by the context modeling stage or by the binarization process itself. The provided models serve as a probability estimation of the related bins. After encoding of each bin, the related model will be updated with the encoded binary symbol. Hence, the model keeps track of the actual statistics.

5.2.2 Context Modeling for Coding of Motion and Mode Information

In this section we describe in detail the context modeling of our adaptive coding method for the syntax elements macroblock type (MB_type), motion vector data (MVD) and reference frame parameter (Ref_frame).

5.2.2.1 Context Models for Macroblock Type

We distinguish between MB_type for intra and inter frames. In the following, we give a description of the context models which have been designed for coding of the MB_type information in both cases. The subsequent process of mapping a non-binary valued MB_type symbol to a binary sequence in the case of inter frames will be given in detail in section 5.2.4.

5.2.2.1.1 Intra Pictures

For intra pictures, there are two possible modes for each macroblock, i.e. Intra4x4 and Intra16x16, so that signalling the mode information is reduced to transmitting a binary decision. Coding of this binary decision for a given macroblock is performed by means of context-based arithmetic coding, where the context of a current MB_type C is build by using the MB_types A and B[1] of neighboring macroblocks (as depicted in Figure 7) which are located in the causal past of the current coding event C. Since A and B are binary decisions, we define the actual context number *ctx_mb_type_intra(C)* of C by *ctx_mb_type_intra(C) = A + 2*B*, which results in four different contexts according to the 4 possible combinations of MB_type states for A and B.

In the case of MB_type Intra16x16, there are three additional parameters related to the chosen intra prediction mode, the occurrence of significant AC-coefficients and the coded block pattern for the chrominance coefficients, which have to be signaled. In contrast to the current test model, this information is not included in the mode information, but is coded separately by using distinct models as described in Section 5.2.3.

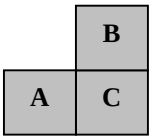
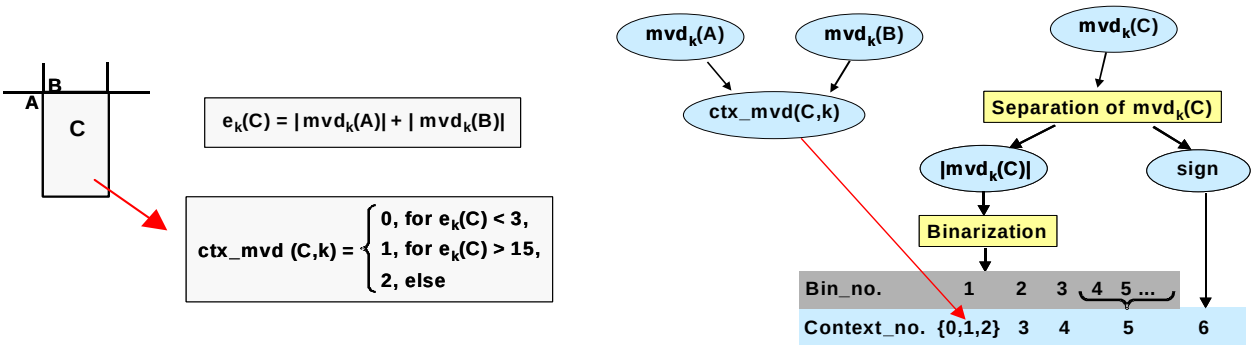


Figure 7: Neighboring symbols A and B used for conditional coding of a current symbol C.

5.2.2.1.2 P- and B-Pictures

Currently there a 10 different macroblock types for P-frames and 18 different macroblock types for B-frames, provided that the additional information of the 16x16 Intra mode is not considered as part of the mode information. Coding of a given MB_type information C is done similar to the case of intra frames by using a context model which involves the MB_type information A and B of previously encoded (or decoded) macroblocks (cp. Figure 7). However, here we only use the information whether the neighboring macroblocks of the given macroblock are of type Skip (P-frame) or Direct (B-frame), such that the actual context number *ctx_mb_type_inter(C)* is given in C-style notation for P-frame coding by *ctx_mb_type_inter(C) = ((A==Skip)?0:1) + 2*((B==Skip)?0:1)* and by *ctx_mb_type_inter(C) = ((A==Direct)?0:1) + 2*((B==Direct)?0:1)* for B-frame coding. Thus, we obtain 4 different contexts, which, however, are only used for coding of the first bin of the binarization ***b****(C)* of C, where the actual binarization of C will be performed as outlined in Section 5.2.4. For coding the second bin, a separate model is provided and for all remaining bins of ***b****(C)* two additional models are used as further explained in Sect. 5.2.3. Thus, a total number of 7 different models are supplied for coding of macroblock type information relating to P-and B-frames.

[1] For mathematical convenience the meaning of the variables A, B and C is context dependent.



(a) (b)

Figure 8: Illustration of the encoding process for a given residual motion vector component mvdk(C) of a block C: (a) Context selection rule. (b) Separation of mvdk(C) into sign and magnitude, binarization of the magnitude and assignment of context models to bin_nos.

5.2.2.2 Context Models for Motion Vector Data

Motion vector data consists of residual vectors obtained by applying motion vector prediction. Thus, it is a reasonable approach to build a model conditioned on the local prediction error. A simple measure of the local prediction error at a given block C is given by evaluating the L1-norm $e_k(A,B) = |mvd_k(A)| + |mvd_k(B)|$ of two neighboring motion vector prediction residues $mvd_k(A)$ and $mvd_k(B)$ for each component of a motion vector residue $mvd_k(C)$ of a given block, where A and B are neighboring blocks of block C, as shown in Figure 8 (a). If one of the neighboring blocks belongs to an adjacent macroblock, we take the residual vector component of the leftmost neighboring block in the case of the upper block B, and in the case of the left neighboring block A we use the topmost neighboring block. If one of the neighboring blocks is not available, because, for instance, the current block is at the picture boundary, we discard the corresponding part of e_k. By using e_k, we now define a context model *ctx_mvd(C,k)* for the residual motion vector component $mvd_k(C)$ consisting of three different context models:

$$ctx\_mvd(C,k) = \begin{cases} 0, e_k(C) < 3, \\ 1, e_k(C) > 15, \\ 2, else. \end{cases}$$

For the actual coding process, we separate $mvd_k(C)$ in sign and modulus, where only the first bin of the binarization of the modulus $|mvd_k(C)|$ is coded using the context models *ctx_mvd(C,k).* For the remaining bins, we have three additional models: two for the second and the third bin and a third model for all remaining bins. In addition, the sign coding routine is provided with a separate model. This results in a total sum of 7 different models for each vector component (see Figure 8).

In the case of B-frame coding, an additional syntax element has to be signaled when the bi-directional mode is chosen. This element represents the block size (Blk_size), which is chosen for forward or backward motion prediction. The related code number value ranges between 0 and 6 according to the 7 possible block shapes **Error! Reference source not found.**. Coding of Blk_size is done by using the binarization of the P_MB_type as described in Section 5.2.3.

5.2.2.3 Context Models for Reference Frame Parameter

If the option of temporal prediction from more than one reference frame is enabled, the chosen reference frame for each macroblock must be signaled. Given a macroblock and its reference frame parameter as a symbol C according to the definition in Sect. 3, a context model is built by using symbols A and B of the reference frame parameter belonging to the two neighboring macroblocks (cp. Figure 7). The actual context number of C is then defined by *ctx_ref_frame(C) = ((A==0)?0:1) + 2*((B==0)?0:1)*, such that *ctx_ref_frame(C)* indicates one of four models used for coding of the first bin of the binary equivalent

***b**(C)* of C. Two additional models are given for the second bin and all remaining bins of ***b**(C)*, which sums up to a total number of six different models for the reference frame information.

5.2.3 Context Modeling for Coding of Texture Information

This section provides detailed information about the context models used for the syntax elements of coded block pattern (CBP), intra prediction mode (IPRED) and (RUN, LEVEL) information.

5.2.3.1 Context Models for Coded Block Pattern

Except for MB_type Intra16x16, the context modeling for the coded block pattern is treated as follows. There are 4 luminance CBP bits belonging to 4 8x8 blocks in a given macroblock. Let C denote such a Y-CBP bit, then we define *ctx_cbp_luma(C) = A + 2*B,* where A and B are Y-CBP bits of the neighboring 8x8 blocks, as depicted in Figure 7. The remaining 2 bits of CBP are related to the chrominance coefficients. In our coding approach, these bits are translated into two dependant binary decisions, such that, in a first step, we send a bit *cbp_chroma_sig* which signals whether there are significant chrominance coefficients at all. The related context model is of the same kind as that of the Y-CBP bits, i.e. *ctx_cbp_chroma_sig(C) = A + 2*B,* where A and B are now notations for the corresponding *cbp_chroma_sig* bits of neighboring macroblocks. If *cbp_chroma_sig = 1* (non-zero chroma coefficients exist), a second bit *cbp_chroma_ac* related to the significance of AC chrominance coefficients has to be signalled. This is done by using a context model conditioned on the *cbp_chroma_ac* decisions A and B of neighboring macroblocks, such that *ctx_cbp_chroma_AC(C) = A + 2*B.* Note, that due to the different statistics there are different models for Intra and Inter macroblocks, so that the total number of different models for CBP amounts to 2*3*4=24. For the case of MB_type Intra16x16, there are three additional models, one for the binary AC decision and two models for each of the two chrominance CBP bits.

5.2.3.2 Context Models for Intra Prediction Mode

In Intra4x4 mode, coding of the intra prediction mode C of a given block is conditioned on the intra prediction mode of the previous block A to the left of C (cp. Figure 7). In fact, it is not the prediction mode number itself which is signaled and which is used for conditioning but rather its predicted order similar as it is desribed in Sect. 3.5.8. There are 6 different prediction modes and for each mode, two different models are supplied: one for the first bin of the binary equivalent of C and the other for all remaining bins. Together with two additional models for the two bits of the prediction modes of MB_type Intra16x16 (in binary representation), a total number of 14 different models for coding of intra prediction modes is given.

5.2.3.3 Context Models for Run/Level

Coding of (RUN, LEVEL) pairs is conditioned on the scanning mode, the DC/AC block type, the luminance/chrominance, and the intra/inter macroblock decision. Thus, a total number of 9 different block types are given according to Table 3, which, in turn, results in 9 different contexts. In contrast to the current test model, RUN and LEVEL are coded separately in our coding approach, as described in the following two subsections.

Ctx_run_level	**Block Type**
0	Double Scan
1	Single Scan, Inter
2	Single Scan, Intra
3	Intra16x16, DC
4	Intra16x16, AC
5	Chroma, DC, Inter
6	Chroma, DC, Intra
7	Chroma, AC, Inter
8	Chroma, AC, Intra

Table 3: Numbering of the different context models used for coding of RUN and LEVEL

5.2.3.3.1 Context-based Coding of LEVEL Information

For a given block C, the LEVEL information is first separated into sign and magnitude. According to its context *ctx_run_level(C)* four different models are chosen, where one model is used for coding of the sign information and the remaining 3 models are used for the first, the second and all remaining bins of the binarization of LEVEL. If LEVEL≠0 (EOB), the corresponding RUN is coded in a subsequent coding routine, as described in the following section.

5.2.3.3.2 Context-based Coding of RUN Information

For each context *ctx_run_level(C)* two separate models are provided for the coding of RUN; one model for the first bin and the second model for all remaining bins of the binary sequence related to RUN.

Code symbol	Binarization							
0	1							
1	0	1						
2	0	0	1					
3	0	0	0	1				
4	0	0	0	0	1			
5	0	0	0	0	0	1		
6	0	0	0	0	0	0	1	
. . .	.	.	.	.	.	.	.	.
Bin_no.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**. .**

Table 4: Binarization by means of the unary code tree

5.2.4 Binarization of Non-Binary Valued Symbols

A non-binary valued symbol will be decomposed into a sequence of binary decisions. Except for the MB_type syntax element we use the binarization given by the unary code tree in Table 4.

P_MB_type	Binarization
0	0
1	1 0 0
2	1 0 1
3	1 1 0 0 0
4	1 1 0 0 1
5	1 1 0 1 0
6	1 1 0 1 1
7	1 1 1 0 0
8	1 1 1 0 1
9	1 1 1 1 0
Bin_no	**1 2 3 4 5**

(a)

B_MB_type	Binarization
0	0
1	1 0 0
2	1 0 1
3	1 1 0 0 0
4	1 1 0 0 1
5	1 1 0 1 0
6	1 1 0 1 1
7	1 1 1 0 0 0 0
.	.
17	1 1 1 1 0 1 0
Bin_no	**1 2 3 4 5 6 7**

(b)

Table 5: (a) Binarization for P-frame MB_type and (b) for B-frame MB_type

For the binary decomposition of the MB_type symbols of P- or B-frames, which are of limited range (0 … 9) or (0 … 17) respectively, an alternative binarization is used, which is shown in Table 5.

5.2.5 Adaptive Binary Arithmetic Coding

At the beginning of the overall encoding of a given frame the probability models associated with all 126 different contexts are initialized with a pre-computed start distribution. For each symbol to encode the frequency count of the related binary decision is updated, thus providing a new probability estimate for the next coding decision. However, when the total number of occurrences of a given model exceeds a pre-defined threshold, the frequency counts will be scaled down. This periodical rescaling exponentially weighs down past observations and helps to adapt to the non-stationarity of a source. The binary arithmetic coding engine used in our presented approach is a straightforward implementation similar to that given in Witten et al., “Arithmetic Coding for Data Compression“, *Comm. of the ACM*, 30 (6), 1987, pp.520-541.

6 Test model issues

6.1 Motion Estimation and Mode Decision

6.1.1 Low-complexity mode

6.1.1.1 Finding optimum prediction mode

Both for intra prediction and motion compensated prediction, a similar loop as indicated in Figure 9 is run through. The different elements will be described below.

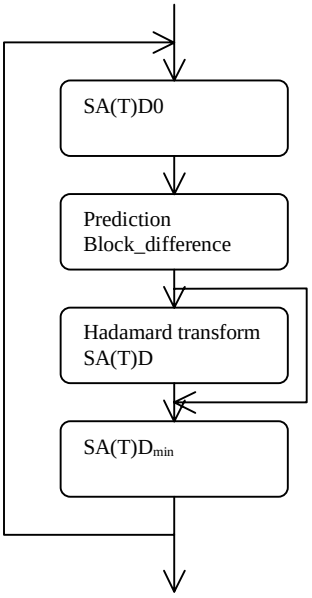


Figure 9 Loop for prediction mode decision

6.1.1.1.1 SA(T)D0

The SA(T)D to be minimised is given a 'bias' value initially in order to favour prediction modes that need few bits to be signalled. This bias is basically a parameter representing bit usage times $QP_0(QP)$

Intra mode decision: SA(T)D0 = $QP_0(QP)$xOrder_of_prediction_mode (see above)

Motion vector search: SA(T)D0 = $QP_0(QP)$x(Bits_to_code_vector + 2xcode_number_of_ref_frame)

In addition there are two special cases:

- For motion prediction of a 16x16 block with 0 vector components, 16x$QP_0(QP)$ is subtracted from SA(T)D to favour the skip mode.
- For the whole intra macroblock, 24x$QP_0(QP)$ is added to the SA(T)D before comparison with the best SA(T)D for inter prediction. This is an empirical value to prevent using too many intra blocks.

6.1.1.1.2 Block_difference

For the whole block the difference between the original and prediction is produced

Diff(i,j) = Original(i,j) - Prediction(i,j)

6.1.1.1.3 Hadamard transform

For integer pixel search (see below) we use SAD based on Diff(i,j) for decision. Hence no Hadamard is done and we use SAD instead of SATD.

$$SAD = \sum_{i,j} |Diff(i, j)|$$

However, since we will do a transform of Diff(i,j) before transmission, we will do a better optimisation if a transform is done before producing SAD. Therefore a two dimensional transform is performed in the decision loop for selecting intra modes and for fractional pixel search (see below). To simplify implementation, the Hadamard transform is chosen in this mode decision loop. The relation between pixels and basis vectors (BV) in a 4 point Hadamard transform is illustrated below (not normalized):

	Pixels →			
B	1	1	1	1
V	1	1	-1	-1
↓	1	-1	-1	1
	1	-1	1	-1

This transformation is performed horizontally and vertically and result in DiffT(i,j). Finally SATD for the block and for the present prediction mode is produced.

$$SATD = (\sum_{i,j} |DiffT(i, j)|) / 2$$

Choose the prediction mode that results in the minimum $SA(T)D_{min}$.

6.1.1.2 Encoding on macroblock level

6.1.1.2.1 Intra coding

When starting to code a macroblock, intra mode is checked first. For each 4x4 block, full coding indicated in Figure 9 is performed. At the end of this loop the complete macroblock is intra coded and a $SATD_{intra}$ is calculated.

6.1.1.2.2 Table for intra prediction modes to be used at the encoder side

Table 6 gives the table of intra prediction modes according to probability of each mode to be used on the decoder side. On the encoder side we need a sort of inverse table. Prediction modes for A and B are known as in Table 1. For the encoder we have found a Mode that we want to signal with an ordering number in the bitstream (whereas on the decoder we receive the order in the bitstream and want to convert this to a mode). Table 6 is therefore the relevant table for the encoder. Example: Prediction mode for A and B is 2. The string in Table 6 is 2 1 0 3 4 5. This indicates that prediction mode 0 has order 2 (third most probable). Prediction mode 1 is second most probable and prediction mode 2 has order 0 (most probable) etc. As in Table 1 '–' indicates that this instance can not occur because A or B or both are outside the picture.

Table 6 Prediction ordering to be used in the bitstream as a function of prediction mode (see text).

B\A	outside	0	1	2	3	4	5
outside	0-----	021---	102---	120---	012---	012---	012---
0	0---12	025314	104325	240135	143025	035214	045213
1	0---12	014325	102435	130245	032145	024315	015324
2	0---12	012345	102345	210345	132045	032415	013245
3	0---12	135024	214035	320154	143025	145203	145032
4	1---02	145203	125403	250314	245103	145203	145302
5	1---20	245310	015432	120534	245130	245301	135420

6.1.1.2.3 Inter mode selection

Next motion vector search is performed for motion compensated prediction. A search is made for all 7 possible block structures for prediction as well as from the 5 past decoded pictures. This result in 35 combinations of blocksizes and reference frames.

6.1.1.2.4 Integer pixel search

The search positions are organised in a 'spiral' structure around the predicted vector (see vector prediction). The numbering from 0 and upwords for the first positions are listed below:

```
. .  .  .  .  .
.15  9 11 13 16
.17  3  1  4 18
.19  5  0  6 20
.21  7  2  8 22
.23 10 12 14 24
```

A parameter MC_range is used as input for each sequence. To speed up the search process, the search range is further reduced:

- Search range is reduced to: Range = MC_range/2 for all block sizes except 16x16 in prediction from the most recent decoded picture.
- The range is further reduced to: Range = Range/2 for search relative to all older pictures.

After Range has been found, the centre for the spiral search is adjusted so that:

- No vector position is outside the picture.
- The (0,0) vector position is within the search range. This is done by clipping the horizontal and vertical positions of the search centre to ±Range.

6.1.1.2.5 Fractional pixel search

Fractional pixel search is performed in two steps. This is illustrated below where capital letters represent integer positions, numbers represent ½ pixel positions and lower case letters represent ¼ pixel positions.

```
A       B       C

    1   2   3

D   4   E   5   F
      a b c
    6 d 7 e 8
      f g h
G       H       I
```

Assume that the integer search points to position E. Then ½ pixel positions 1,2,3,4,5,6,7,8 are searched. Assume that 7 is the best position. Then the ¼ pixel positions a,b,c,d,e,f,g,h are searched. (Notice that by this procedure a position with 'more low pass filtering' – see 3.7.1 - is automatically checked).

After fractional pixel search has been performed for the complete macroblock, the SATD for the whole macroblock is computed: $SATD_{inter}$.

6.1.1.2.6 Decision between intra and inter

If $SATD_{intra} < SATD_{inter}$ intra coding is used. Otherwise inter coding is used.

6.1.2 High-complexity mode

6.1.2.1 Motion Estimation

For each block or macroblock the motion vector is determined by full search on integer-pixel positions followed by sub-pixel refinement.

6.1.2.1.1 Integer-pixel search

As in low-complexity mode, the search positions are organized in a spiral structure around a prediction vector. The full search range given by MC_range is used for all INTER-modes and reference frames. To speed up the search process , the prediction vector of the 16x16 block is used as center of the spiral search for all INTER-modes. Thus the SAD values for 4x4 blocks can be pre-calculated for all motion

vectors of the search range and than used for fast SAD calculation of all larger blocks. The search range is not forced to contain the (0,0)-vector.

6.1.2.1.2 Fractional pixel search

The fractional pixel search is performed as in the low-complexity case.

6.1.2.1.3 Finding the best motion vector

The integer-pixel motion search as well as the sub-pixel refinement returns the motion vector that minimizes

$$J(\mathbf{m}, \lambda_{MOTION}) = SA(T)D(s, c(\mathbf{m})) + \lambda_{MOTION} \cdot R(\mathbf{m} - \mathbf{p})$$

with $\mathbf{m} = (m_x, m_y)^T$ being the motion vector, $\mathbf{p} = (p_x, p_y)^T$ being the prediction for the motion vector, and λ_{MOTION} being the Lagrange multiplier. The rate term $R(\mathbf{m} - \mathbf{p})$ represents the motion information only and is computed by a table-lookup. The rate is estimated by using the universal variable length code (UVLC) table, even if CABAC is used as entropy coding method. For integer-pixel search, SAD is used as distortion measure. It is computed as

$$SAD(s,\ c(\mathbf{m})) = \sum_{x=1,\ y=1}^{B,\ B} \left| s[x, y] - c[x - m_x, y - m_y] \right|, \qquad B = 16, 8 \text{ or } 4\ .$$

with *s* being the original video signal and *c* being the coded video signal. In the sub-pixel refinement search, the distortion measure SATD is calculated after a Hadamard transform (see section 6.1.1.1.3). The Lagrangian multiplier λ_{MOTION} is given by

$$\lambda_{MOTION,P} = \sqrt{5} \cdot e^{QP/20} \cdot \sqrt{\frac{QP+5}{34-QP}}$$

P-frames and

$$\lambda_{MOTION,B} = 2 \cdot \sqrt{5} \cdot e^{QP/20} \cdot \sqrt{\frac{QP+5}{34-QP}}$$

for B-frames, where QP is the macroblock quantization parameter.

6.1.2.1.4 Finding the best reference frame

The determination of the reference frame *REF* and the associated motion vectors for the *NxM* inter modes in P-frames and the *FWD NxM* modes in B-frames is done after motion estimation by minimizing

$$J(REF \mid \lambda_{MOTION}) = SATD(s, c(REF, \mathbf{m}(REF))) + \lambda_{MOTION} \cdot (R(\mathbf{m}(REF) - \mathbf{p}(REF)) + R(REF))\ .$$

The rate term $R(REF)$ represents the number of bits associated with choosing REF and is computed by table-lookup using UVLC. The reference frame and block sizes for the bi-directional mode are chosen as combination of the “best” forward and backward mode.

6.1.2.2 Mode decision

6.1.2.2.1 Macroblock mode decision

The macroblock mode decision is done by minimizing the Lagrangian functional

$$J(s,c,MODE \mid QP,\lambda_{MODE}) = SSD(s,c,MODE \mid QP) + \lambda_{MODE} \cdot R(s,c,MODE \mid QP)$$

where QP is the macroblock quantizer, λ_{MODE} is the Lagrange multiplier for mode decision, and $MODE$ indicates a mode chosen from the set of potential prediction:

I-frame: $MODE \in \{INTRA4x4,\ INTRA16x16\}$,

P-frame: $MODE \in \left\{ \begin{matrix} INTRA4x4,\ INTRA16x16,\ SKIP, \\ 16x16,\ 16x8,\ 8x16,\ 8x8,\ 8x4,\ 4x8,\ 4x4 \end{matrix} \right\}$,

B-frame: $MODE \in \left\{ \begin{matrix} INTRA4x4,\ INTRA16x16,\ BIDIRECT,\ DIRECT, \\ FWD16x16,\ FWD16x8,\ FWD8x16,\ FWD8x8,\ FWD8x4, \\ FWD4x8,\ FWD4x4,\ BAK16x16,\ BAK16x8,\ BAK8x16, \\ BAK8x8,\ BAK8x4,\ BAK4x8,\ BAK4x4 \end{matrix} \right\}$.

Note that the $SKIP$ mode refers to the 16x16 mode where no motion and residual information is encoded. SSD is the sum of the squared differences between the original block s and its reconstruction c being given as

$$SSD(s,c,MODE \mid QP) = \sum_{x=1,y=1}^{16,16} \left(s_Y[x,y] - c_Y[x,y,MODE \mid QP]\right)^2 + \sum_{x=1,y=1}^{8,8} \left(s_U[x,y] - c_U[x,y,MODE \mid QP]\right)^2 + \sum_{x=1,y=1}^{8,8} \left(s_V[x,y] - c_V[x,y,MODE \mid QP]\right)^2,$$

and $R(s,c,MODE \mid QP)$ is the number of bits associated with choosing $MODE$ and QP including the bits for the macroblock header, the motion, and all DCT blocks. $c_Y[x,y,MODE \mid QP]$ and $s_Y[x,y]$ represent the reconstructed and original luminance values; c_U, c_V and s_U, s_V the corresponding chrominance values.

The Lagrangian multiplier λ_{MODE} is given by

$$\lambda_{MODE,P} = 5 \cdot e^{QP/10} \cdot \left(\frac{QP+5}{34-QP}\right)$$

for I- and P-frames and

$$\lambda_{MODE,B} = 20 \cdot e^{QP/10} \cdot \left(\frac{QP+5}{34-QP}\right)$$

for B-frames, where QP is the macroblock quantization parameter.

6.1.2.2.2 INTER 16x16 mode decision

The *INTER16x16* mode decision is performed by choosing the INTER16x16 mode which results in the minimum SATD value.

6.1.2.2.3 INTER 4x4 mode decision

For the *INTRA4x4* prediction, the mode decision for each 4x4 block is performed similar to the macroblock mode decision by minimizing

$$J(s,c,IMODE \mid QP,\lambda_{MODE}) = SSD(s,c,IMODE \mid QP) + \lambda_{MODE} \cdot R(s,c,IMODE \mid QP)$$

where QP is the macroblock quantizer, λ_{MODE} is the Lagrange multiplier for mode decision, and $IMODE$ indicates an intra prediction mode:

$$IMODE \in \{DC, HOR, VERT, DIAG, DIAG\_RL, DIAG\_LR\}.$$

SSD is the sum of the squared differences between the original 4x4 block luminance signal s and its reconstruction *c,* and $R(s,c,IMODE \mid QP)$ represents the number of bits associated with choosing *IMODE.* It includes the bits for the intra prediction mode and the DCT-coefficients for the 4x4 luminance block. The rate term is computed using the UVLC entropy coding, even if CABAC is used for entropy coding.

6.1.2.3 Algorithm for motion estimation and mode decision

The procedure to encode one macroblock s in a I-, P- or B-frame in the high-complexity mode is summarized as follows.

1. Given the last decoded frames, λ_{MODE} , λ_{MOTION} , and the macroblock quantizer QP

2. Choose intra prediction modes for the *INTRA 4x4* macroblock mode by minimizing

$$J(s,c,IMODE \mid QP,\lambda_{MODE}) = SSD(s,c,IMODE \mid QP) + \lambda_{MODE} \cdot R(s,c,IMODE \mid QP)$$

   with $IMODE \in \{DC, HOR, VERT, DIAG, DIAG\_RL, DIAG\_LR\}$.

3. Determine the best $INTRA16x16$ prediction mode by choosing the mode which results in the minimum SATD.

4. Perform motion estimation and reference frame selection by minimizing

$$J(REF,\mathbf{m}(REF) \mid \lambda_{MOTION}) = SA(T)D(s,c(REF,\mathbf{m}(REF))) + \lambda_{MOTION} \cdot (R(\mathbf{m}(REF) - \mathbf{p}(REF)) + R(REF))$$

for each reference frame and motion vector of a possible macroblock mode.

5. Choose the macroblock prediction mode by minimizing

$$J(s,c,MODE \mid QP,\lambda_{MODE}) = SSD(s,c,MODE \mid QP) + \lambda_{MODE} \cdot R(s,c,MODE \mid QP),$$

given QP and λ_{MODE} when varying $MODE$. $MODE$ indicates a mode out of the set of potential macroblock modes:

I-frame: $MODE \in \{INTRA4x4,\ INTRA16x16\}$,

P-frame: $MODE \in \begin{Bmatrix} INTRA4x4,\ INTRA16x16,\ SKIP, \\ 16x16,\ 16x8,\ 8x16,\ 8x8,\ 8x4,\ 4x8,\ 4x4 \end{Bmatrix}$,

B-frame: $MODE \in \begin{Bmatrix} INTRA4x4,\ INTRA16x16,\ BIDIRECT,\ DIRECT, \\ FWD16x16,\ FWD16x8,\ FWD8x16,\ FWD8x8,\ FWD8x4, \\ FWD4x8,\ FWD4x4,\ BAK16x16,\ BAK16x8,\ BAK8x16, \\ BAK8x8,\ BAK8x4,\ BAK4x8,\ BAK4x4 \end{Bmatrix}$.

The computation of $J(s,c,SKIP \mid QP,\lambda_{MODE})$ and $J(s,c,DIRECT \mid QP,\lambda_{MODE})$ is simple. The costs for the other macroblock modes are computed using the intra prediction modes or motion vectors and reference frames, which have been estimated in steps 2- 4.

6.2 Quantization

For each transform coefficient K the quantized LEVEL is produced in the following way:

LEVEL = (KxA(QP) + fx2^{20})/2^{20} |f| is 1/3 for intra and 1/6 for inter blocks and f has the same sign as K.

6.3 Elimination of single coefficients in inter macroblocks

6.3.1 Luma

With the small 4x4 blocks, it may happen that for instance a macroblock has only one nonzero coefficient with |Level| =1. This will probably be a very expensive coefficient and it could have been better to set it to zero. For that reason a procedure to check single coefficients have been implemented for inter luma blocks. During the quantization process, a parameter **Single_ctr** is accumulated depending on Run and Level according to the following rule:

If Level = 0 or (|Level| = 1 and Run > 5) nothing is added to **Single_ctr**.

If |Level| > 1, 9 is added to **Single_ctr**.

If |Level| = 1 and Run < 6, a value T(Run) is added to **Single_ctr**. where T(0:5) =(3,2,2,1,1,1)

If the accumulated **Single_ctr** for a 8x8 block is less than 4, all coefficients of that luma block are set to zero. Similarly, if the accumulated **Single_ctr** for the whole macroblock is less than 6, all coefficients of that luma macroblock are set to zero.

6.3.2 Chroma

A similar method to the one for luma is used. **Single_ctr** is calculated similarly for each chroma component, but for AC coefficients only and for the whole macroblock.

If the accumulated **Single_ctr** for each chroma component of a macroblock is less than 7, all the AC chroma coefficients of that component for the whole macroblock are set to zero.

7 B-pictures

7.1 Introduction

This is a first definition of B pictures to be used in TML. It is mainly intended to get started on work of testing relevant coding tools. Due to the early stage of definition, a separate description and definition of syntax elements is included in this section. In a later version of TML it is foreseen that B-frames will be fully incorporated in the remaining definition. The use of B pictures is indicated in PTYPE.

Temporal scalability is achieved using bi-directionally predicted pictures, or B pictures. The B pictures are predicted from either or both the previous and subsequent reconstructed pictures to achieve improved coding efficiency as compared to that of P pictures. The B pictures are disposable, since the B pictures are not used as reference pictures for the prediction of any other pictures. This property allows B pictures to be discarded without destroying the ability to decode the sequence and adversely affecting the quality of any subsequent pictures, thus providing temporal scalability. Figure A.1 illustrates the predictive structure with two B pictures inserted between I/P pictures.

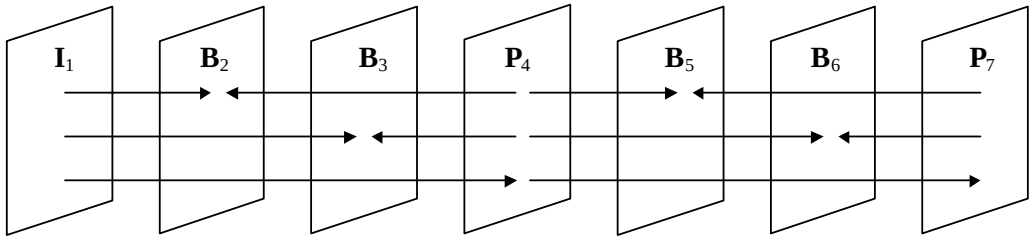


FIGURE A.1/H.26L

The location of B pictures in the bitstream is in a data-dependence order rather than in temporal order. Pictures that are dependent on other pictures shall be located in the bitstream after the pictures on which they depend. For example, as illustrated in Figure A.1, B_2 and B_3 are dependent on I_1 and P_4, and B_5 and B_6 are dependent on P_4 and P_7. Therefore the bitstream syntax order of the encoded pictures would be I_1, P_4, B_2, B_3, P_7, B_5, B_6, …. However, the display order of the decoded pictures should be I_1, B_2, B_3, P_4, B_5, B_6, P_7, …. The difference between the bitstream order of encoded pictures and the display order of decoded pictures will increase latency and memory to buffer the P pictures.

There is no limit to the number of B pictures that may be inserted between each I/P picture pair. The maximum number of such pictures may be signaled by external means (for example Recommendation H.245). The picture height, width, and pixel aspect ratio of a B picture shall always be equal to those of its temporally subsequent reference picture.

The B pictures described in this section support multiple reference frame prediction. The maximum number of previous reference frames that may be used for prediction in B pictures must be less than or equal to the number of reference frames used in the immediately following P frame, and it may be signaled by external means (for example Recommendation H.245). The use of this mode is indicated by PTYPE.

7.2 Five Prediction modes

There are five different prediction modes supported by B pictures. They are direct, forward, backward, bi-directional and the intra prediction modes. Both direct mode and bi-directional mode are bi-directional prediction. The only difference is that the bi-directional mode uses separate motion vectors for forward and backward prediction, whereas the forward and backward motion vectors of the direct mode are derived from the motion vectors used in the corresponding macroblocks of the subsequent reference frame. In the direct mode, the same number of motion vectors are used as are used in the reference macroblock for prediction. To calculate prediction blocks for the direct and bi-directional prediction mode, the forward and backward motion vectors are used to obtain appropriate blocks from reference frames and then these blocks are averaged by dividing the sum of the two prediction blocks by two.

Forward prediction means prediction from a previous reference picture, and backward prediction means prediction from a temporally subsequent reference picture.

The intra prediction means to encode the macroblock by using intra coding.

7.3 Finding optimum prediction mode

SA(T)D is initialized by a bias value to favor a prediction mode that needs few bits to be transmitted. This bias value is bit usage times $QP_0(QP)$ for the given coding mode.

For flat regions having zero motion, B pictures basically fail to make effective use of zero motion and instead are penalized in performance by selecting 16x16 intra mode. Therefore, in order to prevent assigning 16x16 intra mode to a region with little details and zero motion, SA(T)D of direct mode is subtracted by $16xQP_0(QP)$ to bias the decision toward selecting the direct mode.

And SA(T)D of 4x4 intra mode employs the same manner as section 5.1.

The calculation of SA(T)D at each mode is as follows.

- Forward prediction mode :

  $SA(T)D0 = QP_0(QP)$ x (2xcode_number_of_Ref_frame + Bits_to_code_MVDFW)

- Backward prediction mode :

  $SA(T)D0 = QP_0(QP)$ x Bits_to_code_MVDBW

- Bi-directional prediction mode :

  $SA(T)D0 = QP_0(QP)$ x (2xcode_number_of_Ref_frame + Bits_to_code_forward_Blk_size +
  Bits_to_code_backward_Blk_size +
  Bits_to_code_MVDFW +
  Bits_to_code_MVDBW)

- Direct prediction mode :

  $SA(T)D = SA(T)D - 16 \times QP_0(QP)$

- 4x4 Intra mode :

  $SA(T)D0 = QP_0(QP)$ x Order_of_prediction_mode

  $SA(T)D = SA(T)D + 24 \times QP_0(QP)$

Finally the mode with the minimum $SA(T)D_{min}$ is selected as an optimum prediction mode.

7.4 Syntax

Some additional syntax elements are needed for B pictures. The structure of B picture related fields is shown in Figure A.2. On the Ptype, two picture types shall be added to include B pictures with and without multiple reference frame prediction. On the MB_type, different macroblock types shall be defined to indicate the different prediction types for B pictures. The fields of Blk_size, MVDFW, and MVDBW shall be inserted to enable bi-directional prediction.

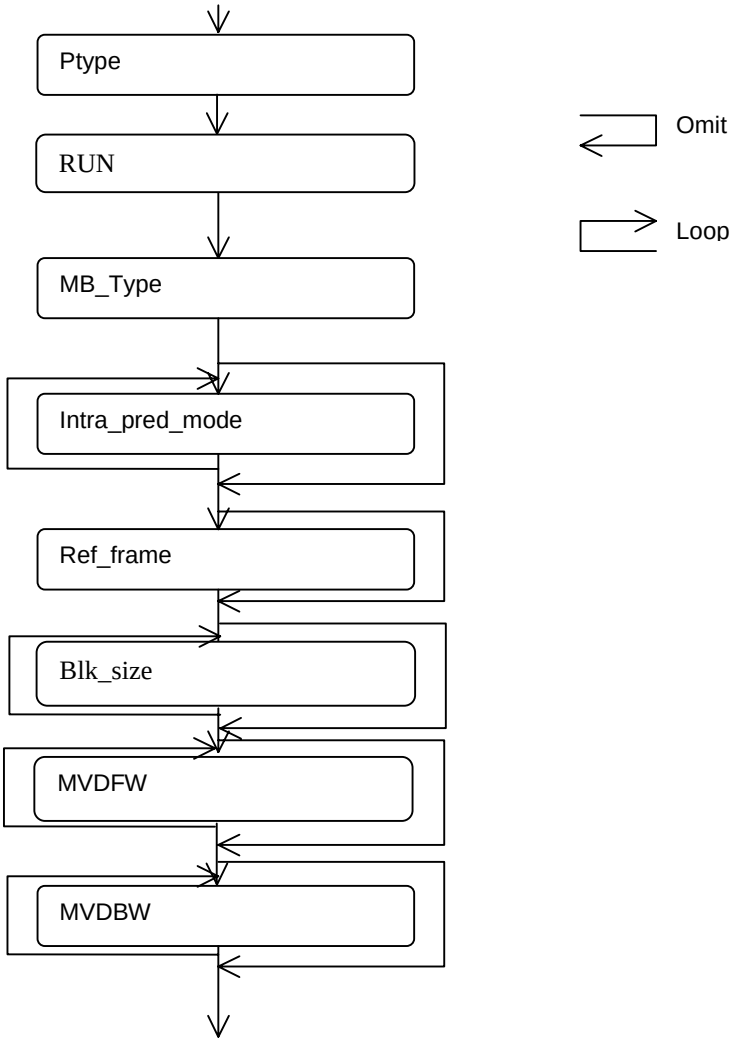


FIGURE A.2/H.26L

7.4.1 Picture type (Ptype) and RUN

See sections 3.2 and 3.3 for definitions.

7.4.2 Macro block type (MB_type)

The MB_type indicates the prediction mode and block size used to encode each macroblock. As mentioned earlier, five different prediction modes are supported by B pictures. For the forward, backward and bi-directional prediction modes, a macroblock is predicted from either or both of the previous and subsequent pictures with block size NxM. Table A.1 shows the macroblock types and the included data elements for B pictures.

In “Direct” prediction type, no motion vector data is transmitted.

The “Forward_NxM” indicates that the macroblock is prediction from a previous picture with block size NxM. The “backward_NxM” indicates that the macroblock is prediction from a subsequent picture with block size NxM. For each NxM block, motion vector data is provided. Therefore, depending on N and M, up to 16 sets of motion vector data have to be transmitted for a macroblock.

For the “Bi-directional” prediction type, the parameter Blk_size is used to indicate the block size used for forward and backward motion prediction (the Blk_size field is described in detail below). Both forward and backward motion vector data sets are transmitted. Depending on the block size indicated in Blk_size, up to 16 fields of motion vector data is transmitted for each of forward and backward prediction for a macroblock.

The “Intra_4x4” and “Intra_16x16” prediction type indicates that the macroblock is encoded by intra coding with different intra prediction modes which are defined in the same manner as section 3.4. No transmitted motion vector data is needed for intra mode.

7.4.3 Intra prediction mode (Intra_pred_mode)

As present, Intra_pred_mode indicates which intra prediction mode is used for a macroblock. Intra_pred_mode is present when Intra_4x4 prediction type is indicated in the MB_type. The code_number is same as that described in the Intra_pred_mode entry of Table 1.

7.4.4 Reference Frame (Ref_frame)

At present, Ref_frame indicates the position of the reference frame in the reference frame buffer to be used for forward motion compensated for current macroblock. Ref_frame is present only when the Ptype signals the use of multiple reference frames and only when the present MB_type indicates Forward_NxM or Bi-directional prediction type. Decoded I/P pictures are stored in the reference frame buffer in first-in-first-out manner and the most recently decoded I/P frame is always stored at position 0 in the reference frame buffer. The code_number for Ref_frame is described in Table A.2.

TABLE A.1/H.26L

MB_Type and related data elements for B pictures

0	Direct					
Code_number	Prediction Type	Intra_pred_mode	Ref_frame[1]	Blk_size	MVDFW	MVDBW
1	Forward_16x16		X		X	
2	Backward_16x16					X
3	Bi-directional		X	X	X	X
4	Forward_16x8		X		X	
5	Backward_16x8					X
6	Forward_8x16		X		X	
7	Backward_8x16					X
8	Forward_8x8		X		X	
9	Backward_8x8					X
10	Forward_8x4		X		X	
11	Backward_8x4					X
12	Forward_4x8		X		X	
13	Backward_4x8					X
14	Forward_4x4		X		X	
15	Backward_4x4					X
16	Intra_4x4	X				
17	Intra_16x16[2]					
…	…					

[1] Ref_frame is a valid field only when the usage of multiple reference frames is present in Ptype, e.g., when Ptype=4 the Ref_frame field is present.

[2] Intra_16x16 indicates 16x16 based intra mode and should represent 24 different prediction modes as defined in section 3.4.9 in Q15-J-28. For code_number greater than 16 in Table A.1, please see the code numbers from 9 and upwards in the field of inter MB_type of Table 1 in Q15-J-28 for reference.

TABLE A.2/H.26L

Code_number for ref_frame

Code_number	Reference frame
0	The most recent previous frame (1 frame back)
1	2 frames back
2	3 frames back
…	…

7.4.5 Block Size (Blk_size)

If present, Blk_size indicates which block size is used for forward and backward motion prediction in a macroblock as described in Table A.3. Blk_size is present only when Bi-directional prediction type is indicated in the MB_type. There are two sets of Blk_size data, one for forward motion vector data, and another for backward motion vector data.

TABLE A.3/H.26L

Code_number for Blk_size

Code_number	Block Size
0	1 16x16 block
1	4 8x8 blocks
2	2 16x8 blocks
3	2 8x16 blocks
4	2 8x4 blocks
5	8 4x8 blocks
6	16 4x4 blocks

7.4.6 Motion vector data (MVDFW, MVDBW)

MVDFW is the motion vector data for the forward vector, if present. MVDBW is the motion vector data for the backward vector, if present. If so indicated by MB_type or Blk_size (bi-directional prediction type only), vector data for 1-16 blocks are transmitted. The order of transmitted motion vector data is the same as that indicated in Figure2. For the code_number of motion vector data, please refer to Table 1.

7.5 Decoder Process for motion vector

7.5.1 Differential motion vectors

Motion vectors for forward, backward, or bi-directionally predicted macroblock are differentially encoded. A prediction has to be added to the motion vector differences to get the motion vectors for the macroblock. The predictions are formed in way similar to that described in section 3.6.2. The only difference is that forward motion vectors are predicted only from forward motion vectors in surrounding macroblocks, and backward motion vectors are predicted only from backward motion vectors in surrounding macroblocks.

If a neighboring macroblock does not have a motion vector of the same type or does not use the same reference frame for multiple reference frame prediction, the candidate predictor for that macroblock is set to zero for that motion vector type.

7.5.2 Motion vectors in direct mode

In direct mode the same block structure as for the macroblock in the temporally subsequent picture is assumed. For each of the subblocks the forward and backward motion vectors are computed as scaled versions of the corresponding vector components of the macroblock in the temporally subsequent picture as described below.

As the multiple reference frame prediction is used, the forward reference frame for the direct mode is the same as the one used for the corresponding macroblock in the temporally subsequent reference picture. The forward and backward motion vectors for direct mode macroblocks are calculated as follows.

$$MV_F = (TR_B * MV) / TR_D$$

$$MV_B = (TR_B - TR_D) * MV / TR_D$$

Where the vector component MV_F is the forward motion vectors, MV_B is the backward motion vector, and MV represents the motion vectors in the corresponding macroblock in the subsequent reference picture. Note that if the subsequent reference is an intra-coded frame or the reference macroblock is an intra-coded block, the motion vectors are set to zero. TR_D is the temporal distance between the temporally previous and next reference frame, and TR_B is the temporal distance between the current frame and previous reference frame.

It should be noted that when multiple reference frame prediction is used, the reference frame for the motion vector predictions is treated as though it were the most recent previous decoded frame. Thus, instead of using the temporal reference of the exact reference frame to compute the temporal distances TR_D and TR_B, the temporal reference in most recent previous reference frame is used to compute the temporal distances TR_D and TR_B.

8 Data Partitioning and Interim File Format

The traditional TML bit stream syntax simply concatenates the VLC coded symbols to form a bit stream, with the exception of the possible byte alignment of the picture (ans slice?) header. Data Partitioning re-arranges the symbols in such a way that all symbols of one data type (e.g. DC coefficients, macrobloc headers, motion vectors) that belong to a single slice are collected in one VLC coded bitstream that starts byte aligned. Decoders can process such a partitioned data streams by fetching symbols from the correct patition. The partition to fetch from is determined through the decoder's state machine, according to the syntax diagram discussed in section 2.4
In order to cleanly divide the concept of data partitioning (which is useful in all error prone environments) from network specifica, an interim file format is used that stores the compressed, VLC coded bits of each partition along with easy-to-process, uncompressed header information. It should be emphasized that this file format is not intended to be transmitted – it rather forms a clean interface between network specific and network unspecific syntax elements. Adaptation layers for IP and highly bit error prone H.223-based networks are discussed later in this document.

8.1 Data Partitioning

Data Partitioning is implemented by concatenating all VLC coded symbols of one data type and one slice (or full picture if slices are not used). At the moment, for a few partitions as indicated below contain data of more than one

data type that are so closely related that a finer diversion seems to be fruitless. The following data types are currently defined:

0	TYPE_HEADER	Picture or Slice Headers (Note 1)
1	TYPE_MBHEADER	Macroblock header information (Note 2)
2	TYPE_MVD	Motion Vector Data
3	TYPE_CBP	Coded Block Patterm
4	TYPE_2x2DC	2x2 DC Coefficients
5	TYPE_COEFF_Y	Luma AC Coefficients
6	TYPE_COEFF_C	Chroma AC Coefficients
7	TYPE_EOS	End-of-Stream Symbol

Note 1: TYPE_HEADER encompasses all Picture/Slice header information
Note 2: TYPE_MBHEADER encompasses The MB-Type, Intra-Prediction mode and Reference Frame ID.

8.2 Interim File Format

The Interim File Format consists of variable length records that include a fixed length header and the initially byte aligned, concatenated, VLC coded symbols. The fixed length header has the following structure:

int32	BitCount	Number of VLC coded bits following the fixed length header
int32	DataType	Data Type of symbols
int32	PicID	Picture ID, currently a copy of the TR
int32	SliceID	Slice ID, an integer value indicating that the informatuion belongs to the n'th slice of PicID
int32	StartMB	The starting MB (in scan order, starting at 0 ???) of that slice
int32	QP	The Quantizer Parameter (copy of the picture/slice header's QP, non-VLC coded
int32	Format	Picture Format, non-VLC coded copy of the picture header's Format field
[]byte	Data	Bitcount # of bits, padded with 0 bits in the last byte

The ordering of partitions is by picture, slice, data-type. If no symbols are coded for a given data type (which is frequently the case for Intra pictures) then that record is not present. Decoders should be designed to accept out-of-order data type records for a give Slice/Picture. Any other syntax violation in particular missing partitions, can be used to trigger syntax-based repair and/or error concealment.

9 Network Adaptation Layer for IP networks

This section covers the Network Adaptation Layer for non-managed, best effort IP networks using RTP [RFC1889] as the transport. The section will likely end up in the form of a standard's track RFC covering an RTP packetization for H.26L.

The NAL takes the information of the Interim File Format as discussed in section 8.2 and converts it into packets that can conveyed directly over RTP. It is designed to be able to take advantage of more than one virtual transport stream (either within one RTP stream by unequal packet content protection currently discussed in the IETF and as Annex I of H.323, or by using several RTP streams with network or application layer unequal error protection).

In doing so, it has to

- arrange partitions in an intelligent way into packets
- eventually split/recombine partitions to match MTU size constraints
- avoid the redundancy of (mandatory) RTP header information and information in the video stream
- define receiver/decoder reactions to packet losses. Note: the IETF tends more and more to do this in a normative way, whereas in the ITU and in MPEG this is typically left to implementers. Issue has to be discussed one day. Current document provides information without makeing any assumptions about that.

9.1 Assumptions

Any packetization scheme has to make some assumptions on typical network conditions and constraints. The following set of assumptions have been used in earlier Q.15 research on packetization and are deemed to be still valid:

- MTU size: around 1500 bytes per packet for anything but dial-up links, 500 bytes for dial-up links.

- Packet loss characteistic: non-bursty (due to drop-tail router implementations, and assuming reasonable pacing algorithms (e.g. no bursting occurs at the sender).
- Packet loss rate: up to 20%

9.2 Combining of Partitions according to Priorities

In order to allow uneqal protection of more important bits of the bitstream, exactly two packets per slice are generated (see Q15-J-53 for a detailed discussion). Slices should be used to ensure that both packets meet the MTU size constraints to avoid network splitting/recombining processes.

The 'First' packet contains the following partitions:
- TYPE_HEADER
- TYPE_MBHEADER
- TYPE_MVD
- TYPE_EOS

The 'Second' packet is assembled using the rest of the partitions
- TYPE_CBP Coded Block Patterm
- TYPE_2x2DC 2x2 DC Coefficients
- TYPE_COEFF_Y Luma AC Coefficients
- TYPE_COEFF_C Chroma AC Coefficients

This configuration allows decoding the first packet independently from the second (although not vice versa). As the first packet is more important both because motion information is important for struction [for what?] and because the 'First' packet is necessary to decode the 'Second', UEP should be used to protect the 'First' packet s higher.

9.3 Packet Structure

Each packet consists of a fixed 32 bit header a series of Part-of-Partition structures (POPs).

The packet header contains information

Bit 31	1 == This packet contains a picture header
Bit 30	1 == This packet contains a slice header
Bits 25..29	Reserved
Bits 10-24	StartMB. It is assumed that no picture has more than 2**14 Macroblocks
Bits 0..9	SliceID. It is assumed that a picture does not have more than 1024 slices

Note: The PictureID (TR) can be easily reconstructed from the RTP Timestamp and is therefore not coded again.
Note: The current software reconstructs QP and Format out of the VLC coded Picture/Slice header symbols. This is architecturally not nice and should be changed, probably by deleting these two values from the interim File Format.
Note: This header is likely to change once bigger picture formats etc. come into play.

Each Part-of-Partition structure contains a header of 16 bits whose format is as follows

Bits 15..12	Data Type
Bits 11..0	Length of VLC-coded POP payload (in bits, starting byte-aligned, 0 indicates 4096 bits of paylaod)

Th reasoning behind the introduction of POP packets lies in avoiding large fixed length headers for (typically) small partitions. See Q15-J-53.

9.4 Packetization Process

The packetization process converts the Interim File Format (or, in a real world system, data partitioned symbols arriving through a software interface) into packets. The following RTP header fields are used (see RFC1889 for exact semantics):
- Timestamp: is calculated according to the rules of RFC1889 and RFC2429 based on a 90 KHz timestamp.
- Marker Bit: set for the very last packet of a picture ('Second' packet of the last Slice), otherwise cleared.
- Sequence Number is increased by one for every generated packet, and starts with 0 for easier debugging (this in contrast to RFC1889, where a random initialization is mandatory for security purposes).
- Version (V): 2
- Padding (P): 0
- Extension (X): 0

- Csourcecount (CC): 0
- Payload Type (PT): 0 (This in contrast to RFC1889 where 0 is forbidden)

The RTP header is followed by the payload, which follows the packet structure of section 9.3.

The RTP packet file, used as the input to packet loss simulators and similar tools (note that this format is identical to the ones used for IP-related testing during the H.263++ project, so that the loss simulators, error paytterns etc, can be re-used):

- Int32 size of the following packet in bytes
- []byte packet content, starting with the RTP header.

9.5 De-packetization

The De-packetization process reconstructs a file in the Interim File Format from an RTP packet file (that is possible subject to packet losses). This task is straightforwad and reverse to the packetization process. (Note that the QP and Format fields curretly have to be reconstructed using the VLC-coded symbols in the TYPE_HEADER partition. This bug in the Interim File Format spec should be fixed some time).

9.6 Repair and Error Concealment

In order to take advantage of potential UEP for the 'First' packet and the ability of the decoder to reconstruct data where CBP/coefficient information was lost, a very simple error concealment strategy is used. This strategy repairs the bitstream by replacing a lost CBP partition with CBPs that indicate no coded coefficients. Unfortunately, the CBP codewords for Intra and Inter blocks are different, so that such a repair cannot be done context-insensitive. Instead of (partly) VLC-decoding the CBP partition in the NAL module in order to insert the correct type of CBP symbol in the (lost) partition, the decoder itself can be changed to report the appropriate CBP symbols saying "No coefficients" whenever the symbol fetch for a CBP symbol returns with the indication of a lost/empty partition.