## ~~9~~8 B-pictures

### ~~9.1~~8.1 Introduction

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

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

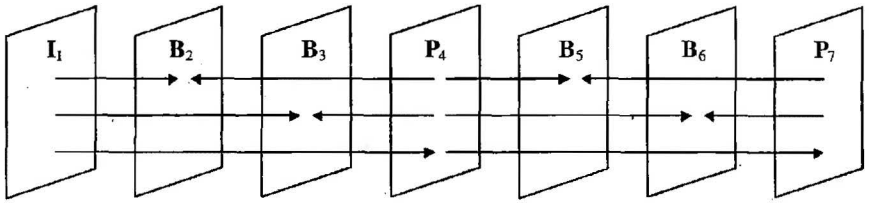



FIGURE 18

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

### ~~9.2~~8.2 Five Prediction modes

There are five different prediction modes supported by B pictures. They are direct, forward, backward, bi-directional and the intra prediction modes. Both direct mode and bi-directional mode are bi-directional prediction. The only difference is that the bi-directional mode uses separate motion vectors for forward and backward prediction, whereas the forward and backward motion vectors of the direct mode are derived from the motion vectors used in the corresponding macroblocks of the subsequent reference frame. In the direct mode, the same number of motion vectors are used as are used in the reference

macroblock for prediction. To calculate prediction blocks for the direct and bi-directional prediction mode, the forward and backward motion vectors are used to obtain appropriate blocks from reference frames and then these blocks are averaged by dividing the sum of the two prediction blocks by two.

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

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

### ~~9.3~~8.3 Finding optimum prediction mode

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

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

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

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

- Forward prediction mode :
  $SA(T)D0 = QP_0(QP)$ x (2xcode_number_of_Ref_frame + Bits_to_code_MVDFW)
- Backward prediction mode :
  $SA(T)D0 = QP_0(QP)$ x Bits_to_code_MVDBW
- Bi-directional prediction mode :
  $SA(T)D0 = QP_0(QP)$ x (2xcode_number_of_Ref_frame + Bits_to_code_forward_Blk_size +
  Bits_to_code_backward_Blk_size +
  Bits_to_code_MVDFW +
  Bits_to_code_MVDBW)
- Direct prediction mode :
  $SA(T)D = SA(T)D - 16 \times QP_0(QP)$
- 4x4 Intra mode :
  $SA(T)D0 = QP_0(QP)$ x Order_of_prediction_mode
- $SA(T)D = SA(T)D + 24 \times QP_0(QP)$

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

### ~~9.4~~8.4 Syntax

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

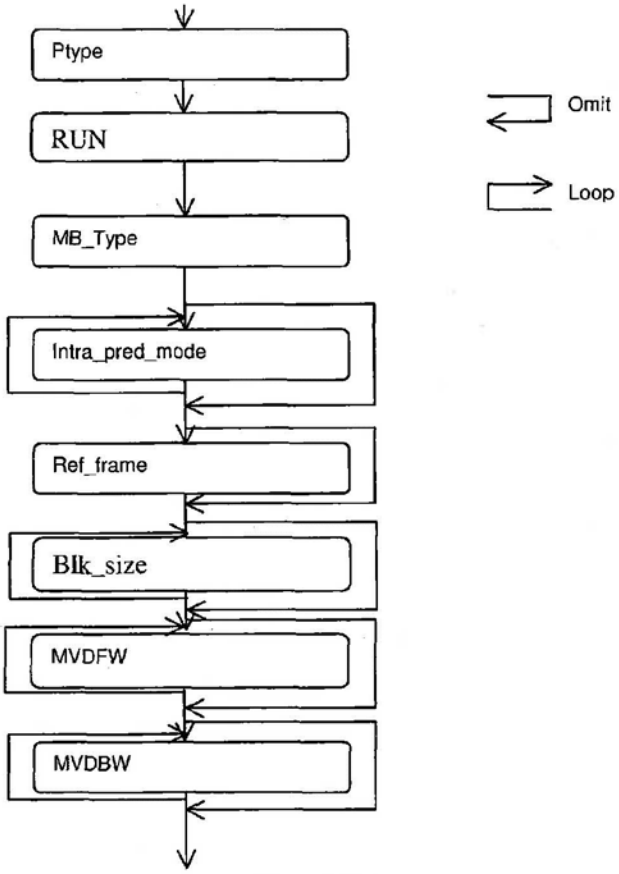



FIGURE 19

**Syntax diagram for B pictures.**

### ~~9.4.1~~8.4.1 Picture type (Ptype) and RUN

See sections 0 and 3.3 for definitions.

### ~~9.4.2~~8.4.2 Macro block type (MB_type)

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

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

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

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

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

**~~9.4.3~~8.4.3 Intra prediction mode (Intra_pred_mode)**

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

**~~9.4.4~~8.4.4 Reference Frame (Ref_frame)**

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

**TABLE 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, e.g., when Ptype=4 the Ref_frame field is present.

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

**TABLE 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 |
| ... | ... |

### ~~9.4.5~~8.4.5 Block Size (Blk_size)

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

**TABLE 13**

**Code_number for Blk_size**

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

### ~~9.4.6~~8.4.6 Motion vector data (MVDFW, MVDBW)

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

## ~~9.5~~8.5 Decoder Process for motion vector

### ~~9.5.1~~8.5.1 Differential motion vectors

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

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

### ~~9.5.2~~8.5.2 Motion vectors in direct mode

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

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

$MV_F = (TR_B * MV) / TR_D$

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

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

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

## ~~10~~9 SP-pictures

### ~~10.1~~9.1 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.

### ~~10.2~~9.2 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.

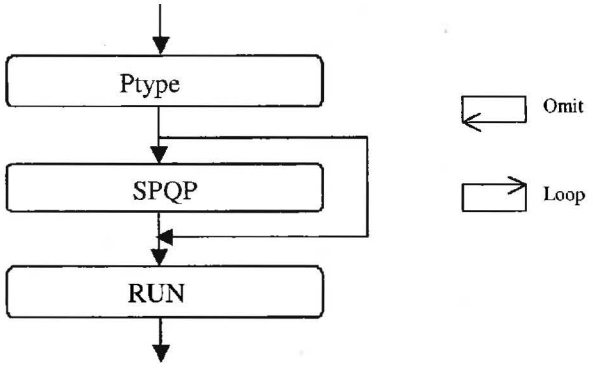



**FIGURE 20**

**Syntax diagram for SP-frames**

### ~~10.3~~9.3 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. 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 and 4.1, 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 4.5.

Since an additional 2x2 transform is performed for the DC coefficients of chroma blocks in a macroblock after 4x4 transform of each, Section 4.2, the decoding of the chroma component is performed in the similar manner as described above with the following difference: For chroma macroblocks in SP-frames, an additional 2x2 transform is applied after 4x4 transform of the predicted chroma blocks. And then the steps described above are repeated for the DC coefficients as well as for the AC coefficients. Note also that for chroma blocks, the values of $QP_1$ and $QP_2$ are both changed according to the relation between QP values of luma and chroma specified in Section 4.3.4.

## ~~8~~10 Hypothetical Reference Decoder

### ~~8.1~~10.1 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.

### ~~8.2~~10.2 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 10.3. 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.

### ~~8.3~~10.3 Decoding Time of a Picture

The decoding time $t_i$ of picture $i$ is equal to its presentation time $\tau_i$, 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 $\tau_i'$ 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.

### ~~8.4~~10.4 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.

### ~~8.5~~10.5 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.

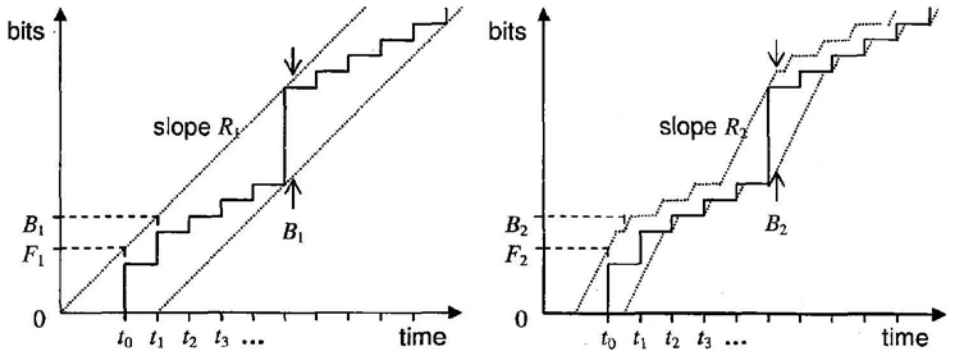



FIGURE 21

**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.

### ~~8.6~~10.6 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.

### ~~8.7~~10.7 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 $F_{min}(R) / R$.

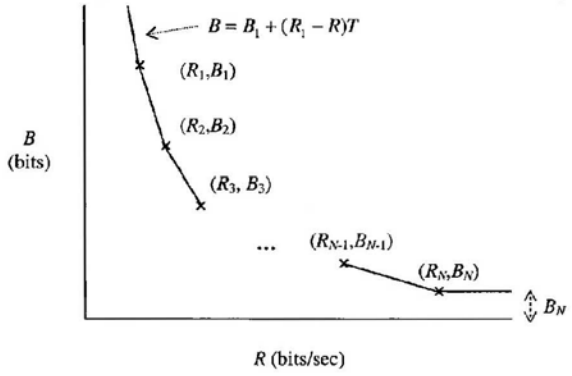



FIGURE 22

**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 $F \geq F_{min}(R)$.

### ~~8.8~~10.8 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 10.5.

## ~~12~~11 Supplemental Enhancement Information

### ~~12.1~~11.1 Syntax

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.

# 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 FIGURE 23 is run through. The different elements will be described below.

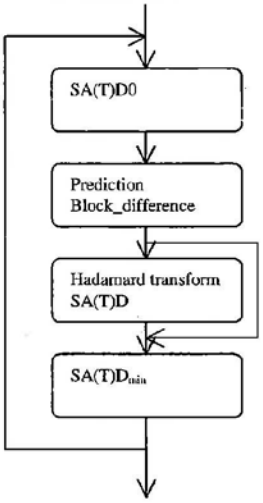



FIGURE 23

**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 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: $SA(T)D0 = QP_0(QP) \times Order\_of\_prediction\_mode$ (see above)

Motion vector search: $SA(T)D0 = QP_0(QP) \times (Bits\_to\_code\_vector + 2 \times code\_number\_of\_ref\_frame)$

In addition there are two special cases:

- For motion prediction of a 16x16 block with 0 vector components, $16 \times QP_0(QP)$ is subtracted from SA(T)D to favour the skip mode.
- For the whole intra macroblock, $24 \times QP_0(QP)$ 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.

##### 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.

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

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

### 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 FIGURE 23 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

TABLE 14 gives the table of intra prediction modes according to probability of each mode to be used on the decoder side. On the encoder side we need a sort of inverse table. Prediction modes for A and B are known as in

TABLE 1. For the encoder we have found a Mode that we want to signal with an ordering number in the bitstream (whereas on the decoder we receive the order in the bitstream and want to convert this to a mode). TABLE 14 is therefore the relevant table for the encoder. Example: Prediction mode for A and B is 2. The string in TABLE 14 is 2 1 0 3 4 5. This indicates that prediction mode 0 has order 2 (third most probable). Prediction mode 1 is second most probable and prediction mode 2 has order 0 (most probable) etc. As in

TABLE 1 '–' indicates that this instance can not occur because A or B or both are outside the picture.

**TABLE 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.

#### 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 upwords for the first positions are listed below:

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

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

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

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

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

#### 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 3.7.1 - is automatically checked).

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

#### 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 $\lambda_{MOTION}$ 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 I.1.1.1.3). The Lagrangian multiplier $\lambda_{MOTION}$ is given by

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

P-frames and

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

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

#### 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, $\lambda_{MODE}$ is the Lagrange multiplier for mode decision, and $MODE$ indicates a mode chosen from the set of potential prediction:

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

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

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

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

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

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 $\lambda_{MODE}$ 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, $\lambda_{MODE}$ 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, $\lambda_{MODE}$, $\lambda_{MOTION}$, and the macroblock quantizer $QP$
2. Choose intra prediction modes for the *INTRA 4x4* macroblock mode by minimizing
   $J(s,c,IMODE \mid QP,\lambda_{MODE}) = SSD(s,c,IMODE \mid QP) + \lambda_{MODE} \cdot R(s,c,IMODE \mid QP)$
   with $IMODE \in \{DC, HOR, VERT, DIAG, DIAG\_RL, DIAG\_LR\}$.
3. Determine the best $INTRA16x16$ prediction mode by choosing the mode which results in the minimum SATD.
4. Perform motion estimation and reference frame selection by minimizing
   $J(REF,\mathbf{m}(REF) \mid \lambda_{MOTION}) = SA(T)D(s,c(REF,\mathbf{m}(REF))) + \lambda_{MOTION} \cdot (R(\mathbf{m}(REF) - \mathbf{p}(REF)) + R(REF))$
   for each reference frame and motion vector of a possible macroblock mode.
5. Choose the macroblock prediction mode by minimizing
   $J(s,c,MODE \mid QP,\lambda_{MODE}) = SSD(s,c,MODE \mid QP) + \lambda_{MODE} \cdot R(s,c,MODE \mid QP)$,
   given $QP$ and $\lambda_{MODE}$ 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) + fx$2^{20}$)/$2^{20}$ |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 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 mcroblock in a P picture, the set of possible macroblock types is given as

$$S_{MB} = \{ \text{COPY\_MB, INTER\_16x16\_MB, INTER\_16x8\_MB, INTER\_8x16\_MB, INTER\_8x8\_MB, INTER\_8x4\_MB, INTER\_4x8\_MB, INTER\_4x4\_MB, INTRA\_4x4, INTRA\_16x16} \}$$

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

$$m' = \min_{m \in S_{MB}} \{ D_m + \lambda R_m \}$$

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_?x?_MB types, the distortion $D_m$ is computed as

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

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_?x?_MB types increases making INTRA_?x?_MB types more popular.

The $\lambda$ 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 Luma AC Coefficients
- TYPE_COEFF_C Chroma AC Coefficients

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

### 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 forbiddcn)

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.

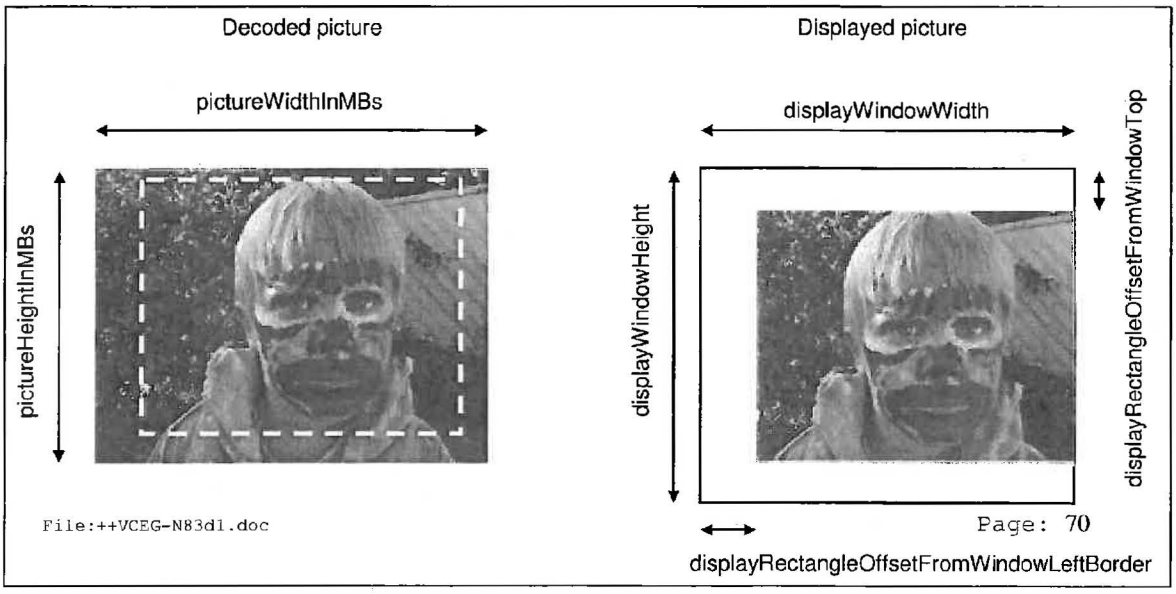


FIGURE 24

**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".

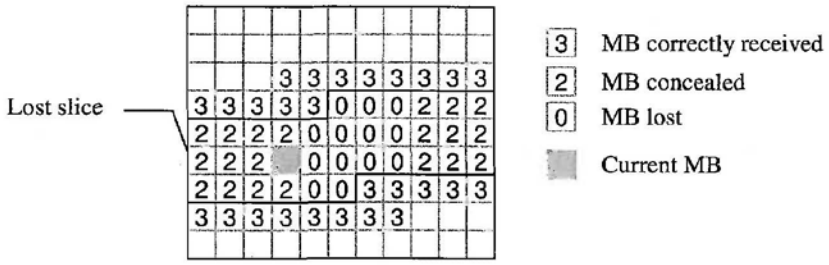



**FIGURE 25**

**MB status map at the decoder**

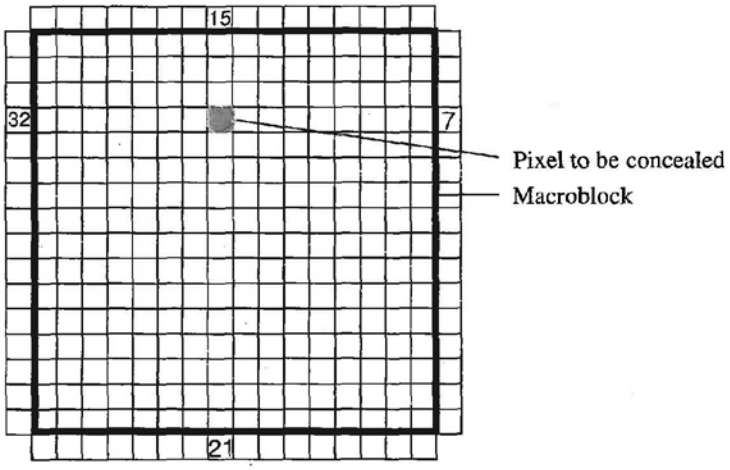
## IV.2 INTRA Frame Concealment

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.

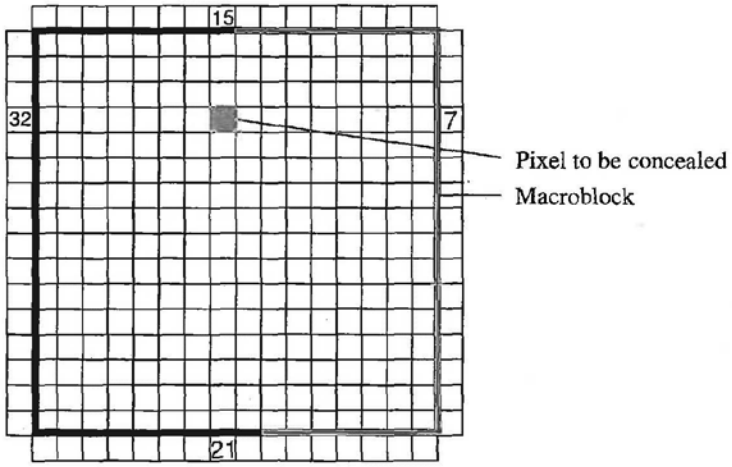



**FIGURE 26**

**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.

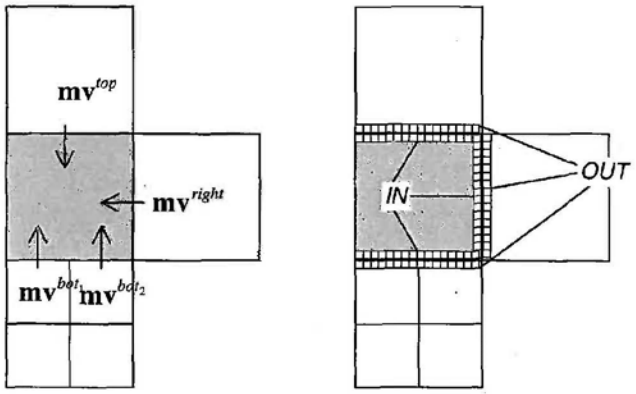



**FIGURE 27**

**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:

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

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.

E. 128

### 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.