Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG Document **JVT-A003r1**
File: JVT-A003r1.doc
Pattaya, Thailand, 3-7 December 2001 Generated: 2002-01-18

Title: Joint Model Number 1, Revision 1(JM-1r1)

Status: Approved Output Document

Contact: Thomas Wiegand
Heinrich Hertz Institute (HHI), Einsteinufer 37, D-10587 Berlin, Germany
Tel: +49 - 30 - 31002 617, Fax: +49 - 030 - 392 72 00, wiegand@hhi.de

Purpose: Information

This document presents the Joint Model Number 1 of the Joint Video Team formed by ITU-T SG16 Q.6 (VCEG) and ISO/IEC JTC 1/SC 29/WG 11 (MPEG). This document is a description of a reference coding method to be used for the development of a new video compression ITU-T Recommendation (H.26L) and ISO Standard (MPEG-4, Part 10).

It contains various corrections and improvements relative to TML-9 based on suggestions from the JVT experts reflector.

In addition to JVT-A003, this version contains improvements regarding the Slice header description contributed by Karsten Sühring, HHI. This contribution provides a description that is much closer to the implementation in the TML code. However, due to time restrictions, this description has not been fully completed and needs further work for which it serves as an excellent starting point.

CONTENTS

1 Scope

This document is a description of a reference coding method to be used for the development of a new video compression ITU-T Recommendation (H.26L) and ISO Standard (MPEG-4, Part 10). The basic configuration of the algorithm is similar to H.263 and MPEG-4, Part 2.

2 Source Coder

2.1 Picture formats

The image width and height of the source data are restricted to be multiples of 16. At the moment only colour sequences using 4:2:0 chrominance sub-sampling are supported.

2.2 Subdivision of a picture into macroblocks

Pictures are divided into macroblocks of 16x16 pixels. For instance, a QCIF picture is divided into 99 macroblocks as indicated in FIGURE 1. A number of consecutive macroblocks in coding order (see FIGURE 1) can be organized in slices. Slices represent independent coding units in a way that they can be decoded without referencing other slices of the same frame.

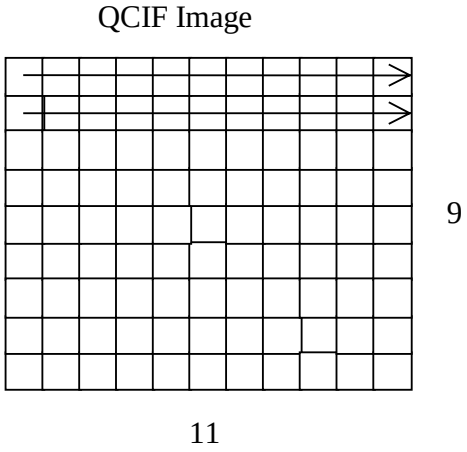


FIGURE 1

Subdivision of a QCIF picture into 16x16 macroblocks

2.3 Order of the bitstream within a macroblock

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

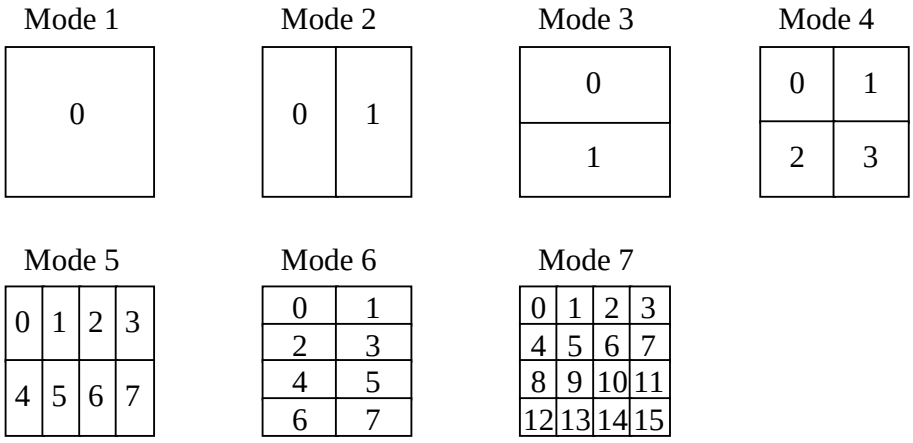
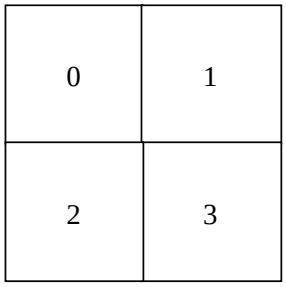


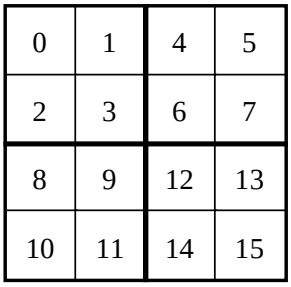
FIGURE 2

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

CBPY 8x8 block order



Luma residual coding 4x4 block order



Chroma residual coding 4x4 block order

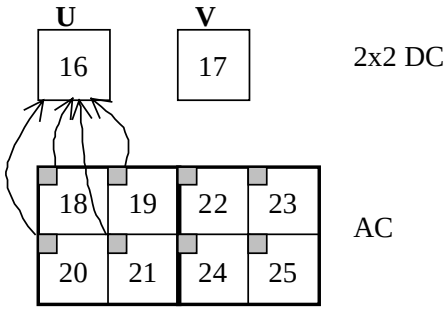


FIGURE 3

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

3 Syntax and Semantics

3.1 Organization of syntax elements (NAL concept)

Tbd.

3.1.1 Slice Mode

Tbd.

3.1.2 Data Partitioning Mode

Data Partitioning re-arranges the symbols in a way that all symbols of one data type (e.g. DC coefficients, macroblock 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 partition. The partition to fetch from is determined through the decoder's state machine, according to the syntax diagram discussed in section 3.4.

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	Luminance AC Coefficients
6	TYPE_COEFF_C	Chrominance 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.

3.2 Syntax diagrams

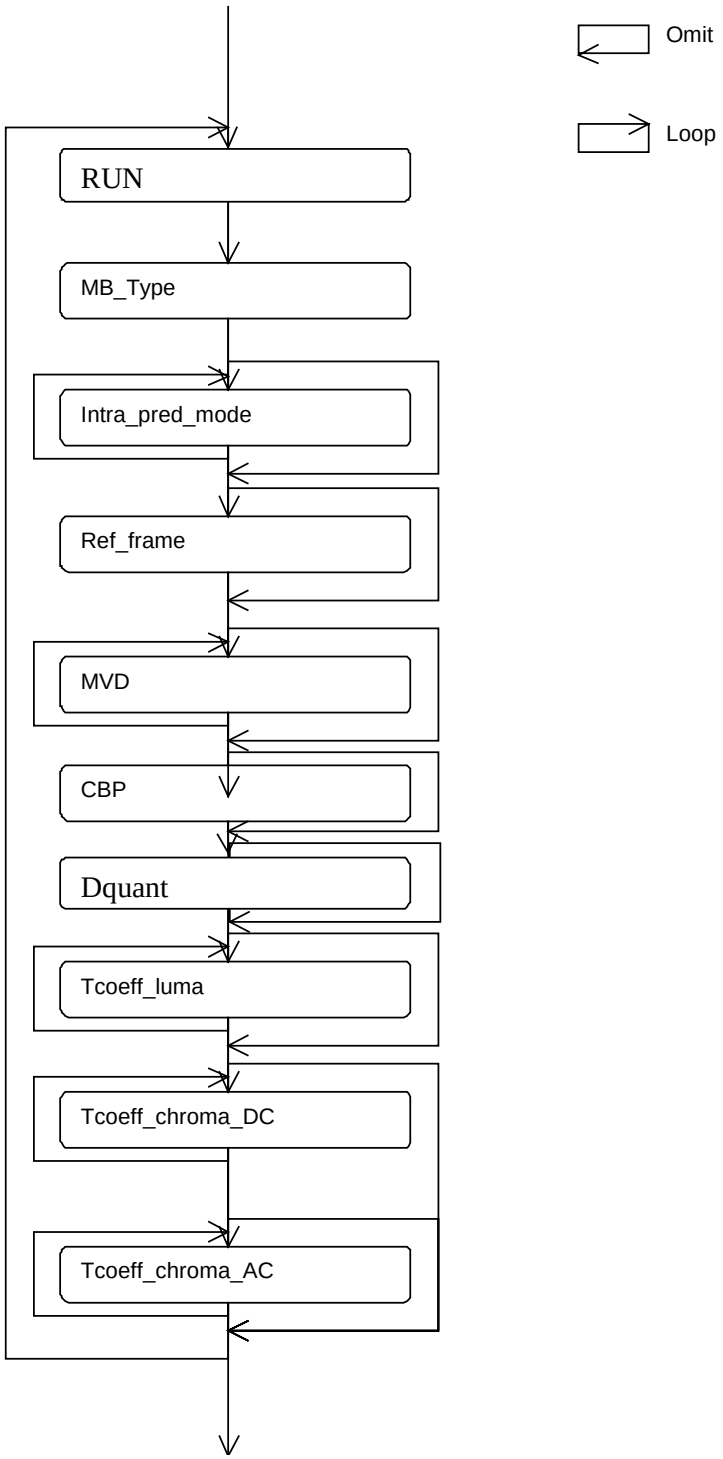


FIGURE 4

Syntax diagram for all the elements in the macroblock layer

3.3 Slice Layer

Editor: The slice layer needs a significant amount of work together with the NAL concept. The Picture sync and picture type codewords are present to ease the VCL development.

The information on the slice layer is coded depending on the NAL type. The coding, the order and even the presence of some fields may differ between the different NALs.

3.3.1 Slicesync

A slice sync may be inserted as the first codeword if it is needed by the NAL. If UVLC codes are used for the transmission of the slice level it should be a 31 bit long (L = 31) codeword with the INFO part set to zero.

3.3.2 Temporal reference (TRType/TR)

Two codewords. Temporal reference type (TRType) indicates how TR is transmitted

TRType = 0 Absoltue TR

TRType <> 0 Error

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.

3.3.3 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 signalled for each macroblock.

3.3.4 Size information

A series of up to three codewords. The first codeword indicates a size change. If set to zero the size is unchaged otherwise (if set to one) it is followed by two codewords containing the new width and height.

3.3.5 Reference Picture ID

3.3.6 First MB in slice

The number of the first macroblock contained in this slice.

3.3.7 Slice QP (SQP)

Information about the quantizer QUANT to be used for luminance for the picture. (See under Quantization concerning QUANT for chrominance). 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: $QUANTH.263(QP) \approx QP0(QP) = 2QP/6$. QP0() will be used later for scaling purposes when selecting prediction modes.

3.3.8 SP Slice QP

For SP frames the SP slice QP is transmitted.

3.3.9 Number of macroblocks in slice

For CABAC entropy coding the number of macroblocks contained in the slice is transmitted.

3.3.10 Picture Number (PN)

PN shall be incremented by 1 for each coded and transmitted picture, in modulo MAX_PN operation, relative to the PN of the previous stored picture. For non-stored pictures, PN shall be incremented from the value in the most temporally recent stored picture, which precedes the non-stored picture in bitstream order.

The PN serves as a unique ID for each picture stored in the multi-picture buffer within MAX_PN coded and stored pictures. Therefore, a picture cannot be kept in the buffer after more than MAX_PN-1 subsequent coded and stored pictures unless it has been assigned a long-term picture index as specified below. The encoder shall ensure that the bitstream shall not specify retaining any short-term picture after more than MAX_PN-1 subsequent stored pictures. A decoder which encounters a picture number on a current picture having a value equal to the picture number of some other short-term stored picture in the multi-picture buffer should treat this condition as an error.

3.3.11 Reference Picture Selection Layer (RPSL)

RPSL can be signaled with the following values:

Code number 0: The RPS layer is not sent,

Code number 1: The RPS layer is sent.

If RPSL is not sent, the default buffer indexing order presented in the next subsection shall be applied. RPS layer information sent at the slice level does not affect the decoding process of any other slice.

If RPSL is sent, the buffer indexing used to decode the current slice and to manage the contents of the picture buffer is sent using the following code words.

3.3.12 Re-Mapping of Picture Numbers Indicator (RMPNI)

RMPNI is present in the RPS layer if the picture is a P or B picture. RMPNI indicates whether any default picture indices are to be re-mapped for motion compensation of the current slice – and how the re-mapping of the relative indices into the multi-picture buffer is to be specified if indicated. If RMPNI indicates the presence of an ADPN or LPIR field, an additional RMPNI field immediately follows the ADPN or LPIR field.

A picture reference parameter is a relative index into the ordered set of pictures. The RMPNI, ADPN, and LPIR fields allow the order of that relative indexing into the multi-picture buffer to be temporarily altered from the default index order for the decoding of a particular slice. The default index order is for the short-term pictures (i.e., pictures which have not been given a long-term index) to precede the long-term pictures in the reference indexing order. Within the set of short-term pictures, the default order is for the pictures to be ordered starting with the most recent buffered reference picture and proceeding through to the oldest reference picture (i.e., in decreasing order of picture number in the absence of wrapping of the ten-bit picture number field). Within the set of long-term pictures, the default order is for the pictures to be ordered starting with the picture with the smallest long-term index and proceeding up to the picture with long-term index equal to the most recent value of MLIP1−1.

For example, if the buffer contains three short-term pictures with short-term picture numbers 300, 302, and 303 (which were transmitted in increasing picture-number order) and two long-term pictures with long-term picture indices 0 and 3, the default index order is:

default relative index 0 refers to the short-term picture with picture number 303,

default relative index 1 refers to the short-term picture with picture number 302,

default relative index 2 refers to the short-term picture with picture number 300,

default relative index 3 refers to the long-term picture with long-term picture index 0, and

default relative index 4 refers to the long-term picture with long-term picture index 3.

The first ADPN or LPIR field that is received (if any) moves a specified picture out of the default order to the relative index of zero. The second such field moves a specified picture to the relative index of one, etc. The set of remaining pictures not moved to the front of the relative indexing order in this manner shall retain their default order amongst themselves and shall follow the pictures that have been moved to the front of the buffer in relative indexing order.

If there is not more than one reference picture used, no more than one ADPN or LPIR field shall be present in the same RPS layer unless the current picture is a B picture. If the current picture is a B picture and more than one reference picture is used, no more than two ADPN or LPIR fields shall be present in the same RPS layer.

Any re-mapping of picture numbers specified for some slice shall not affect the decoding process for any other slice.

In a P picture an RMPNI "end loop" indication is followed by RPBT.

Within one RPS layer, RMPNI shall not specify the placement of any individual reference picture into more than one re-mapped position in relative index order.

TABLE 1

RMPNI operations for re-mapping of reference pictures

Code Number	Re-mapping Specified
0	ADPN field is present and corresponds to a negative difference to add to a picture number prediction value
1	ADPN field is present and corresponds to a positive difference to add to a picture number prediction value
2	LPIR field is present and specifies the long-term index for a reference picture
3	End loop for re-mapping of picture relative indexing default order

3.3.12.1 Absolute Difference of Picture Numbers (ADPN)

ADPN is present only if indicated by RMPNI. ADPN follows RMPNI when present. The code number of the UVLC corresponds to ADPN – 1. ADPN represents the absolute difference between the picture number of the currently re-mapped picture and the prediction value for that picture number. If no previous ADPN fields have been sent within the current RPS layer, the prediction value shall be the picture number of the current picture. If some previous ADPN field has been sent, the prediction value shall be the picture number of the last picture that was re-mapped using ADPN.

If the picture number prediction is denoted PNP, and the picture number in question is denoted PNQ, the decoder shall determine PNQ from PNP and ADPN in a manner mathematically equivalent to the following:

```
if (RMPNI == '1') {   // a negative difference
  if (PNP – ADPN < 0)
    PNQ = PNP – ADPN + MAX_PN;
  else
    PNQ = PNP – ADPN;
}else{            // a positive difference
  if (PNP + ADPN > MAX_PN-1)
    PNQ  = PNP + ADPN – MAX_PN;
  else
    PNQ  = PNP + ADPN;
```

```
}
```

The encoder shall control RMPNI and ADPN such that the decoded value of ADPN shall not be greater than or equal to MAX_PN.

As an example implementation, the encoder may use the following process to determine values of ADPN and RMPNI to specify a re-mapped picture number in question, PNQ:

```
DELTA = PNQ – PNP;
if (DELTA < 0) {
  if (DELTA < –MAX_PN/2-1)
    MDELTA = DELTA + MAX_PN;
  else
    MDELTA = DELTA;
}else{
  if(DELTA > MAX_PN/2)
    MDELTA = DELTA – MAX_PN;
  else
    MDELTA = DELTA;
}

ADPN = abs(MDELTA);
```

where abs() indicates an absolute value operation. Note that the code number of the UVLC corresponds to the value of ADPN – 1, rather than the value of ADPN itself.

RMPNI would then be determined by the sign of MDELTA.

3.3.12.2 Long-term Picture Index for Re-Mapping (LPIR)

LPIR is present only if indicated by RMPNI. LPIR follows RMPNI when present. LPIR is transmitted using UVLC codewords. It represents the long-term picture index to be re-mapped. The prediction value used by any subsequent ADPN re-mappings is not affected by LPIR.

3.3.13 Reference Picture Buffering Type (RPBT)

RPBT specifies the buffering type of the currently decoded picture. It follows an RMPNI “end loop” indication when the picture is not an I picture. It is the first element of the RPS layer if the picture is an I picture. The values for RPBT are defined as follows:

Code number 0: Sliding Window,

Code number 1: Adaptive Memory Control.

In the “Sliding Window” buffering type, the current decoded picture shall be added to the buffer with default relative index 0, and any marking of pictures as “unused” in the buffer is performed automatically in a first-in-first-out fashion among the set of short-term pictures. In this case, if the buffer has sufficient “unused” capacity to store the current picture, no additional pictures shall be marked as “unused” in the buffer. If the buffer does not have sufficient “unused” capacity to store the current picture, the picture with the largest default relative index among the short-term pictures in the buffer shall be marked as “unused”. In the sliding window buffering type, no additional information is transmitted to control the buffer contents.

In the "Adaptive Memory Control" buffering type, the encoder explicitly specifies any addition to the buffer or marking of data as “unused” in the buffer, and may also assign long-term indices to short-term pictures. The current picture and other pictures may be explicitly marked as “unused” in the buffer, as specified by the encoder. This buffering type requires further information that is controlled by memory management control operation (MMCO) parameters.

3.3.13.1 Memory Management Control Operation (MMCO)

MMCO is present only when RPBT indicates “Adaptive Memory Control”, and may occur multiple times if present. It specifies a control operation to be applied to manage the multi-picture buffer memory. The

MMCO parameter is followed by data necessary for the operation specified by the value of MMCO, and then an additional MMCO parameter follows – until the MMCO value indicates the end of the list of such operations. MMCO commands do not affect the buffer contents or the decoding process for the decoding of the current picture – rather, they specify the necessary buffer status for the decoding of subsequent pictures in the bitstream. The values and control operations associated with MMCO are defined in TABLE 2.

If MMCO is Reset, all pictures in the multi-picture buffer (but not the current picture unless specified separately) shall be marked "unused" (including both short-term and long-term pictures).

The picture height and width shall not change within the bitstream except within a picture containing a Reset MMCO command.

A "stored picture" does not contain an MMCO command in its RPS layer which marks that (entire) picture as "unused". If the current picture is not a stored picture, its RPS layer shall not contain any of the following types of MMCO commands:

- An Reset MMCO command,
- Any MMCO command which marks any other picture (other than the current picture) as "unused" that has not also been marked as "unused" in the ERPS layer of a prior stored picture, or
- Any MMCO command which assigns a long-term index to a picture that has not also been assigned the same long-term index in the ERPS layer of a prior stored picture

TABLE 2

Memory Management Control Operation (MMCO) Values

Code Number	Memory Management Control Operation	Associated Data Fields Following
0	End MMCO Loop	None (end of ERPS layer)
1	Mark a Short-Term Picture as "Unused"	DPN
2	Mark a Long-Term Picture as "Unused"	LPIN
3	Assign a Long-Term Index to a Picture	DPN and LPIN
4	Specify the Maximum Long-Term Picture Index	MLIP1
5	Reset	None

3.3.13.2 Difference of Picture Numbers (DPN)

DPN is present when indicated by MMCO. DPN follows MMCO if present. DPN is transmitted using UVLC codewords and is used to calculate the PN of a picture for a memory control operation. It is used in order to assign a long-term index to a picture, mark a short-term picture as "unused". If the current decoded picture number is PNC and the decoded UVLC code number is DPN, an operation mathematically equivalent to the following equations shall be used for calculation of PNQ, the specified picture number in question:

```
if (PNC – DPN < 0)
  PNQ = PNC – DPN + MAX_PN;
else
  PNQ = PNC – DPN;
```

Similarly, the encoder may compute the DPN value to encode using the following relation:

```
if (PNC – PNQ < 0)
  DPN = PNC – PNQ + MAX_PN;
else
  DPN = PNC – PNQ;
```

For example, if the decoded value of DPN is zero and MMCO indicates marking a short-term picture as “unused”, the current decoded picture shall be marked as “unused” (thus indicating that the current picture is not a stored picture).

3.3.13.3 Long-term Picture Index (LPIN)

LPIN is present when indicated by MMCO. LPIN specifies the long-term picture index of a picture. It follows DPN if the operation is to assign a long-term index to a picture. It follows MMCO if the operation is to mark a long-term picture as “unused”.

3.3.13.4 Maximum Long-Term Picture Index Plus 1 (MLIP1)

MLIP1 is present if indicated by MMCO. MLIP1 follows MMCO if present. If present, MLIP1 is used to determine the maximum index allowed for long-term reference pictures (until receipt of another value of MLIP1). The decoder shall initially assume MLIP1 is "0" until some other value has been received. Upon receiving an MLIP1 parameter, the decoder shall consider all long-term pictures having indices greater than the decoded value of MLIP1 – 1 as “unused” for referencing by the decoding process for subsequent pictures. For all other pictures in the multi-picture buffer, no change of status shall be indicated by MLIP1.

3.3.14 Supplemental Enhancement Information

Supplemental enhancement information (SEI) is encapsulated into chunks of data separate from coded slices, for example. It is up to the network adaptation layer to specify the means to transport SEI chunks. Each SEI chunck may contain one or more SEI messages. Each SEI message shall consist of a SEI header and SEI payload. The SEI header starts at a byte-aligned position from the first byte of a SEI chunk or from the first byte after the previous SEI message. The SEI header consists of two codewords, both of which consist of one or more bytes. The first codeword indicates the SEI payload type. Values from 00 to FE shall be reserved for particular payload types, whereas value FF is an escape code to extend the value range to yet another byte as follows:

```
payload_type = 0;
for (;;) {
  payload_type += *byte_ptr_to_sei;
  if (*byte_ptr_to_sei < 0xFF)
    break;
  byte_ptr_to_sei++;
}
```

The second codeword of the SEI header indicates the SEI payload size in bytes. SEI payload size shall be coded similarly to the SEI payload type.

The SEI payload may have a SEI payload header. For example, a payload header may indicate to which picture the particular data belongs. The payload header shall be defined for each payload type separately.

3.4 Macroblock layer

Following the syntax diagram for the macroblock elements, the various elements are described.

3.4.1 Number of Skipped Macroblocks (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 collocated picture material from the last decoded frame. For a B macroblock skip means direct mode without coefficients. RUN indicates the number of skipped macroblocks in an inter- or B-picture before a coded macroblock. If a picture or slice ends with one or more skipped macroblocks, they are represented by an additional RUN which counts the number of skipped macroblocks.

3.4.2 Macro block type (MB_Type)

Refer to TABLE 7. There are different MB-Type tables for Intra and Inter frames.

3.4.2.1 Intra Macroblock modes

Intra 4x4	Intra coding as defined in sections 0 to 0.
Imode, nc, AC	See definition in section 0. These modes refer to 16x16 intra coding.

3.4.2.2 Inter Macroblock Modes

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.

3.4.3 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 labeled a to p below). The pixels A to I from neighboring 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

FIGURE 5

Syntax diagram for the Picture header.

3.4.3.1.1 Prediction for Intra-4x4 mode for luminance blocks

For the luminance signal, there are 6 intra prediction modes labeled 0 to 5. Mode 0 is ‘DC-prediction’ (see below). The other modes represent directions of predictions as indicated below.

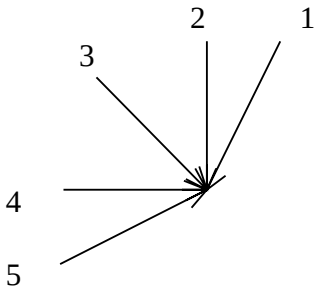


FIGURE 6

Syntax diagram for the Picture header.

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.

Mode 1: Vertical/Diagonal Prediction

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

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.

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

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.

Mode 5: Horizontal/Diagonal prediction

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

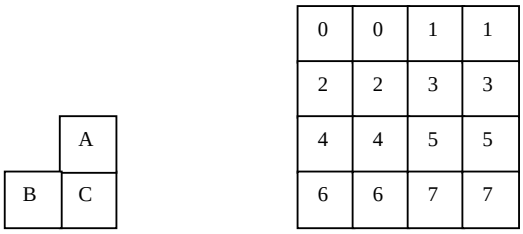
a is predicted by:	(E+F)/2
b is predicted by	F
c,e are predicted by	(F+G)/2
f,d are predicted by	G
i,g are predicted by	(G+H)/2
h,j,k,l,m,n,o,p are predicted by	H

Coding of Intra 4x4 prediction modes

Since each of the 4x4 luminance 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 7a. 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. . When an adjacent block is coded by 16x16 intra mode, prediction mode is “mode 0: DC_prediction”; when it is coded in inter mode, prediction mode is “mode 0: DC_prediction” in the usual case and “outside” in the case of constrained intra update. This ordering is listed in TABLE 3.

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 TABLE 7) 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 cannot occur because A or B or both are outside the picture.

For more efficient coding, information on intra prediction of two 4x4 luminance blocks are coded in one codeword (Prob0 and Prob1 in TABLE 7). The order of the resulting 8 codewords is indicated in FIGURE 7b.



a

FIGURE 7

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 3

Prediction mode as a function of ordering signaled 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

3.4.3.1.2 Prediction for Intra-16x16 mode for luminance

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 Luminance macroblock. We have 4 different prediction modes:

Mode 0 (vertical)

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

Mode 1 (horizontal)

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

Mode 2 (DC prediction)

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

where only the average of 16 pixels are used when the other 16 pixels are outside the picture or slice. If all 32 pixels are outside the picture, the prediction for all pixels in the block is 128.

Mode 3 (Plane prediction)

Pred(i,j) = max(0, min(255, (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

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

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

Residual coding

The residual coding is based on 4x4 transform. But similar to coding of chrominance 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.

Coding of mode information for Intra-16x16 mode

See TABLE 7. 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 chrominance (see 3.4.6)

3.4.3.1.3 Prediction in intra coding of chrominance blocks

For chrominance prediction there is only one mode. No information is therefore needed to be transmitted. The prediction is indicated in the figure below. The 8x8 chrominance 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)

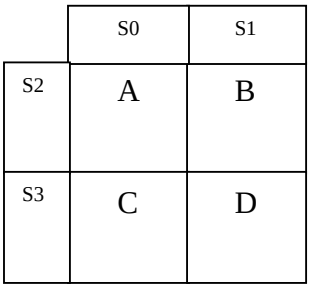


FIGURE 8

Prediction of Chrominance blocks.

3.4.4 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
..	..

3.4.5 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. All fractional pixel positions shall be interpolated as described below. If a pixel referred in the interpolation process (necessarily integer accuracy) is outside of the reference frame it shall be replaced by the nearest pixel belonging to the frame (an edge or corner pixel). Reconstructed motion vectors shall be clipped to +-19 integer pixels outside of the frame.

3.4.5.1 Prediction of vector components

In all macroblocks with square motion vector blocks (16x16, 8x8, 4x4) "median prediction" (see 3.4.5.1.1) is used. In case the macroblock may be classified to have directional segmentation the prediction is defined in 3.4.5.1.2.

3.4.5.1.1 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

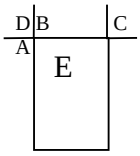


FIGURE 9

Median prediction of motion vectors.

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.

3.4.5.1.2 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.

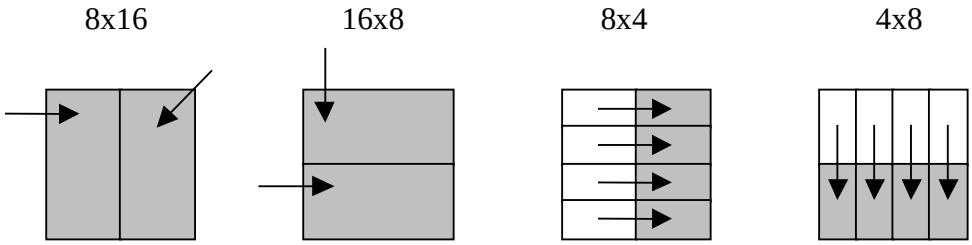


FIGURE 10

Directional segmentation prediction

3.4.5.2 Chrominance vectors

Chrominance vectors are derived from the Luminance vectors. Since chrominance has half resolution compared to Luminance, the chrominance vectors are obtained by a division of two:

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

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

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

3.4.6 Coded Block Pattern (CBP)

The CBP contains information of which 8x8 blocks - Luminance and chrominance - 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 luminance 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 chrominance we define 3 possibilities:

nc=0:	no chrominance coefficients at all.
nc=1	There are nonzero 2x2 transform coefficients. All chrominance AC coefficients = 0. Therefore we do not send any EOB for chrominance AC coefficients.
nc=2	There may be 2x2 nonzero coefficients and there is at least one nonzero chrominance 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 chrominance 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.

3.4.7 Change of Quantizer Value (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

The value of Dquant 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 Decoding Process

4.1 Slice Decoding

4.1.1 Multi-Picture Decoder Process

The decoder stores the reference pictures for inter-picture decoding in a multi-picture buffer. The decoder replicates the multi-picture buffer of the encoder according to the reference picture buffering type and any memory management control operations specified in the bitstream. The buffering scheme may also be operated when partially erroneous pictures are decoded.

Each transmitted and stored picture is assigned a Picture Number (PN) which is stored with the picture in the multi-picture buffer. PN represents a sequential picture counting identifier for stored pictures. PN is constrained, using modulo MAX_PN arithmetic operation. For the first transmitted picture, PN should be "0". For each and every other transmitted and stored picture, PN shall be increased by 1. If the difference (modulo MAX_PN) of the PNs of two consecutively received and stored pictures is not 1, the decoder should infer a loss of pictures or corruption of data. In such a case, a back-channel message indicating the loss of pictures may be sent to the encoder.

Besides the PN, each picture stored in the multi-picture buffer has an associated index, called the default relative index. When a picture is first added to the multi-picture buffer it is given default relative index 0 – unless it is assigned to a long-term index. The default relative indices of pictures in the multi-picture buffer are modified when pictures are added to or removed from the multi-picture buffer, or when short-term pictures are assigned to long-term indices.

The pictures stored in the multi-picture buffers can also be divided into two categories: long-term pictures and short-term pictures. A long-term picture can stay in the multi-picture buffer for a long time (more than MAX_PN-1 coded and stored picture intervals). The current picture is initially considered a short-term picture. Any short-term picture can be changed to a long-term picture by assigning it a long-term index according to information in the bitstream. The PN is the unique ID for all short-term pictures in the multi-picture buffer. When a short-term picture is changed to a long-term picture, it is also assigned a long-term picture index (LPIN). A long-term picture index is assigned to a picture by associating its PN to an LPIN. Once a long-term picture index has been assigned to a picture, the only potential subsequent use of the long-term picture's PN within the bitstream shall be in a repetition of the long-term index assignment. The PNs of the long-term pictures are unique within MAX_PN transmitted and stored pictures. Therefore, the PN of a long-term picture cannot be used for assignment of a long-term index after MAX_PN-1 transmitted subsequent stored pictures. LPIN becomes the unique ID for the life of a long-term picture.

PN (for a short-term picture) or LPIN (for a long-term picture) can be used to re-map the pictures into re-mapped relative indices for efficient reference picture addressing.

4.1.1.1 Decoder Process for Short/Long-term Picture Management

The decoder may have both long-term pictures and short-term pictures in its multi-picture buffer. The MLIP1 field is used to indicate the maximum long-term picture index allowed in the buffer. If no prior value of MLIP1 has been sent, no long-term pictures shall be in use, i.e. MLIP1 shall initially have an implied value of "0". Upon receiving an MLIP1 parameter, a new MLIP1 shall take effect until another value of MLIP1 is received. Upon receiving a new MLIP1 parameter in the bitstream, all long-term pictures with associated long-term indices greater than or equal to MLIP1 shall be considered marked "unused". The frequency of transmitting MLIP1 is out of the scope of this Recommendation. However, the encoder should send an MLIP1 parameter upon receiving an error message, such as an Intra request message.

A short-term picture can be changed to a long-term picture by using an MMCO command with an associated DPN and LPIN. The short-term picture number is derived from DPN and the long-term picture index is LPIN. Upon receiving such an MMCO command, the decoder shall change the short-term picture with PN indicated by DPN to a long-term picture and shall assign it to the long-term index indicated by LPIN. If a long-term picture with the same long-term index already exists in the buffer, the previously-

existing long-term picture shall be marked “unused”. An encoder shall not assign a long-term index greater than MLIP1–1 to any picture. If LPIN is greater than MLIP1–1, this condition should be treated by the decoder as an error. For error resilience, the encoder may send the same long-term index assignment operation or MLIP1 specification message repeatedly. If the picture specified in a long-term assignment operation is already associated with the required LPIN, no action shall be taken by the decoder. An encoder shall not assign the same picture to more than one long term index value. If the picture specified in a long-term index assignment operation is already associated with a different long-term index, this condition should be treated as an error. An encoder shall only change a short-term picture to a long-term picture within MAX_PN transmitted consecutive stored pictures. In other words, a short-term picture shall not stay in the short-term buffer after more than MAX_PN-1 subsequent stored pictures have been transmitted. An encoder shall not assign a long-term index to a short-term picture that has been marked as “unused” by the decoding process prior to the first such assignment message in the bitstream. An encoder shall not assign a long-term index to a picture number that has not been sent.

4.1.1.2 Decoder Process for Reference Picture Buffer Mapping

The decoder employs indices when referencing a picture for motion compensation on the macroblock layer. In pictures other than B pictures, these indices are the default relative indices of pictures in the multi-picture buffer when the fields ADPN and LPIR are not present in the current slice layer as applicable, and are re-mapped relative indices when these fields are present. In B pictures, the first one or two pictures (depending on BTPSM) in relative index order are used for backward prediction, and the forward picture reference parameters specify a relative index into the remaining pictures for use in forward prediction. (needs to be changed)

The indices of pictures in the multi-picture buffer can be re-mapped onto newly specified indices by transmitting the RMPNI, ADPN, and LPIR fields. RMPNI indicates whether ADPN or LPIR is present. If ADPN is present, RMPNI specifies the sign of the difference to be added to a picture number prediction value. The ADPN value corresponds to the absolute difference between the PN of the picture to be re-mapped and a prediction of that PN value. The first transmitted ADPN is computed as the absolute difference between the PN of the current picture and the PN of the picture to be re-mapped. The next transmitted ADPN field represents the difference between the PN of the previous picture that was re-mapped using ADPN and that of another picture to be re-mapped. The process continues until all necessary re-mapping is complete. The presence of re-mappings specified using LPIR does not affect the prediction value for subsequent re-mappings using ADPN. If RMPNI indicates the presence of an LPIR field, the re-mapped picture corresponds to a long-term picture with a long-term index of LPIR. If any pictures are not re-mapped to a specific order by RMPNI, these remaining pictures shall follow after any pictures having a re-mapped order in the indexing scheme, following the default order amongst these non-re-mapped pictures.

If the decoder detects a missing picture, it may invoke some concealment process, and may insert an error-concealed picture into the multi-picture buffer. Missing pictures can be identified if one or several picture numbers are missing or if a picture not stored in the multi-picture buffer is indicated in a transmitted ADPN or LPIR. Concealment may be conducted by copying the closest temporally preceding picture that is available in the multi-picture buffer into the position of the missing picture. The temporal order of the short-term pictures in the multi-picture buffer can be inferred from their default relative index order and PN fields. In addition or instead, the decoder may send a forced INTRA update signal to the encoder by external means (for example, Recommendation H.245), or the decoder may use external means or back-channel messages (for example, Recommendation H.245) to indicate the loss of pictures to the encoder. A concealed picture may be inserted into the multi-picture buffer when using the "Sliding Window" buffering type. If a missing picture is detected when decoding a Slice, the concealment may be applied to the picture as if the missing picture had been detected at the picture layer.

4.1.1.3 Decoder Process for Multi-Picture Motion Compensation

Multi-picture motion compensation is applied if the use of more than one reference picture is indicated. For multi-picture motion compensation, the decoder chooses a reference picture as indicated using the reference frame fields on the macroblock layer. Once, the reference picture is specified, the decoding process for motion compensation proceeds.

4.1.1.4 Decoder Process for Reference Picture Buffering

The buffering of the currently decoded picture can be specified using the reference picture buffering type (RPBT). The buffering may follow a first-in, first-out ("Sliding Window") mode. Alternatively, the buffering may follow a customized adaptive buffering ("Adaptive Memory Control") operation that is specified by the encoder in the forward channel.

The "Sliding Window" buffering type operates as follows. First, the decoder determines whether the picture can be stored into “unused” buffer capacity. If there is insufficient “unused” buffer capacity, the short-term picture with the largest default relative index (i.e. the oldest short-term picture in the buffer) shall be marked as “unused”. The current picture is stored in the buffer and assigned a default relative index of zero. The default relative index of all other short-term pictures is incremented by one. The default relative indices of all long-term pictures are incremented by one minus the number of short-term pictures removed.

In the "Adaptive Memory Control" buffering type, specified pictures may be removed from the multi-picture buffer explicitly. The currently decoded picture, which is initially considered a short-term picture, may be inserted into the buffer with default relative index 0, may be assigned to a long-term index, or may be marked as “unused” by the encoder. Other short-term pictures may also be assigned to long-term indices. The buffering process shall operate in a manner functionally equivalent to the following: First, the current picture is added to the multi-picture buffer with default relative index 0, and the default relative indices of all other pictures are incremented by one. Then, the MMCO commands are processed:

If MMCO indicates a reset of the buffer contents , all pictures in the buffer are marked as “unused” except the current picture (which will be the picture with default relative index 0).

If MMCO indicates a maximum long-term index using MLIP1, all long-term pictures having long-term indices greater than or equal to MLIP1 are marked as “unused” and the default relative index order of the remaining pictures are not affected.

If MMCO indicates that a picture is to be marked as “unused” in the multi-picture buffer and if that picture has not already been marked as “unused”, the specified picture is marked as “unused” in the multi-picture buffer and the default relative indices of all subsequent pictures in default order are decremented by one.

If MMCO indicates the assignment of a long-term index to a specified short-term picture and if the specified long-term index has not already been assigned to the specified short-term picture, the specified short-term picture is marked in the buffer as a long-term picture with the specified long-term index. If another picture is already present in the buffer with the same long-term index as the specified long-term index, the other picture is marked as “unused”. All short-term pictures that were subsequent to the specified short-term picture in default relative index order and all long-term pictures having a long-term index less than the specified long-term index have their associated default relative indices decremented by one. The specified picture is assigned to a default relative index of one plus the highest of the incremented default relative indices, or zero if there are no such incremented indices.

4.2 Motion Compensation

4.2.1 Fractional pixel accuracy

4.2.1.1 1/4 luminance sample interpolation

The pixels labeled as 'A' represent original pixels at integer positions and other symbols represent the pixels to be interpolated

```
A   d   b   d   A
e   h   f   h
b   g   c   g   b
e   h   f   i
A       b       A
```

The interpolation proceeds as follows

1 The 1/2 sample positions labeled as ‘b’ are obtained by first applying 6-tap filter (1,-5,20,20,-5,1) to nearest pixels at integer locations in horizontal or vertical direction. The resulting intermediate value *b* is divided by 32, rounded to the nearest integer and clipped to lie in the range [0, 255].

2 The 1/2 sample position labeled as ‘c’ is obtained using 6-tap filtering (1,-5,20,20,-5,1) of intermediate the values *b* of the closest 1/2 sample positions in vertical or horizontal direction. The obtained value is divided by 1024, rounded to the nearest integer and clipped to lie in the range [0, 255].

3 The 1/4 sample positions labeled as ‘d’, ‘g’, ‘e’ and ‘f’ are obtained by averaging (with truncation to the nearest integer) the two nearest samples at integer or 1/2 sample position using d=(A+b)/2, g=(b+c)/2, e=(A+b)/2, f=(b+c)/2.

4 The 1/4 sample positions labeled as 'h' are obtained by averaging (with truncation to nearest integer) the two nearest pixels ‘b’ in diagonal direction.

5 The pixel labeled 'i' (the "funny" position) is computed as $(A_1+A_2+A_3+A_4+2)/4$ using the four nearest original pixels.

4.2.1.2 1/8 Pel Luminance Samples Interpolation

The pixels labeled as 'A' represent original pixels at integer positions and other symbols represent 1/2, 1/4 and 1/8 pixels to be interpolated.

A	d	**b^1**	d	**b^2**	d	**b^3**	d	**A**
d	e	d	f	d	f	d	e	
b^1	d	**c^{11}**	d	**c^{12}**	d	**c^{13}**	d	
d	f	d	g	d	g	d	f	
b^2	d	**c^{21}**	d	**c^{22}**	d	**c^{23}**	d	
d	f	d	g	d	g	d	e	
b^3	d	**c^{31}**	d	**c^{32}**	d	**c^{33}**	d	
d	e	d	f	d	f	d	e	
A								**A**

The interpolation proceeds as follows

1 The 1/2 and 1/4 sample positions labeled as 'b' are obtained by first calculating intermediate values ‘*b*’ using 8 tap filtering of nearest pixels at integer locations in horizontal or vertical direction. The final value of 'b' is calculated as (*b*+128)/256 and clipped to lie in the range [0, 255]. The 1/2 and 1/4 sample positions labeled as 'c' are obtained by 8 tap filtering of intermediate values *b* in horizontal or vertical direction, dividing the result of filtering by 65536, rounding it to the nearest integer and clipping it to lie in the range [0, 255]. The filter tap values depend on pixel position and are listed below:

‘b’, ‘c’ position	filter tabs
1/4	(-3, 12, -37, 229, 71, -21, 6, -1)
2/4	(-3, 12, -39, 158, 158, -39, 12, -3)
3/4	(-1, 6, -21, 71, 229, -37, 12, -3)

2 The 1/8 sample positions labeled as ‘d’ are calculated as the average (with truncation to the nearest integer) of the two closest ‘A’, ‘b’ or ‘c’ values in the horizontal or vertical direction.

3 The 1/8 sample positions labeled as ‘e’ are calculated by averaging with truncation of two 1/4 pixels labeled as ‘b^1’, which are the closest in the diagonal direction *[Editor: what about the e-position column 8, row 6 ?]*. The 1/8 sample positions labeled as ‘g’ are calculated via $(A+3c^{22}+2)/4$ and the 1/8 sample positions marked as ‘f’ are calculated via $(3b^1+ b^1+2)/4$ (pixel 'b^2' closer to ‘f’ is multiplied by 3).

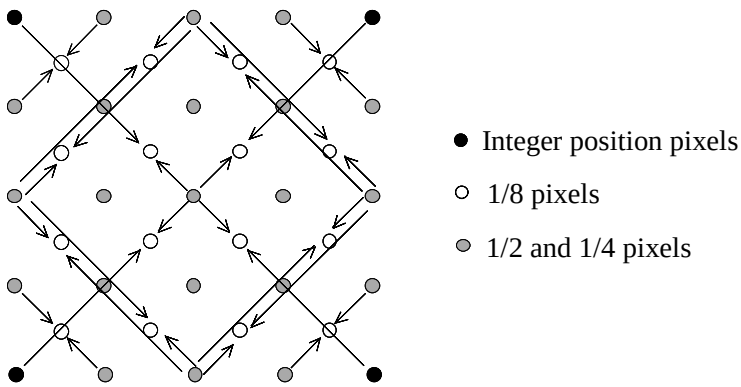


FIGURE 11

Diagonal interpolation for 1/8 pel interpolation.

4.3 Transform Coefficient Decoding

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.

4.3.1 Transform for Blocks of Size 4x4 Samples

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'.

4.3.2 Transform for Blocks of Size 2x2 Samples (DC vales for Chrominance)

With the low resolution of chrominance 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 chrominance. The 2 dimensional 2x2 transform procedure is illustrated below. DC0,1,2,3 are the DC coefficients of 2x2 chrominance 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

4.3.3 Zig-zag Scanning and Quantization

4.3.3.1 Simple Zig-Zag Scan

Except for Intra coding of luminance 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:

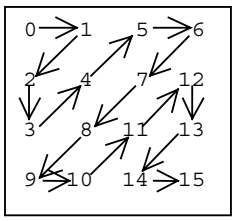


FIGURE 12

Simple zig-zag scan.

4.3.3.2 Double Zig-Zag 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.

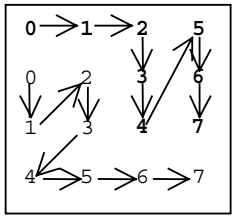


FIGURE 13

Double zig-zag scan.

4.3.3.3 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 luminance quantization/dequantization. This could be called QP_{luma}. For chrominance 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)
3881,4351,4890,5481,6154,6914,7761,8718,9781,10987,12339,13828,15523,17435,19561,21873,24552,27656,30847,34870,38807,43747,49103,54683,61694,68745,77615,89113,100253,109366,126635,141533

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.

4.3.3.4 Scanning and Quantization of 2x2 chrominance 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'.

Chrominance 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 TABLE 7.

4.3.4 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

4.4 Deblocking Filter

After the reconstruction of a macroblock a conditional filtering of this macroblock takes place, that effects the *boundaries* of the 4x4 block structure. For this purpose each 4x4 luminance block in a reconstructed macroblock is assigned a filtering **Strength**, which has the following value:

TABLE 4

Assignment of filtering strength.

4x4 block condition	Strength
Macro block is INTRA-coded, or member of SP-frame	2
Macro block is non INTRA, but has nonzero coefficients	1
Else	0

Filtering across 4x4 block boundary takes place if:

- Strength ≠ 0 on at least one of the sides of the boundary, or
- the absolute difference between one of the motion vector components of the two adjacent blocks is at least one integer pixel (four ¼ pixels). For macro blocks in B-frames which are predicted "bidirectional" or "direct", this condition has to be valid for at least one of the involved vectors, or
- the adjacent motion vectors refer to different reference frames

Filtering takes place on a macroblock level. In a first step the 16 pel of the 4 vertical edges (horizontal filtering) of the 4x4 raster are filtered. Than the 4 horizontal edges (vertical filtering) follow. Note, that this process also affects the boundaries of the already reconstructed macroblocks above and to the right of the current macroblock.

Block boundaries of chrominance blocks always correspond to a block boundary of luminance blocks. Therefore corresponding **Strength** and **motion vector difference** for luminance are also used for chrominance boundaries. That is, for every 4x4 luminance block two 2x2 chrominance blocks are processed.

4.4.1 Picture Content Dependent Parameters for Filtering

The set of eight pixels across a 4x4 block horizontal or vertical boundary is denoted as

$p_4,p_3,p_2,p_1 \mid q_1,q_2,q_3,q_4$

FIGURE 14

Illustration of de-blocking filtering.

with the actual boundary between p_1 and q_1. Up to two pixels can be updated as a result of the filtering process on both sides of the boundary (that is at most p_2, p_1, q_1, q_2). The number of updated pixels on both sides of the boundary depends on corresponding activity parameters a_p and a_q. They are limited by the so called overall activity parameter *n*:

$n = \min(3, 4 - |p_1-q_1| * \alpha(QP) / 128)$

Activity parameter a_p is equal to the smallest integer i>0 not larger than n for which the inequality $|p_1-p_i+1| \geq \beta(QP)$ holds. If no i < n satisfies the condition, a_p is set to n. a_q is obtained correspondingly. For the quantizer dependant threshold parameters α and β see TABLE 5 below.

TABLE 5

QP dependent threshold parameters α and β.

QP	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
α	128	128	128	128	128	128	128	128	128	128	122	96	75	59	47	37
β	0	0	0	0	0	0	0	0	3	3	3	4	4	4	6	6
QP	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

α	29	23	18	15	13	11	9	8	7	6	5	4	3	3	2	2
β	6	7	8	8	9	9	10	10	11	11	12	12	13	13	14	14

4.4.2 Filtering Process

There are 3 conditions, that determine which of the boundary pixels are filtered:

- If $(a_p > 1)$ and $(a_q > 1)$ the pixels p_1 and q_1 are filtered by:

$$\Delta = \text{Clip}(-c, c, ((q_1 - p_1) << 2 + (p_2 - q_2) + 4) >> 3)$$
$$P_1 = \text{Clip}(0, 255, (p_1+\Delta))$$
$$Q_1 = \text{Clip}(0, 255, (q_1-\Delta))$$

- If $(a_p == 3)$ and $(a_q > 1)$ p_2 is filtered by:

$$\Delta = \text{Clip}(-c_p, c_p, (p_3+p_1 - p_2<<1) >> 1)$$
$$P_2 = \text{Clip}(0, 255, (p_2+\Delta))$$

- If $(a_p > 1)$ and $(a_q == 3)$ q_2 is filtered by:

$$\Delta = \text{Clip}(-c_q, c_q, (q_3+q_1 - q_2<<1) >> 1)$$
$$Q_2 = \text{Clip}(0, 255, (q_2+\Delta))$$

Here Clip() denotes a clipping function with the parameters Clip(Min, Max, Value)
with the clipping constants:

$$c_p = \text{ClipTbl}(QP, \text{Strength}_p)$$
$$c_q = \text{ClipTbl}(QP, \text{Strength}_q)$$
$$c = (c_p + c_q + a_p + a_q) / 2$$

TABLE 6

ClipTbl (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	28	29	30	31
ClipTbb(qp,0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ClipTbl(qp,1)	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	2	2	2	2	3	3	3	3	4	5	5	
ClipTbl(qp,2)	0	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	

4.4.3 Stronger filtering for intra coded macroblocks

In case of intra coding and areas in the picture with little texture the subjective quality can be improved by stronger filtering across macroblock boundaries (that is: *not* 4x4 block boundaries). This stronger filtering is performed only if one or both of the adjacent macroblocks are intra coded and $a_p = 3$ and $a_q = 3$ and $1 < |p_1-q_1| < QP / 4$.

In this case luminance filtering is performed using the formulas shown below. For chrominance pixels p_3 to q_3, are not filtered.

$$P_1 = (25*p_3 + 26*p_2 + 26*p_1 + 26*q_1 + 25*q_2 + 64) / 128$$
$$P_2 = (25*p_4 + 26*p_3 + 26*p_2 + 26*P_1 + 25*q_1 + 64) / 128$$
$$P_3 = (26*p_4 + 51*p_3 + 26*P_2 + 25*P_1 + 64) / 128$$
$$Q_1 = (25*p_2 + 26*p_1 + 26*q_1 + 26*q_2 + 25*q_3 + 64) / 128$$
$$Q_2 = (25*p_1 + 26*Q_1 + 26*q_2 + 26*q_3 + 25*q_4 + 64) / 128$$
$$Q_3 = (25*Q_1 + 26*Q_2 + 51*q_3 + 26*q_4 + 64) / 128$$

Note, that when filtering across the next 4x4 block boundary, pixel q_3 may get overwritten. To compensate the effect Q_3 is stored and used as P_2 when filtering across the next 4x4 block boundary.

4.5 Entropy Coding

4.5.1 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.

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

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). TABLE 7 lists the connection between code number and most of the parameters used in the present coding method.

TABLE 7

Connection between codeword number and parameter values.

Code_ number	RUN	MB_Type		Intra_pred_mode[1]		MVD DQUANT	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	5	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 0.

4.5.2 Context-based Adaptive Binary Arithmetic Coding (CABAC)

4.5.2.1 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):

- *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.
- *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.
- *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.

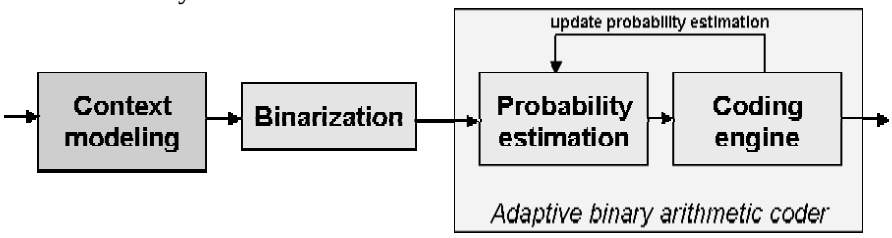


FIGURE 15

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 15. 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 4.5.2.2 and Section 4.5.2.3. 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 4.5.2.5. 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.

4.5.2.2 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).

4.5.2.2.1 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 4.5.2.5.

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 16) 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 + B*, which results in three 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 4.5.2.3.

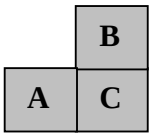


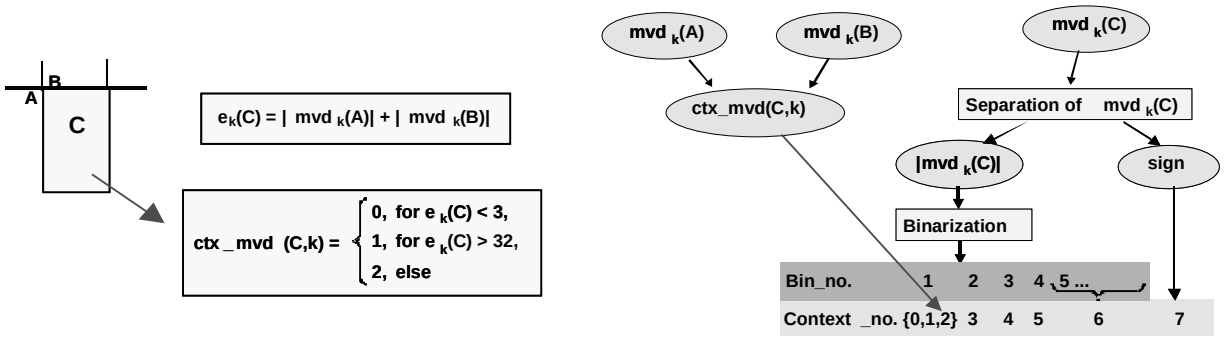
FIGURE 16

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

4.5.2.2.1.2 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 16). 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) + ((B==Skip)?0:1)* and by *ctx_mb_type_inter(C) = ((A==Direct)?0:1) + ((B==Direct)?0:1)* for B-frame coding. Thus, we obtain 3 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 4.5.2.5. 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. 4.5.2.3. Thus, a total number of 6 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 17

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.

4.5.2.2.2 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 17 (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:

$$ctx\_mvd(C,k) = \begin{cases} 0, e_k(C) < 3, \\ 1, e_k(C) > 32, \\ 2, else. \end{cases}$$

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 of the modulus, we have 4 additional models: one for the second, one for the third bin, one for the fourth, and one model for all remaining bins. In addition, the sign coding routine is provided with a separate model. This results in a total sum of 8 different models for each vector component (see FIGURE 17).

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 in FIGURE 2. Coding of Blk_size is done by using the binarization of the P_MB_type as described in Section 4.5.2.3.

4.5.2.2.3 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. **Error! Reference source not found.**, a context model is built by using symbols A and B of the reference frame parameter belonging to the two neighboring macroblocks (cp. FIGURE 16). 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.

4.5.2.3 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.

4.5.2.3.1 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 16. 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.

4.5.2.3.2 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 16). 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. 0. 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.

4.5.2.3.3 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 8, 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.

TABLE 8

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

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

4.5.2.3.3.1 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.

4.5.2.3.3.2 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.

4.5.2.4 Context Modeling for Coding of Dquant

For a given macroblock, the value of Dquant is first mapped to a positive value using the arithmetic wrap as described in section 3.4.7. This wrapped value *C* is then coded conditioned on the corresponding Dquant *A* of the left neighboring macroblock, such that that *ctx_dquant(C)=(A !=0);* This results in 2 context models for the first bin. Two additional models are used for the second and all remaining bins of the binarized value *C*. Thus, a total sum of four context models are used for Dquant.

TABLE 9

Binarization by means of the unary code tree

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**	**. .**

4.5.2.5 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 9.

TABLE 10

(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 10.

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**

B_MB_typ
0
1
2
3
4
5
6
7
.
17
Bin_no

4.5.2.6 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.

5 B-pictures

5.1 Introduction

Temporal scalability is achieved using bi-directionally predicted pictures, or B pictures. The use of B pictures is indicated in PTYPE. 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. FIGURE 18 illustrates the predictive structure with two B pictures inserted between I/P pictures.

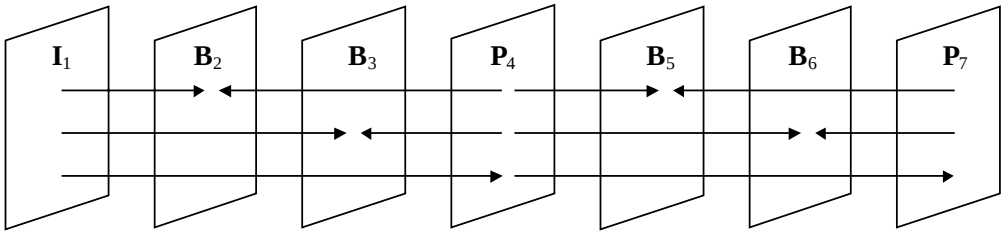


FIGURE 18

Illustration of B picture concept.

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 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.

5.2 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 as in the co-located macroblock of the temporal following reference picture is used. 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.

5.3 Syntax

Some additional syntax elements are needed for B pictures. The structure of B picture related fields is shown in FIGURE 19. 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 are defined to indicate the different prediction types for B pictures. The fields of Blk_size, MVDFW, and MVDBW are inserted to enable bi-directional prediction. These fields replace the syntax element MVD.

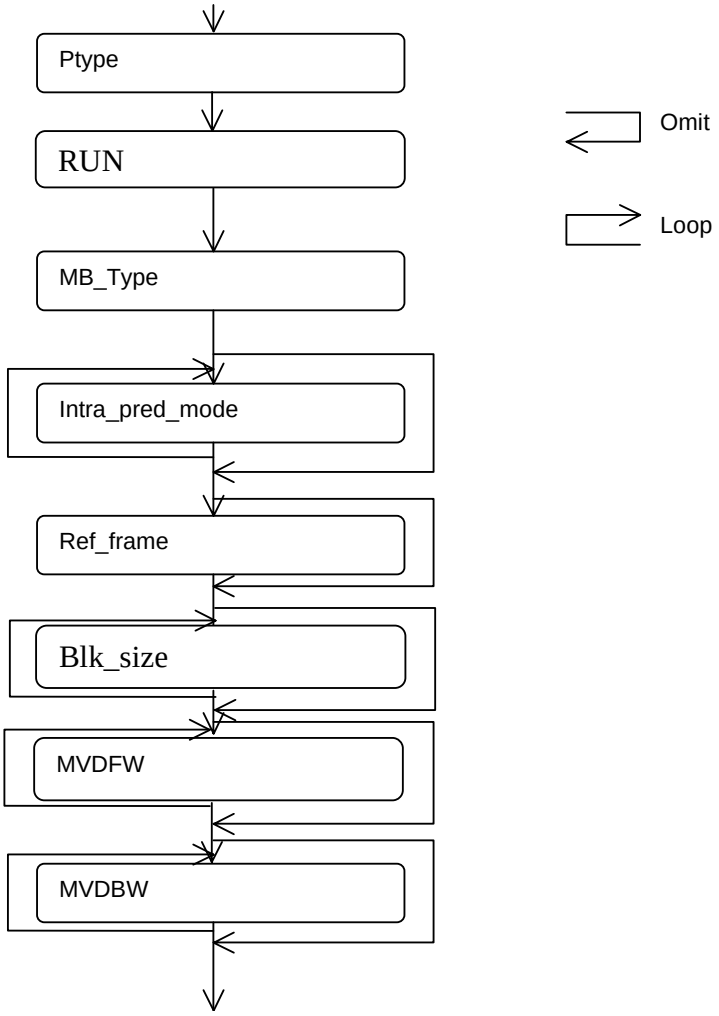


FIGURE 19

Syntax diagram for B pictures.

5.3.1 Picture type (Ptype) and RUN

See section 2.1 for definition.

5.3.2 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 in 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 11shows 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 predicted from a previous picture with block size NxM. The “backward_NxM” indicates that the macroblock is predicted 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 are 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 in section 3.4.3 and TABLE 7. No motion vector data is transmitted for the intra modes.

5.3.3 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 7.

5.3.4 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 12.

TABLE 11

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, i.e. when Ptype=4..

[2] Intra_16x16 indicates 16x16 based intra mode and should represent 24 different prediction modes as defined in section 3.4.2.1. For code_number greater than 16 in TABLE 11, please see the code numbers from 9 and upwards in the field of inter MB_type of TABLE 7 for reference.

TABLE 12

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
…	…

5.3.5 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 13. 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 13

Code_number for Blk_size

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

5.3.6 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 thatindicated in FIGURE 2. For the code_number of motion vector data, please refer to TABLE 7.

5.4 Decoder Process for motion vector

5.4.1 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 3.4.5. 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, the candidate predictor for that macroblock is set to zero for that motion vector type.

5.4.2 Motion vectors in direct mode

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

If 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.

$MV_F = (TR_B * MV) / TR_D$

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

Here, MV_F is the forward motion vector, MV_B is the backward motion vector, and MV represents the motion vector in the corresponding macroblock in the subsequent reference picture. Note that if the subsequent reference picture 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 in use, the actual reference frame (the same as for the co-located macroblock of the following picture) is used for the calculation of the temporal distances TR_D and TR_B.

6 SP-pictures

6.1 Introduction

SP-pictures make use of motion-compensated predictive coding to exploit temporal redundancy in the sequence similar to P-pictures while still allowing identical reconstruction of the frame even when different reference frames are being used. This picture type provides functionalities for bitstream switching, splicing, random access, VCR functionalities such as fast-forward and error resilience/recovery. SP-pictures make use of the existing coding blocks and the syntax elements for P-type frames while the difference is being forward transform coding and quantizing the predicted block during the reconstruction of blocks in SP-pictures.

6.2 Syntax changes

The only syntax change related to SP-frames is illustrated in FIGURE 20 as changes to syntax diagram to FIGURE 4. An additional quantization parameter is sent after the quantization parameter used for prediction error. This additional quantization parameter is used for (de)quantization of the predicted block. Otherwise, SP-frames have the identical syntax elements as P-frames. And all the syntax elements are interpreted identical to P-frames, i.e., 4x4 or 4x8 or 8x8 etc. blocks for motion compensation, reference frame number, etc.

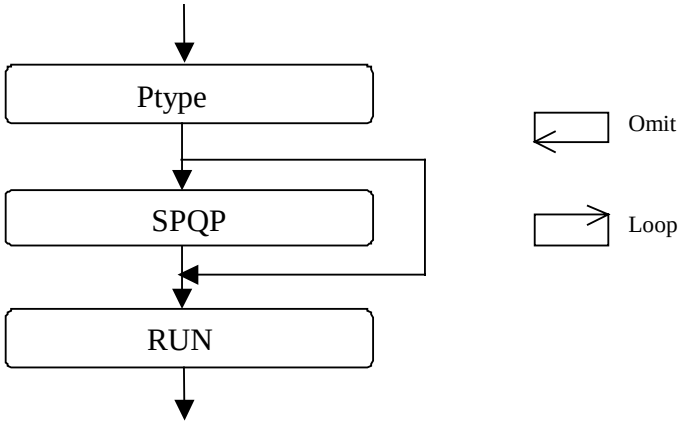


FIGURE 20

Syntax diagram for SP-frames

6.3 SP-frame decoding

SP-frames consist of intra and inter-type macroblocks. Intra macroblocks are encoded/decoded as in I and P-type pictures, therefore the following section refers to the decoding of the inter-type macroblocks in SP-frames.

Similar to P-type frame decoding, first the motion-compensated predicted block is formed by using received motion vector information, the reference frame number and the already decoded frames. Then, forward transform is applied to the predicted block. The resulting coefficients and the received level values are then used to calculate reconstructed image coefficients as follows:

$L_{rec} = (K_{pred} x A(QP_2) + L_{err} x F(QP_1,QP_2) + f x 2^{20}) / 2^{20}$

where

$F(QP_1,QP_2) = (A(QP_2) x 2^{20} + A(QP_1)/2) / A(QP_1)$

and A(QP) is defined in Section 4.3.3.3. Here QP_1 is signaled by PQP value and the QP_2 is signaled by the additional QP parameter SPQP. Notice that when QP1=QP2, then the calculation of L_{rec} reduces simply to the sum of the received level and the level found by quantizing the predicted block coefficient. The

coefficients, L_{rec}, are then dequantized using $QP=QP_2$ and inverse transform is performed for these dequantized levels, as defined in sections 4.3.3 and 4.3.1, respectively. Finally, the result of the inverse transformation is shifted by 20 bit (with rounding).

While applying deblocking filter for macroblocks in SP-frames, all macroblocks are treated as Intra macroblocks as described in Section 4.4.

Since an additional 2x2 transform is performed for the DC coefficients of chrominance blocks in a macroblock after 4x4 transform of each, Section 4.3.2, the decoding of the chrominance component is performed in the similar manner as described above with the following difference: For chrominance macroblocks in SP-frames, an additional 2x2 transform is applied after 4x4 transform of the predicted chrominance blocks. And then the steps described above are repeated for the DC coefficients as well as for the AC coefficients. Note also that for chrominance blocks, the values of QP_1 and QP_2 are both changed according to the relation between QP values of luminance and chrominance specified in Section 4.3.3.3.

7 Hypothetical Reference Decoder

7.1 Purpose

The hypothetical reference decoder (HRD) is a mathematical model for the decoder, its input buffer, and the channel. The HRD is characterized by the channel's peak rate R (in bits per second), the buffer size B (in bits), and the initial decoder buffer fullness F (in bits). These parameters represent levels of resources (transmission capacity, buffer capacity, and delay) used to decode a bit stream.

A closely related object is the leaky bucket (LB), which is a mathematical constraint on a bit stream. A leaky bucket is characterized by the bucket's leak rate R_1 (in bits per second), the bucket size B_1 (in bits), and the initial bucket fullness B_1–F_1 (in bits). A given bit stream may be constrained by any number of leaky buckets $(R_1,B_1,F_1),\ldots,(R_N,B_N,F_N)$, $N \geq 1$. The LB parameters for a bit stream, which are encoded in the bit stream header, precisely describe the minimum levels of the resources R, B, and F that are sufficient to guarantee that the bit stream can be decoded.

7.2 Operation of the HRD

The HRD input buffer has capacity B bits. Initially, the buffer begins empty. At time t_{start} it begins to receive bits, such that it receives $S(t)$ bits through time t. $S(t)$ can be regarded as the integral of the instantaneous bit rate through time t. The instant at which $S(t)$ reaches the initial decoder buffer fullness F is identified as the decoding time t_0 of the first picture in the bit stream. Decoding times t_1, t_2, t_3, …, for subsequent pictures (in bit stream order) are identified relative to t_0, per Section 7.3. At each decoding time t_i, the HRD instantaneously removes and decodes all d_i bits associated with picture i, thereby reducing the decoder buffer fullness from b_i bits to $b_i - d_i$ bits. Between time t_i and t_{i+1}, the decoder buffer fullness increases from $b_i - d_i$ bits to $b_i - d_i + [S(t_{i+1}) - S(t_i)]$ bits. That is, for $i \geq 0$,

$$b_0 = F$$

$$b_{i+1} = b_i - d_i + [S(t_{i+1}) - S(t_i)].$$

The channel connected to the HRD buffer has peak rate R. This means that unless the channel is idle (whereupon the instantaneous rate is zero), the channel delivers bits into the HRD buffer at instantaneous rate R bits per second.

7.3 Decoding Time of a Picture

The decoding time t_i of picture i is equal to its presentation time τ_i, if there are no B pictures in the sequence. If there are B pictures in the sequence, then $t_i = \tau_i - m_i$, where $m_i = 0$ if picture i is a B picture; otherwise m_i equals $\tau_i - \tau_i'$, where τ_i' is the presentation time of the I or P picture that immediately precedes picture i (in presentation order). If there is no preceding I or P picture (i.e., if $i = 0$), then $m_i = m_0 = t_1 - t_0$. The presentation time of a picture is determinable from its temporal reference and the frame rate.

7.4 Schedule of a Bit Stream

The sequence (t_0,d_0), (t_1,d_1), (t_2,d_2), … is called the schedule of a bit stream. The schedule of a bit stream is intrinsic to the bit stream, and completely characterizes the instantaneous coding rate of the bit stream over its lifetime. A bit stream may be pre-encoded, stored to a file, and later transmitted over channels with different peak bit rates to decoders with different buffer sizes. The schedule of the bit stream is invariant over such transmissions.

7.5 Containment in a Leaky Bucket

A leaky bucket with leak rate R_1, bucket size B_1, and initial bucket fullness B_1–F_1 is said to contain a bit stream with schedule (t_0,d_0), (t_1,d_1), (t_2,d_2), … if the bucket does not overflow under the following conditions. At time t_0, d_0 bits are inserted into the leaky bucket on top of the B_1–F_1 bits already in the bucket, and the bucket begins to drain at rate R_1 bits per second. If the bucket empties, it remains empty until the next insertion. At time t_i, $i \geq 1$, d_i bits are inserted into the bucket, and the bucket continues to drain at rate R_1 bits per second. In other words, for $i \geq 0$, the state of the bucket just prior to time t_i is

$b_0 = B_1 – F_1$

$b_{i+1} = \max\{0, b_i + d_i – R_1(t_{i+1}–t_i)\}$.

The leaky bucket does not overflow if $b_i + d_i \le B_1$ for all $i \ge 0$.

Equivalently, the leaky bucket contains the bit stream if the graph of the schedule of the bit stream lies between two parallel lines with slope R_1, separated vertically by B_1 bits, possibly sheared horizontally, such that the upper line begins at F_1 at time t_0, as illustrated in the figure below. Note from the figure that the same bit stream is containable in more than one leaky bucket. Indeed, a bit stream is containable in an infinite number of leaky buckets.

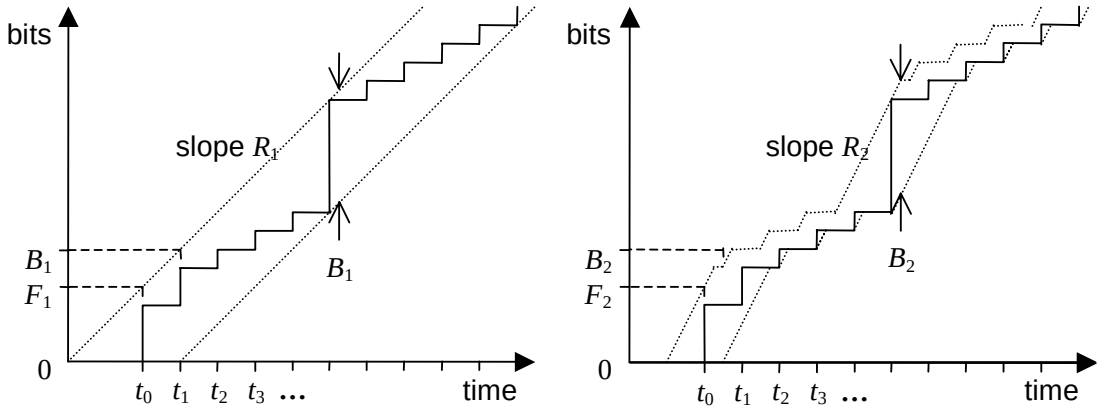


FIGURE 21

Illustration of the leaky bucket concept.

If a bit stream is contained in a leaky bucket with parameters (R_1,B_1,F_1), then when it is communicated over a channel with peak rate R_1 to a hypothetical reference decoder with parameters $R=R_1$, $B=B_1$, and $F=F_1$, then the HRD buffer does not overflow or underflow.

7.6 Bit Stream Syntax

The header of each bit stream shall specify the parameters of a set of $N \ge 1$ leaky buckets, $(R_1,B_1,F_1),\ldots,(R_N,B_N,F_N)$, each of which contains the bit stream. In the current Test Model, these parameters are specified in the first $1+3N$ 32-bit integers of the Interim File Format, in network (big-endian) byte order:

$N, R_1, B_1, F_1, \ldots, R_N, B_N, F_N$.

The R_n shall be in strictly increasing order, and both B_n and F_n shall be in strictly decreasing order.

These parameters shall not exceed the capability limits for particular profiles and levels, which are yet to be defined.

7.7 Minimum Buffer Size and Minimum Peak Rate

If a bit stream is contained in a set of leaky buckets with parameters (R_1,B_1,F_1), …, (R_N,B_N,F_N), then when it is communicated over a channel with peak rate R, it is decodable (i.e., the HRD buffer does not overflow or underflow) provided $B \ge B_{min}(R)$ and $F \ge F_{min}(R)$, where for $R_n \le R \le R_{n+1}$,

$B_{min}(R) = \alpha B_n + (1 – \alpha)B_{n+1}$

$F_{min}(R) = \alpha F_n + (1 – \alpha)F_{n+1}$

$\alpha = (R_{n+1} – R) / (R_{n+1} – R_n)$.

For $R \le R_1$,

$B_{min}(R) = B_1 + (R_1 – R)T$

$F_{min}(R) = F_1$,

where $T = t_{L-1} - t_0$ is the duration of the bit stream (i.e., the difference between the decoding times for the first and last pictures in the bit stream). And for $R \geq R_N$,

$B_{min}(R) = B_N$

$F_{min}(R) = F_N$.

Thus, the leaky bucket parameters can be linearly interpolated and extrapolated.

Alternatively, when the bit stream is communicated to a decoder with buffer size B, it is decodable provided $R \geq R_{min}(B)$ and $F \geq F_{min}(B)$, where for $B_n \geq B \geq B_{n+1}$,

$R_{min}(B) = \alpha R_n + (1 - \alpha) R_{n+1}$

$F_{min}(B) = \alpha F_n + (1 - \alpha) F_{n+1}$

$\alpha = (B - B_{n+1}) / (B_n - B_{n+1})$.

For $B \geq B_1$,

$R_{min}(B) = R_1 - (B - B_1)/T$

$F_{min}(B) = F_1$.

For $B \leq B_N$, the stream may not be decodable.

In summary, the bit stream is guaranteed to be decodable in the sense that the HRD buffer does not overflow or underflow, provided that the point (R,B) lies on or above the lower convex hull of the set of points $(0,B_1+R_1T)$, (R_1,B_1), …, (R_N,B_N), as illustrated in the figure below. The minimum start-up delay necessary to maintain this guarantee is $F_{min}(R) / R$.

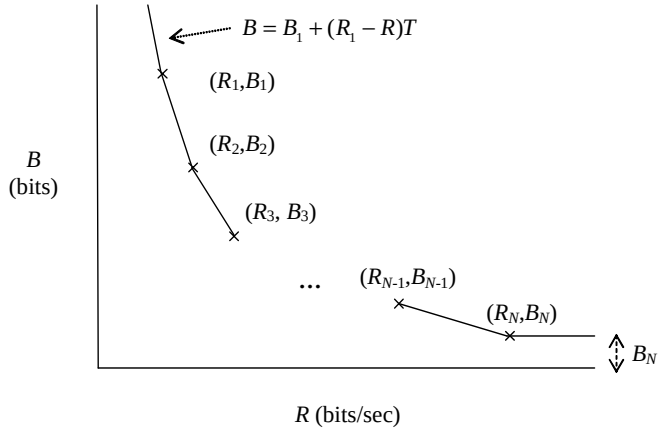


FIGURE 22

Illustration of the leaky bucket concept.

A compliant decoder with buffer size B and initial decoder buffer fullness F that is served by a channel with peak rate R shall perform the tests $B \geq B_{min}(R)$ and $F \geq F_{min}(R)$, as defined above, for any compliant bit stream with LB parameters $(R_1,B_1,F_1),\ldots,(R_N,B_N,F_N)$, and shall decode the bit stream provided that $B \geq B_{min}(R)$ and $F \geq F_{min}(R)$.

7.8 Encoder Considerations (informative)

The encoder can create a bit stream that is contained by some given N leaky buckets, or it can simply compute N sets of leaky bucket parameters after the bit stream is generated, or a combination of these. In the former, the encoder enforces the N leaky bucket constraints during rate control. Conventional rate control algorithms enforce only a single leaky bucket constraint. A rate control algorithm that simultaneously enforces N leaky bucket constraints can be obtained by running a conventional rate control algorithm for each of the N leaky bucket constraints, and using as the current quantization parameter (QP) the maximum of the QPs recommended by the N rate control algorithms.

Additional sets of leaky bucket parameters can always be computed after the fact (whether rate controlled or not), from the bit stream schedule for any given Rn, from the iteration specified in Section 7.5.

Appendix I Non-normative Encoder Recommendation

I.1 Motion Estimation and Mode Decision

I.1.1 Low-complexity mode

I.1.1.1 Finding optimum prediction mode

Both for intra prediction and motion compensated prediction, a similar loop as indicated in **Error! Reference source not found.** is run through. The different elements will be described below.

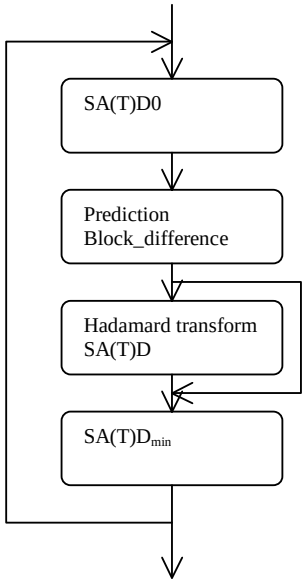


FIGURE 23

Loop for prediction mode decision

I.1.1.1.1 SA(T)D0

The SA(T)D to be minimised is given a 'bias' value SA(T)D0 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 4x4 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.

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 16x$QP_0(QP)$ to bias the decision toward selecting the direct mode.

The calculation of SA(T)D0 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 :
  $SA(T)D0 = -16 \times QP_0(QP)$
- Intra 4x4 mode :
  $SA(T)D0 = 24 \times QP_0(QP)$
- Intra 16x16 mode :
  $SA(T)D0 = 0$

I.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)

I.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.

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

I.1.1.1.4 Mode decision

Choose the prediction mode that results in the minimum value $SA(T)D_{min} = \min(SA(T)D+SA(T)D0)$.

I.1.1.2 Encoding on macroblock level

I.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 **Error! Reference source not found.** is performed. At the end of this loop the complete macroblock is intra coded and a $SATD_{intra}$ is calculated.

I.1.1.2.2 Table for intra prediction modes to be used at the encoder side

Error! Reference source not found. 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 **3**. 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). **Error! Reference source not found.** is therefore the relevant table for the encoder. Example: Prediction mode for A and B is 2. The string in **Error! Reference source not found.** 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 **3** '–' indicates that this instance can not occur because A or B or both are outside the picture.

TABLE 14

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

I.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. For B-frames, the motion search is also conducted for the temporal following reference picture to obtain backward motion vectors.

I.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 upwards 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 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.

I.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 4.2.1 - is automatically checked). If motion compensation with 1/8 pixel accuracy is used, an additional sub-pixel refinement step is performed in the described way. After fractional pixel search has been performed for the complete macroblock, the SATD for the whole macroblock is computed: $SATD_{inter}$.

I.1.1.2.6 Decision between intra and inter

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

I.1.2 High-complexity mode

I.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.

I.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.

I.1.2.1.2 Fractional pixel search

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

I.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 **Error! Reference source not found.**). The Lagrangian multiplier λ_{MOTION} is given by

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

for 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.

I.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.

I.1.2.2 Mode decision

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

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 given as

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

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.

I.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.

I.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.

I.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 that 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: $MODE \in \{INTRA4x4, INTRA16x16, SKIP, 16x16, 16x8, 8x16, 8x8, 8x4, 4x8, 4x4\}$,

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

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.

I.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.

I.3 Elimination of single coefficients in inter macroblocks

I.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.

I.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.

I.4 Encoding with Anticipation of Slice Losses

Low delay video transmission may lead to losses of slices. The decoder may then stop decoding until the next I picture or P picture may conduct a concealment, for example as explained in Appendix IV, and continue decoding. In the latter case, spatio-temporal error propagation occurs if the concealed picture content is referenced for motion compensation. There are various means to stop spatio-temporal error propagation including the usage of multiple reference pictures and Intra coding of macroblocks. For the latter case, a Lagrangian mode selection algorithm is suggested as follows.

Since transmission errors occur randomly, the decoding result is also a random process. Therefore, the average decoder distortion is estimated to control the encoder for a specified probability of packet losses *p*. The average decoding result is obtained by running *N* complete decoders at the encoder in parallel. The statistical process of loosing a slice is assumed to be independent for each of the *N* decoders. The slice loss process for each decoder is also assumed to be i.i.d. and a certain slice loss probability *p* is assumed to be known at the encoder. Obviously for large *N* the decoder gets a very good estimate of the average decoder distortion. However, with increasing *N* a linear increase of storage and decoder complexity in the encoder is incurred. Therefore, this method might not be practical in real-time encoding processes and complexity and memory efficient algorithms are currently under investigation.

To encode a macroblock in a P picture, the set of possible macroblock types is given as

$\mathcal{S}_{MB}$ = { SKIP, INTER_16x16, INTER_16x8, INTER_8x16,
INTER_8x8, INTER_8x4, INTER_4x8, INTER_4x4,
INTRA_4x4, INTRA_16x16}

For each macroblock the coding mode *m'* is selected according to

$$m' = \min_{m \in \mathcal{S}_{MB}} \{ D_m + \lambda R_m \}$$

with D_m being the distortion in the current macroblock when selecting macroblock mode m and R_m being the corresponding rate, i.e. the number of bits. For the COPY_MB and all INTER_MxN types, the distortion D_m is computed as

$$D_m = \frac{1}{N}\sum_{n=1}^{N}\sum_{i}(f_i - \hat{f}_{i,n,m}(p))^2$$

with f_i being the original pixel value at position i within the macroblock and $\hat{f}_{i,n,m}$ being the reconstructed pixel value at position i for coding macroblock mode m in the simulated decoder n. The distortion for the INTRA macroblocks remains unchanged. Since the various reconstructed decoders also contain transmission errors, the Lagrangian cost function for the COPY_MB and all INTER_MxN types increases making INTRA_NxN types more popular.

The λ parameter for mode decision depends on the quantization parameter q as follows

$$\lambda = (1 - p)\frac{q + 5}{34 - q}e^{0.1q} .$$

Appendix II 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 **Error! Reference source not found.** 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.

II.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 characteristic: 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%

II.2 Combining of Partitions according to Priorities

In order to allow unequal 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 Luminance AC Coefficients
- TYPE_COEFF_C Chrominance 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.

II.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 payload)

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

II.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 II.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.

II.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 straightforward and reverse to the packetization process. (Note that the QP and Format fields currently 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).

II.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.

Appendix III Interim File Format

[Editor: this text needs work by the proponent until the next meeting Jan 2002. If no improvements are made, the text will be dropped.]

III.1 General

A file is self-contained.

A file consists of clumps, which are object-like entities and similar to boxes of ISO/IEC 14496-1:2001 (ISO media file format). Name 'clump' was chosen to differentiate them from MPEG-4's boxes, chunks, and objects as well as from QuickTime's atoms. Note: In fact, the definition of a clump is the same as the definition of a box except for the extended type mechanism of a box. Thus, in order to emphasize the common definition of a clump and a box, it might be appropriate to refer to a box instead of a clump in this document. However, as this was not agreed in the Pattaya meeting, this change was not done yet.

A clump may contain other clumps. A clump may have member attributes. If a clump contains attributes and other clumps, clumps shall follow the attribute values.

The attribute values in the clumps are stored with the most significant byte first, commonly known as network byte order or big-endian format.

A number of clumps contain index values into sequences in other clumps. These indexes start with the value 0 (0 is the first entry in the sequence).

The Syntactic Description Language (SDL) of ISO/IEC 14496-1:2001 is used to define the file format. In addition to the existing basic data types, the UVLC elementary data type is defined in this document. It shall be used to carry variable-length bit-fields that follow the JVT UVLC design.

Unrecognized clumps should be skipped and ignored.

III.2 File Identification

The File Type Clump is the first clump of the file. JVT files shall be identified from a major Brand field equal to 'jvt '.

The preferred file extension is '.jvt'.

III.3 Clump

III.3.1 Definition

Clumps start with a header, which gives both size and type. The header permits compact or extended size (32 or 64 bits). Most clumps will use the compact (32-bit) size. The size is the entire size of the clump, including the size and type header, fields, and all contained clumps. This facilitates general parsing of the file.

III.3.1.1 Syntax

```
aligned(8) class clump (unsigned int(32) clumpType)
     unsigned int(32) size;
     unsigned int(32) type = clumpType;

     if (size==1) {
          unsigned int(64) largesize;
     } else if (size==0) {
          // clump extends to end of file
     }
}
```

III.3.1.2 Semantics

- size is an integer that specifies the number of bytes in this clump, including all its fields and contained clumps; if size is 1 then the actual size is in the field largesize; if size is 0, then this clump is the last one in the file, and its contents extend to the end of the file (normally only used for an Alternate Track Media Clump)
- type identifies the clump type; standard clumps use a compact type, which is normally four printable characters, to permit ease of identification, and is shown so in the clumps below.

III.4 Clump Order

An overall view of the normal encapsulation structure is provided in the following table.

The table shows those clumps that may occur at the top-level in the left-most column; indentation is used to show possible containment. Thus, for example, an Alternate Track Header Clump (atrh) is found in a Segment Clump (segm).

Not all clumps need be used in all files; the mandatory clumps are marked with an asterisk (*). See the description of the individual clumps for a discussion of what must be assumed if the optional clumps are not present.

There are restrictions in which order the clumps shall appear in a file. See the clump definitions for these restrictions.

TABLE 15

Clump types.

ftyp		*	III.5.1	File Type Clump, identifies the file format
jvth		*	III.5.2	File Header Clump, file-level meta-data
cinf			III.5.3	Content Info Clump, describes file contents
atin		*	III.5.4	Alternate Track Info Clump, describes characteristics of tracks
prms		*	III.5.5	Parameter Set Clump, enumerated set of frequently changing coding parameters
segm		*	III.5.6	Segment Clump, contains meta- and media data for a defined period of time
	atrh	*	III.5.7	Alternate Track Header Clump, meta-data for a track
	swpc		III.5.9	Switch Picture Clump, identifies pictures that can be used to switch between tracks.
	atrm	*	III.5.8	Alternate Track Media Clump, media data for a track

III.5 Clump Definitions

III.5.1 File Type Clump

III.5.1.1 Definition

Clump Type: `ftyp'

Container: File

Mandatory: Yes

Quantity: Exactly one

A media-file structured according to the ISO media file format specification may be compatible with more than one detailed specification, and it is therefore not always possible to speak of a single 'type' or 'brand' for the file. This clump identifies a JVT file in a similar fashion without claiming compatibility with the ISO format. However, it enables other file readers to identify the JVT file type. It must be placed first in the file.

III.5.1.2 Syntax

```
aligned(8) class FileTypeClump aligned(8) extends clump('ftyp') {
     unsigned int(32) majorBrand = 'jvt ';
     unsigned int(16) jmMajorVersion;
     unsigned int(16) jmMinorVersion;
     unsigned int(32) compatibleBrands[];   // to end of the clump
}
```

III.5.1.3 Semantics

This clump identifies the specification to which this file complies.

majorBrand is a brand identifier for the interim JVT file format. Only 'jvt ' shall be used for majorBrand, as the file format is not compatible with any other format.

jmMajorVersion and jmMinorVersion define the version of the standard working draft the file complies with. For example, JM-1 files shall have jmMajorVersion equal to 1 and jmMinorVersion equal to 0.

compatibleBrands is a list, to the end of the clump, of brands. Should only include the entry 'jvt '.

Note: As the interim JVT file format is based on the ISO media file format, it might be appropriate to allow a combination of many ISO media file format based file types into the same file. In such a case, the majorBrand might not be equal to 'jvt ' but 'jvt ' should be one of the compatibleBrands. As this option was not discussed in the Pattaya meeting, it is not reflected in the current specification of the interim JVT file format (this document).

III.5.2 File Header Clump

III.5.2.1 Definition

Clump Type: `jvth'

Container: File

Mandatory: Yes

Quantity: One or more

This clump must be placed as the second clump of the file.

The clump can be repeated at any position of the file when no container clump is open. A File Header Clump identifies a random access point to the file. In other words, no data prior to a selected File Header Clump is required to parse any of the succeeding data. Furthermore, any segment can be parsed without a forward reference to any of the data succeeding the particular segment.

III.5.2.2 Syntax

```
aligned(8) class fileHeaderClump extends clump('jvth') {
     unsigned int(8) majorVersion = 0x00;
     unsigned int(8) minorVersion = 0x00;
     unsigned int(32) timescale;
     unsigned int(32) numUnitsInTick;
     unsigned int(64) duration;
     unsigned int(16) pixAspectRatioX;
     unsigned int(16) pixAspectRatioY;
     unsigned int(16) maxPicId;
     unsigned int(8) numAlternateTracks;
     unsigned int(2) numBytesInPayloadCountMinusOne;
     unsigned int(2) numBytesInPictureOffsetMinusTwo;
     unsigned int(2) numBytesInPictureDisplayTimeMinusOne;
     unsigned int(2) numBytesInPictureCountMinusOne;
     unsigned int(2) numBytesInPayloadSizeMinusOne;
}
```

III.5.2.3 Semantics

majorVersion and minorVersion indicate the version of the file format. This specification defines the format for version 0.0 (majorVersion.minorVersion). Version numbering is independent of working draft document and joint model software as well as the version of the standard | recommendation. This allows parsers interpret the high-level syntax of the files, even if decoding of a file according to the indicated joint model or standard version was not supported.

timescale is the number of time units which pass in one second. For example, a time coordinate system that measures time in sixtieths of a second has a time scale of 60.

numUnitsInTick is the number of time units according to timescale that correspond to one clock tick. A clock tick is the minimum unit of time that can be presented in the file. For example, if the clock frequency of a video signal is (30 000) / 1001 Hz, timescale should be 30 000 and numUnitsInTick should be 1001.

duration is an integer that declares length of the file (in the indicated timescale). Value zero indicates that no duration information is available.

pixAspectRatioX and pixAspectRatioY define the pixel geometry, calculated by pixAspectRatioX / pixAspectRatioY. Value zero in either or both of the attributes indicate an unspecified pixel aspect ratio.

maxPicId gives the maximum value for the picture identifier.

numAlternateTracks gives the number of alternative encodings of the same source. Typically each encoding is targeted for different bit-rate. Each file shall contain at least one track.

numBytesInPayloadCountMinusOne indicates the number of bytes that are needed to signal the maximum number of payloads in any picture. For example, numBytesInPayloadCountMinusOne equal to zero indicates that one byte is needed to signal the number of payloads, and the maximum number of payloads is 255.

numBytesInPictureOffsetMinusTwo indicates the number of bytes that are needed to signal picture offsets. For example, numBytesInPictureOffsetMinusTwo equal to zero indicates that the offsets are two-byte integer values with a range of –32768 to 32767.

numBytesInPictureDisplayTimeMinusOne indicates the number of bytes that are needed to signal picture display time offsets.

numBytesInPictureCountMinusOne indicates the number of bytes that are needed to signal the maximum number of pictures in a segment.

numBytesInPayloadSizeMinusOne indicates the number of bytes to signal the maximum payload size in bytes.

III.5.3 Content Info Clump

III.5.3.1 Definition

Clump Type: `cinf´

Container: File

Mandatory: No

Quantity: Zero or more

This clump gives information about the content of the file.

The clump can be repeated at any position of the file when no container clump is open.

III.5.3.2 Syntax

```
aligned(8) class contentInfoClump extends clump('cinf') {
    unsigned int(64) creationTime;
    unsigned int(64) modificationTime;

    unsigned int(8) titleNumBytes;
    if (titleNumBytes)
```

```
        unsigned int(8)[titleNumBytes] title;

    unsigned int(8) authorNumBytes;
    if (authorNumBytes)
        unsigned int(8)[authorNumBytes] author;

    unsigned int(8) copyrightNumBytes;
    if (copyrightNumBytes)
        unsigned int(8)[copyrightNumBytes] copyright;

    unsigned int(16) descriptionNumBytes;
    if (descriptionNumBytes)
        unsigned int(8)[descriptionNumBytes] description;

    unsigned int(16) URINumBytes;
    if (URINumBytes)
        unsigned int(8)[URINumBytes] URI;
}
```

III.5.3.3 Semantics

creationTime is an integer that declares the creation time of the presentation (in seconds since midnight, Jan. 1, 1904).

modificationTime is an integer that declares the most recent time the presentation was modified (in seconds since midnight, Jan. 1, 1904.

titleNumBytes gives the number of bytes in title.

title, if present, contains the title of the file coded according to ISO/IEC 10646-1 UTF-8.

authorNumBytes gives the number of bytes in author.

author, if present, contains the author of the source or the encoded representation in the file coded according to ISO/IEC 10646-1 UTF-8.

copyrightNumBytes gives the number of bytes in copyright.

copyright shall be used only to convey intellectual property information regarding the source or the encoded representation in the file. copyright is coded according to ISO/IEC 10646-1 UTF-8.

descriptionNumBytes gives the number of bytes in description.

description shall be used only to convey descriptive information associated with the information contents of the file. description is coded according to ISO/IEC 10646-1 UTF-8.

URINumBytes gives the number of bytes in URI.

URI contains a uniform resource identifier (URI), as defined in IETF RFC 2396. URI is coded according to ISO/IEC 10646-1 UTF-8. URI shall be used to convey any related information to the file.

III.5.4 Alternate Track Info Clump

III.5.4.1 Definition

Clump Type: ‘atin’

Container: File

Mandatory: Yes

Quantity: One or more.

This clump specifies the characteristics of alternate tracks. The clump shall precede the first Segment Clump. The clump can be repeated at any position of the file when no container clump is open.

III.5.4.2 Syntax

```
aligned(8) class alternateTrackInfo {
     unsigned int(16) displayWindowWidth;
     unsigned int(16) displayWindowHeight;
     unsigned int(16) maxSDUSize;
     unsigned int(16) avgSDUSize;
     unsigned int(32) avgBitRate;
}

aligned(8) class alternateTrackInfoClump
     extends clump('atin') {
     (class alternateTrackInfo) trackInfo[numAlternateTracks];
}
```

III.5.4.3 Semantics

displayWindowWidth and displayWindowHeight declare the preferred size of the rectangular area on which video images are displayed. The values are interpreted as amount of pixels.

An SDU is defined as the payload and the payload header. maxSDUSize gives the size in bytes of the largest SDU of the track. avgSDUSize gives the average size of the SDU over the entire track. Value zero in either attribute indicates that no information is available.

avgBitRate gives the average bit-rate in bits/second over the entire track. Payloads and payload headers taken into account in the calculation.

III.5.5 Parameter Set Clump

III.5.5.1 Definition

Clump Type: 'prms'

Container: File

Mandatory: Yes

Quantity: One or more

This clump specifies a parameter set.

Parameter sets can be repeated in the file to allow random access. A parameter set is uniquely identified within a file based on parameterSetID. Decoders can infer a repetition of a parameter set if a set with the same parameterSetID has already appeared in a file. A redundant copy of a parameter set can safely be ignored.

III.5.5.2 Syntax

```
aligned(8) class parameterSetClump
     extends clump('prms') {
     unsigned int(16) parameterSetID;
     unsigned int(8) profile;
     unsigned int(8) level;
     unsigned int(8) version;
     unsigned int(16) pictureWidthInMBs;
     unsigned int(16) pictureHeightInMBs;
     unsigned int(16) displayRectangleOffsetTop;
     unsigned int(16) displayRectangleOffsetLeft;
     unsigned int(16) displayRectangleOffsetBottom;
     unsigned int(16) displayRectangleOffsetRight;
     unsigned int(8) displayMode;
```

```
    unsigned int(16) displayRectangleOffsetFromWindowTop;
    unsigned int(16) displayRectangleOffsetFromWindowLeftBorder;
    unsigned int(8) entropyCoding;
    unsigned int(8) motionResolution;
    unsigned int(8) partitioningType;
    unsigned int(8) intraPredictionType;
};
```

III.5.5.3 Semantics

parameterSetId gives the identifier of the parameter set. The identifier shall be unique within a file.

profile defines the coding profile in use.

level defines the level in use within the profile.

version defines the version in use within the profile and the level.

pictureWidthInMBs and pictureHeightInMBs define the extents of the coded picture in macroblocks.

displayRectangleOffsetTop, displayRectangleOffsetLeft, displayRectangleOffsetBottom, and displayRectangleOffsetRight define the rectangle to be displayed from the coded picture. Pixel units are used.

displayMode defines the preferred displaying mode. Value zero indicates that the display rectangle shall be rescaled to fit onto the display window. No scaling algorithm is defined. Image shall be as large as possible, no clipping shall be applied, image aspect ratio shall be maintained, and image shall be centered in the display window. Value one indicates that the display rectangle shall be located as indicated in displayRectangleOffsetFromWindowTop and displayRectangleFromWindowLeftBorder. No scaling shall be done and clipping shall be applied to areas outside the display window. No fill pattern is defined for areas in the display window that are not covered by the display rectangle.

displayRectangleOffsetFromWindowTop and displayWindowOffsetFromWindowLeftBorder indicate the location of the top-left corner of the display rectangle within the display window. The values are given in pixels. The values are valid only if displayMode is one.

Error! Reference source not found. clarifies the relation of different display rectangle and window related attributes. The dashed rectangle of in the decoded picture represents the display rectangle, which is indicated by displayRectangleOffsetTop, displayRectangleOffsetLeft, displayRectangleOffsetBottom, and displayRectangleOffsetRight.

entropyCoding equal to zero stands for UVLC, whereas value one stands for CABAC.

motionResolution equal to zero stands for full-pixel motion resolution, one stands for half-pixel motion resolution, two stands for ¼-pixel motion resolution, and three stands for 1/8-pixel motion resolution.

partitioningType equal to zero stands for the single slice mode and one stands for the data partitioning mode.

intraPredictionType equal to zero stands for normal INTRA prediction, whereas one stands for the constrained INTRA prediction.

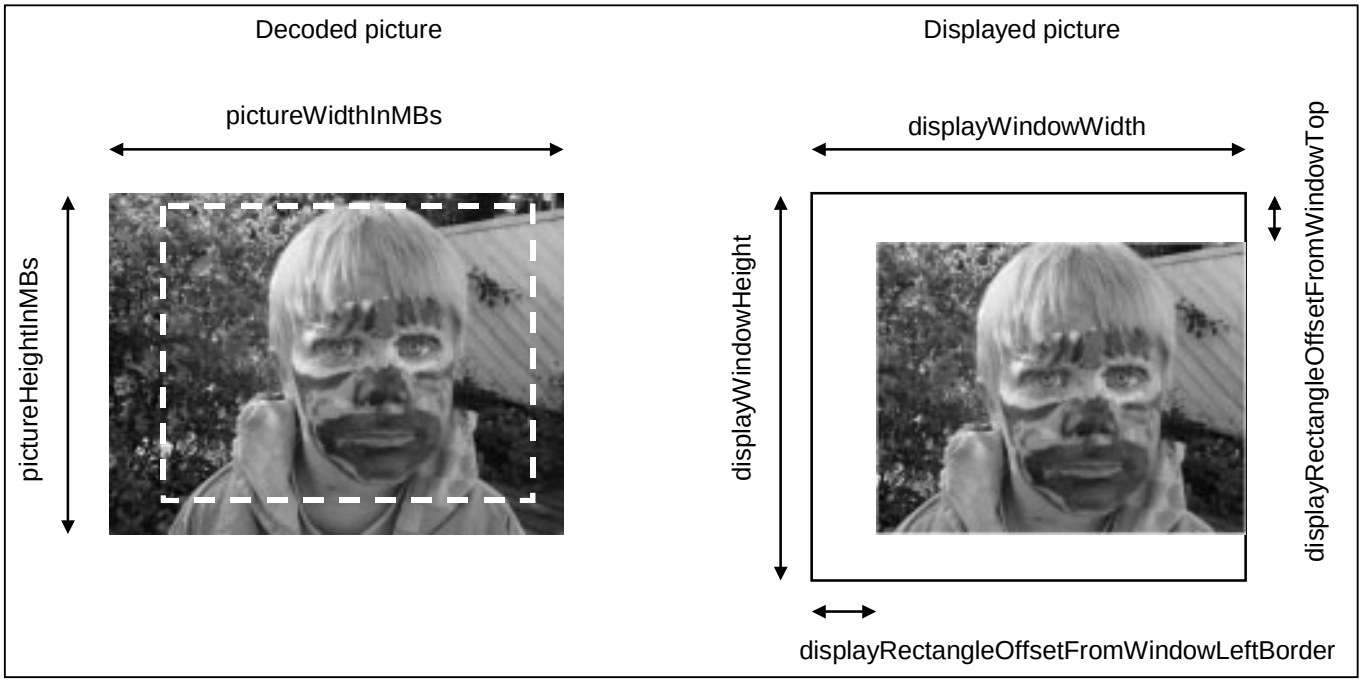


FIGURE 24

Relation of display window and rectangle attributes.

III.5.6 Segment Clump

III.5.6.1 Definition

Clump Type: `segm'
Container: File
Mandatory: Yes
Quantity: One or more

A segment clump contains data from a certain period of time. Segments shall not overlap in time. Segments shall appear in ascending order of time in the file. A segment clump is a container clump for several other clumps.

III.5.6.2 Syntax

```
aligned(8) class SegmentClump extends Clump('segm') {
    unsigned int(64) fileSize;
    unsigned int(64) startTick;
    unsigned int(64) segmentDuration;
}
```

III.5.6.3 Semantics

fileSize indicates the number of bytes from the beginning of the Segment Clump to the end of the file. Value zero indicates that no size information is available. When downloading a file to a device with limited storage capabilities, fileSize can be used to determine if a file fits into the available storage space. In a progressive downloading service, fileSize, startTick, and duration (in the File Header Clump) can be

used to estimate the average bit-rate of the file including meta-data. This estimation can then be used to decide how much initial buffering is needed before starting the playback.

startTick indicates the absolute time of the beginning of the segment since the beginning of the presentation (time zero). Any time offsets within the segment are relative to startTick.

segmentDuration indicates the duration of the segment. Value zero indicates that no duration information is available.

III.5.7 Alternate Track Header Clump

III.5.7.1 Definition

Clump Type: 'atrh'

Container: Segment Clump ('segm')

Mandatory: Yes

Quantity: One or more

An alternate track represents an independent encoding of the same source as for the other alternate tracks. The Alternate Track Header Clump contains meta-data for an alternate track. The clumps shall appear in the same order in all Segment Clumps and they can be indexed starting from zero. Each succeeding clump is associated with an index one greater than the previous one. The index can be used to associate the clump with a particular track and with the information given in the Alternate Track Info Clump.

The clump contains an indication of the number of the pictures in the alternate track in this segment. In addition, the clump contains picture information for each of these pictures. Picture information shall appear in ascending order of picture identifiers (in modulo arithmetic). In other words, picture information shall appear in coding/decoding order of pictures.

A picture information block contains a pointer to the coded representation of the picture. A picture is associated with a display time and with a number of so-called payloads.

A payload refers to a slice, a data partition, or a piece of supplemental enhancement information. A payload header refers to an equivalent definition as in VCEG-N72R1. For example, a payload header of a single slice includes the "first byte", an indication of the parameter set in use, and the slice header.

III.5.7.2 Syntax

```
aligned(8) class payloadInfo {
    unsigned int((numBytesInPayloadSizeMinusOne + 1) * 8) payloadSize;
    unsigned int(8) headerSize;
    unsigned int(4) payloadType;
    unsigned int(1) errorIndication;
    unsigned int(3) reserved = 0;
    if (payloadType == 0) { // single slice
        UVLC parameterSet;
        sliceHeader;

    else if (payloadType == 1) { // partition A
        UVLC parameterSet;
        sliceHeader;
        UVLC sliceID;
    }
    else if (payloadType == 2 || partitionType == 3) { // Partition B
or C
        UVLC pictureID;
        UVLC sliceID;
    }
```

```
    else if (payloadType == 5) { // Supplemental enhancement
information
        // no additional codewords
    }
}

aligned(8) class pictureInfo {
    bit intraPictureFlag;
    aligned(8) int((numBytesInPictureOffsetMinusTwo + 2) * 8)
pictureOffset;
    int((numBytesInPictureDisplayTimeMinusOne + 1) * 8)
pictureDisplayTime;
    unsigned int((numBytesInPayloadCountMinusOne + 1) * 8)
numPayloads;
    (class payloadInfo) payloadData[numPayloads];
}

aligned(8) class AlternateTrackHeaderClump extends Clump('atrh') {
    unsigned int((numBytesInPictureCountMinusOne + 1) * 8)
numPictures;
    (class pictureInfo) pictureData[numPictures];
}
```

III.5.7.3 Semantics

payloadInfo gives information related to a payload. payloadSize indicates the number of bytes in the payload (excluding the payload header). The value of headerSize is the number of bytes in the payload header, i.e., the number of bytes remaining in the structure. The rest of the data is defined in VCEG-N72R1.

pictureInfo gives information related to a picture.

intraPictureFlag is set to one, when the picture is an INTRA picture. The flag is zero otherwise.

A picture pointer is maintained to point to the beginning of the latest picture in the corresponding Alternate Track Media Clump. The pointer is relative to the beginning of the Alternate Track Media Clump. pictureOffset gives the increment or the decrement (in bytes) for the picture pointer to obtain the coded data for the picture. Initially, before updating the pointer for the first picture of the alternate track in a segment, the picture pointer shall be zero.

pictureDisplayTime gives the time when the picture is to be displayed. It is assumed that the picture remains visible until the next picture is to be displayed. The value is relative to the corresponding value of the previous picture.

numPayloads indicates the number of payloads in the picture. payloadData is an array of payloadInfo structures signaling the characteristics of the payloads.

numPictures indicates the number of pictures in the track during the period of the segment. pictureData is an array of pictureInfo structures signaling the meta-data of the pictures.

III.5.8 Alternate Track Media Clump

III.5.8.1 Definition

Clump Type:	'atrm'
Container:	Segment Clump ('segm')
Mandatory:	Yes
Quantity:	One or more

An alternate track represents an independent encoding of the same source as for the other alternate tracks. The Alternate Track Media Clump contains the media-data for an alternate track and for the duration of the segment. The clumps shall appear in the same order in all Segment Clumps and they can be indexed starting from zero. Each succeeding clump is associated with an index one greater than the previous one. The index can be used to associate the clump with a particular track and with the information given in other track-related clumps.

Pictures can appear in the clump in any order. This ensures that disposable pictures, such as conventional B pictures, can be located flexibly. Data for different pictures shall not overlap. Data for a picture consists of payloads, i.e., slices, data partitions, and pieces of supplemental enhancement information. Payloads shall appear in successive bytes, and the order of payloads shall be the same as in the Alternate Track Header Clump.

III.5.8.2 Syntax

```
aligned(8) class AlternateTrackMediaClump extends Clump('atrm') {
}
```

III.5.9 Switch Picture Clump

III.5.9.1 Definition

Clump Type: 'swpc'

Container: Segment Clump ('segm')

Mandatory: No

Quantity: Zero or one

This clump defines which pictures can be used to switch from an alternate track to another. Typically these pictures are SP pictures.

III.5.9.2 Syntax

```
aligned(8) class uniquePicture {
    unsigned int(8) alternateTrackIndex;
    unsigned int((numBytesInPictureCountMinusOne + 1) * 8)
pictureIndex;
}

aligned(8) class switchPictureSet {
    unsigned int(8) numSyncPictures;
    (class uniquePicture) syncPicture[numSyncPictures];
}

aligned(8) class switchPictureClump extends Clump('swpc') {
    unsigned int((numBytesInPictureCountMinusOne + 1) * 8)
numSwitchPictures;
    (class switchPictureSet) switchPicture[numSwitchPictures];
}
```

III.5.9.3 Semantics

uniquePicture uniquely identifies a picture within this segment. It contains two attributes: alternateTrackIndex and pictureIndex. alternateTrackIndex identifies the alternate track where the picture lies, and pictureIndex gives the picture index in coding order.

switchPictureSet gives a set of pictures that represent the same picture contents and can be used to replace any picture in the set as a reference picture for motion compensation. numSyncPictures gives the number

of pictures in the set. syncPicture is an array of uniquePicture structures indicating which pictures belong to the set.

numSwitchPictures indicates the number of picture positions that have multiple interchangeable representations. switchPicture is an array of switchPictureSet structures indicating the set of pictures that can be used interchangeably for each picture position.

Appendix IV Non-Normative Error Concealment

IV.1 Introduction

It is assumed that no erroneous or incomplete slices are decoded. When all received slices of a picture have been decoded, skipped slices are concealed according to the presented algorithms. In practice, record is kept in a macroblock (MB) based status map of the frame. The status of an MB in the status map is "Correctly received" whenever the slice that the MB is included in was available for decoding, "Lost" otherwise. After the frame is decoded if the status map contains "Lost" MBs, concealment is started.

Given the slice structure and MB-based status map of a frame, the concealment algorithms were designed to work MB-based. The missing frame area (pixels) covered by MBs marked as "Lost" in the status map are concealed MB-by-MB (16x16 Y pixels, 8x8 U, V pixels). After an MB has been concealed it is marked in the status map as "Concealed". The order in which "Lost" MBs are concealed is important as also the "Concealed", and not only the "Correctly received" MBs are treated as reliable neighbors in the concealment process whenever no "Correctly received" immediate neighbor of a "Lost" MB exists. In such cases a wrong concealment can result in propagation of this concealment mistake to several neighbor concealed MBs. The processing order chosen is to take the MB columns at the edge of the frame first and then move inwards column-by-column so to avoid a concealment mistake made in the usually "difficult" (discontinuous motion areas, large coded prediction error) center part of the frame propagate to the "easy" (continuous motion area, similar motion over several frames) side parts of the frame.

FIGURE 25 shows a snapshot of the status map during the concealment phase where already concealed MBs have the status of "Concealed", and the currently processed (concealed) MB is marked as "Current MB".

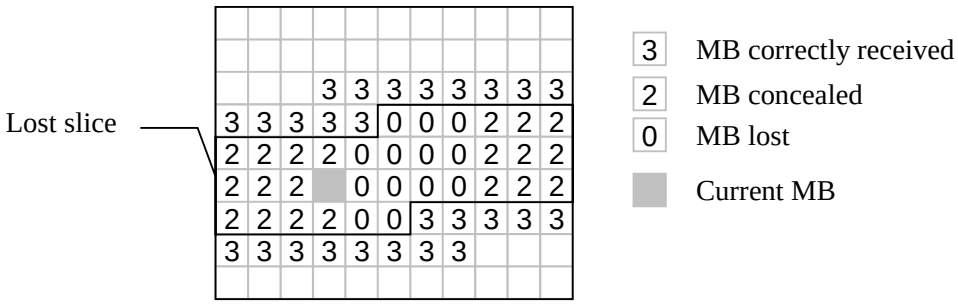


FIGURE 25

MB status map at the decoder

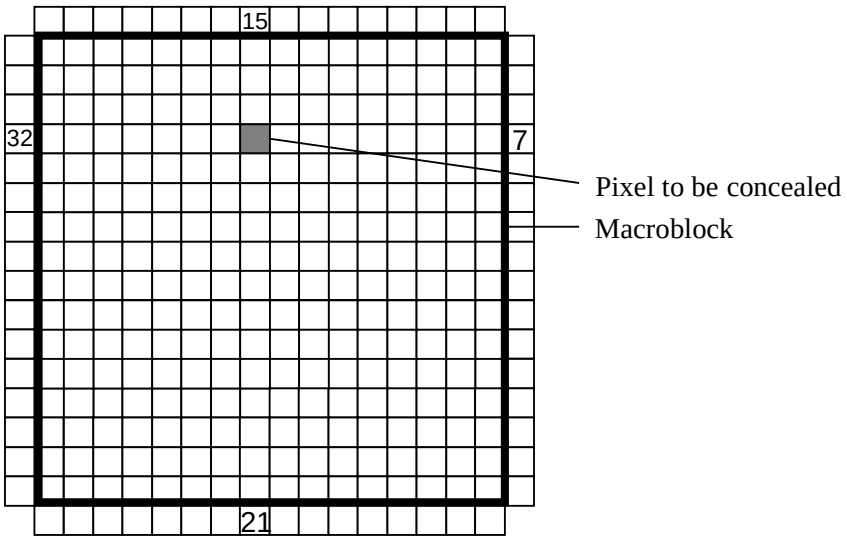
IV.2 INTRA Frame Concealment

Lost areas in INTRA frames have to be concealed spatially as no prior frame may resemble the INTRA frame. The selected spatial concealment algorithm is based on weighted pixel averaging presented in *A. K. Katsaggelos and N. P. Galatsanos (editors), "Signal Recovery Techniques for Image and Video Compression and Transmission", Chapter 7, P. Salama, N. B. Shroff, and E. J. Delp, "Error Concealment in Encoded Video Streams", Kluwer Academic Publishers, 1998.*

Each pixel value in a macroblock to be concealed is formed as a weighted sum of the closest boundary pixels of the selected adjacent macroblocks. The weight associated with each boundary pixel is relative to the inverse distance between the pixel to be concealed and the boundary pixel. The following formula is used:

Pixel value = $(\sum a_i \times (B-d_i)) / \sum (B-d_i)$

where a_i is the pixel value of a boundary pixel in an adjacent macroblock, B is the horizontal or vertical block size in pixels, and d_i is the distance between the destination pixel and the corresponding boundary pixel in the adjacent macroblock.



In FIGURE 26, the shown destination pixel is calculated as follows

Pixel value = (15x(16-3) + 21x(16-12) + 32x(16-7) + 7x(16-8)) / (13 + 4 + 9 + 8) ≈ 18

Only "Correctly received" neighboring MBs are used for concealment if at least two such MBs are available. Otherwise, neighboring "Concealed" MBs are also used in the averaging operation.

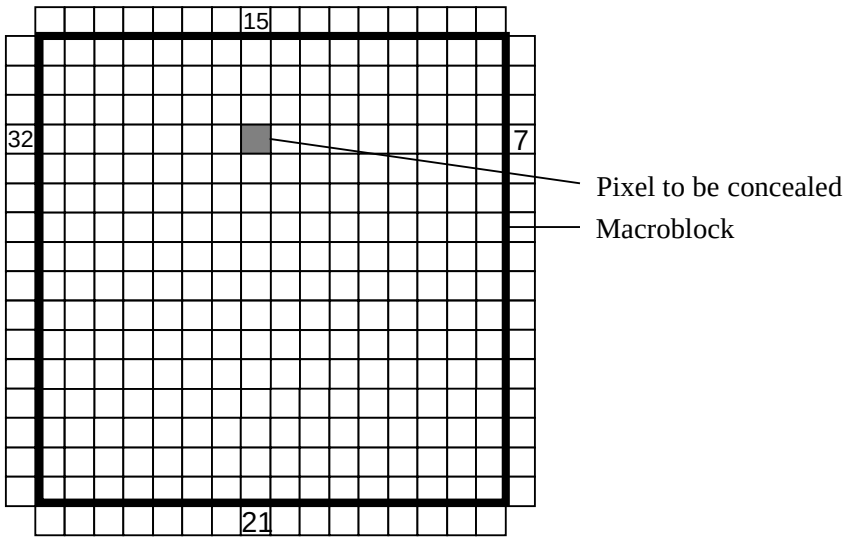


FIGURE 26

Spatial concealment based on weighted pixel averaging

IV.3 INTER and SP Frame Concealment

IV.3.1 General

Instead of directly operating in the pixel domain a more efficient approach is to try to "guess" the motion in the missing pixel area (MB) by some kind of prediction from available motion information of spatial or temporal neighbors. This "guessed" motion vector is then used for motion compensation using the reference frame. The copied pixel values give the final reconstructed pixel values for concealment, and no additional pixel domain operations are used. The presented algorithm is based on *W.-M. Lam, A. R. Reibman, and B. Liu, "Recovery of lost or erroneously received motion vectors," in Proc. ICASSP'93, Minneapolis, Apr. 1993, pp. V417–V420.*

IV.3.2 Concealment using motion vector prediction

The motion activity of the correctly received slices of the current picture is investigated first. If the average motion vector is smaller than a pre-defined threshold (currently ¼ pixels for each motion vector component), all the lost slices are concealed by copying from co-located positions in the reference frame. Otherwise, motion-compensated error concealment is used, and the motion vectors of the lost macroblocks are predicted as described in the following paragraphs.

The motion of a "Lost" MB is predicted from a spatial neighbor MB's motion relying on the statistical observation, that the motion of spatially neighbor frame areas is highly correlated. For example, in a frame area covered by a moving foreground scene object the motion vector field is continuous, which means that it is easy to predict.

The motion vector of the "Lost" MB is predicted from one of the neighbor MBs (or blocks). This approach assumes, that the motion vector of one of the neighbor MBs (or blocks) models the motion in the current MB well. It was found in previous experiments, that median or averaging over all neighbors' motion vectors does not give better results. For simplicity, in the current implementation the smallest neighbor block size that is considered separately as predictor is set to 8x8 Y pixels. The motion of any 8x8 block is calculated as the average of the motion of the spatially corresponding 4x4 or other shaped (e.g. 4x8) blocks.

The decision of which neighbor's motion vectors to use as prediction for the current MB is made based on the smoothness of the concealed (reconstructed) image. During this trial procedure the concealment pixel values are calculated using the motion vector of each candidate (motion compensated pixel values). The motion vector, which results in the smallest luminance change across block boundaries when the block is inserted into its place in the frame is selected. (see FIGURE 27). The zero motion vector case is always considered and this copy concealment (copy pixel values from the co-located MB in the reference frame) is evaluated similarly as the other motion vector candidates.

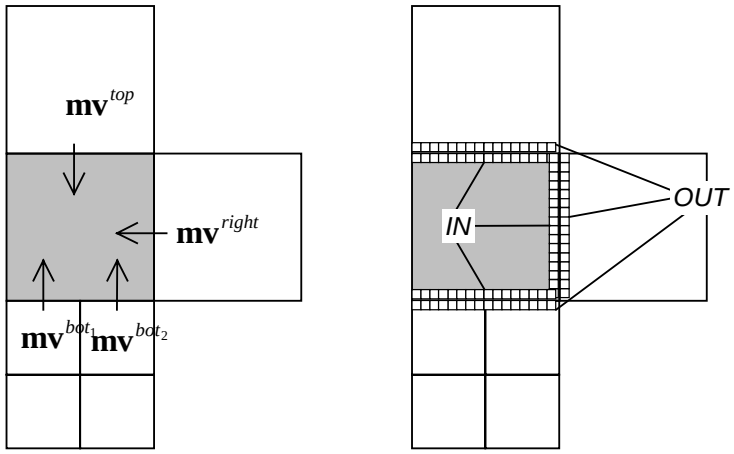


FIGURE 27

Selecting the motion vector for prediction

The winning predictor motion vector is the one which minimizes the side match distortion d_{sm}, which is the sum of absolute Y pixel value difference of the *IN*-block and neighboring *OUT*-block pixels at the boundaries of the current block:

$$\min_{dir\in\{top,bot,left,right\}} \arg \left\langle d_{sm} = \left(\sum_{j=1}^{N} \left| \hat{Y}(\mathbf{mv}^{dir})_j^{IN} - Y_j^{OUT} \right| \right) \middle/ N \right\rangle$$

When "Correctly received" neighbor MBs exist the side match distortion is calculated only for them. Otherwise all the "Concealed" neighbor MBs are included in the calculation.

IV.3.3 Handling of Multiple reference frames

When multiple references are used, the reference frame of the candidate motion vector is used as the reference frame for the current MB. That is, when calculating the side match distortion d_{sm}, the IN-block pixels are from the reference frame of the candidate motion vector.

IV.4 B Frame Concealment

A simple motion vector prediction scheme according to the prediction mode of the candidate MB is used as follows:

If the prediction mode of the candidate MB is

- forward prediction mode, use the forward MV as the prediction the same way as for P frames.
- backward prediction mode, use the backward MV as the prediction.
- bi-directional prediction mode, use the forward MV as the prediction, and discard the backward MV.
- direct prediction mode, use the backward MV as the prediction.

Note that 1) Each MV, whether forward or backward, has its own reference frame. 2) An Intra coded block is not used as a motion prediction candidate.

IV.5 Handling of Entire Frame Losses

TML currently lacks H.263 Annex U type of reference picture buffering. Instead, a simple sliding window buffer model is used, and a picture is referred using its index in the buffer. Consequently, when entire frames are lost, the reference buffer needs to be adjusted. Otherwise, the following received frames would use wrong reference frames. To solve this problem, the reference picture ID is used to infer how many frames are lost, and the picture indices in the sliding window buffer are shifted appropriately.