**top_field_first**: This is a 1-bit flag which when set to "1" indicates that the top field (i.e., the field containing the top line) of reconstructed VOP is the first field to be displayed (output by the decoding process). When top_field_first is set to "0" it indicates that the bottom field of the reconstructed VOP is the first field to be displayed.

**alternate_vertical_scan_flag**: This is a 1-bit flag which when set to "1" indicates the use of alternate vertical scan for interlaced VOPs.

**sprite_transmit_mode**: This is a 2-bit code which signals the transmission mode of the sprite object. At video object layer initialization, the code is set to "piece" mode. When all object and quality update pieces are sent for the entire video object layer, the code is set to the "stop"mode. When an object piece is sent, the code is set to "piece" mode. When an update piece is being sent, the code is set to the "update" mode. When all sprite object pieces andquality update pieces for the current VOP are sent, the code is set to "pause" mode. Table 6-26 shows the different sprite transmit modes.

**Table 6-26 — Meaning of sprite transmit modes**

| code | sprite_transmit_mode |
|---|---|
| 00 | stop |
| 01 | piece |
| 10 | update |
| 11 | pause |

**vop_quant**: This is an unsigned integer which specifies the absolute value of quantiser scale to be used for inverse quantising the macroblock until updated by any subsequent dquant, dbquant, or quant_scale. The length of this field is specified by the value of the parameter quant_precision. The default length is 5-bits which carries the binary representation of quantiser scale values from 1 to 31 in steps of 1.

**vop_alpha_quant[i]**: This is a an unsigned integer which specifies the absolute value of the initial alpha plane quantiser scale to be used for inverse quantising macroblock grayscale alpha data in alpha or auxiliary component i=0,1,2. The alpha plane quantiser scale cannot be less than 1.

**vop_fcode_forward**: This is a 3-bit unsigned integer taking values from 1 to 7; the value of zero is forbidden. It is used in decoding of motion vectors.

**vop_fcode_backward**: This is a 3-bit unsigned integer taking values from 1 to 7; the value of zero is forbidden. It is used in decoding of motion vectors.

**vop_shape_coding_type**: This is a 1 bit flag which specifies whether inter shape decoding is to be carried out for the current P, B, or S(GMC)-VOP. If vop_shape_coding_type is equal to '0', intra shape decoding is carried out, otherwise inter shape decoding is carried out.

Coded data for the top-left macroblock of the bounding rectangle of a VOP shall immediately follow the VOP header, followed by the remaining macroblocks in the bounding rectangle in the conventional left-to-right, top-to-bottom scan order. Video packets shall also be transmitted following the conventional left-to-right, top-to-bottom macroblock scan order. The last MB of one video packet is guaranteed to immediately precede the first MB of the following video packet in the MB scan order.

**load_backward_shape**: This is a one-bit flag which when set to '1' implies that the backward shape of the previous VOP in the same layer is copied to the forward shape for the current VOP and the backward shape of the current VOP is decoded from the bitstream. When this flag is set to '0', the forward shape of the previous VOP is copied to the forward_shape of the current VOP and the backward shape of the previous VOP in the same layer is copied to the backward shape of the current VOP. This flag shall be '1' when (1) background_composition is '1' and vop_coded of the previous VOP in the same layer is '0' or (2) background_composition is '1' and the current VOP is the first VOP in the current layer.

If hierarchy_type is "0" and video_object_layer_shape is "binary" (object based spatial scalability), then this flag shall be set to "0".


© ISO/IEC 2004 – All rights reserved

**backward_shape_width**: This is a 13-bit unsigned integer which specifies the horizontal size, in pixel units, of the rectangle that includes the backward shape. A zero value is forbidden.

**backward_shape_height**: This is a 13-bit unsigned integer which specifies the vertical size, in pixel units, of the rectangle that includes the backward shape. A zero value is forbidden.

**backward_shape_horizontal_mc_spatial_ref**: This is a 13-bit signed integer which specifies, in pixel units, the horizontal position of the top left of the rectangle that includes the backward shape. This is used for decoding and for picture composition.

**backward_shape_vertical_mc_spatial_ref**: This is a 13-bit signed integer which specifies, in pixel units, the vertical position of the top left of the rectangle that includes the backward shape. This is used for decoding and for picture composition.

backward_shape(): The decoding process of the backward shape is identical to the decoding process for the shape of I-VOP with binary only mode (video_object_layer_shape = "10").

**load_forward_shape**: This is a one-bit flag which when set to '1' implies that the forward shape is decoded from the bitstream. This flag shall be '1' when (1) background_composition is '1' and vop_coded of the previous VOP in the same layer is '0' or (2) background_composition is '1' and the current VOP is the first VOP in the current layer.

**forward_shape_width**: This is a 13-bit unsigned integer which specifies the horizontal size, in pixel units, of the rectangle that includes the forward shape. A zero value is forbidden.

**forward_shape_height**: This is a 13-bit unsigned integer which specifies the vertical size, in pixel units, of the rectangle that includes the forward shape. A zero value is forbidden.

**forward_shape_horizontal_mc_spatial_ref**: This is a 13-bit signed integer which specifies, in pixel units, the horizontal position of the top left of the rectangle that includes the forward shape. This is used for decoding and for picture composition.

**forward_shape_vertical_mc_spatial_ref**: This is a 13-bit signed integer which specifies, in pixel units, the vertical position of the top left of the rectangle that includes the forward shape. This is used for decoding and for picture composition.

forward_shape(): The decoding process of the backward shape is identical to the decoding process for the shape of I-VOP with binary only mode (video_object_layer_shape = "10").

**ref_select_code**: This is a 2-bit unsigned integer which specifies prediction reference choices for P- and B-VOPs in enhancement layer with respect to decoded reference layer identified by ref_layer_id. The meaning of allowed values is specified in Table 7-17 and Table 7.18.

**resync_marker**: This is a binary string of at least 16 zero's followed by a one '0 0000 0000 0000 0001'. For an I-VOP or a VOP where video_object_layer_shape has the value "binary_only", the resync marker is 16 zeros followed by a one. The length of this resync marker is dependent on the value of vop_fcode_forward, for a P-VOP or a S(GMC)-VOP, and the larger value of either vop_fcode_forward and vop_fcode_backward for a B-VOP. For a P-VOP and a S(GMC)-VOP, the resync_marker is (15+fcode) zeros followed by a one; for a B-VOP, the resync_marker is max(15+fcode,17) zeros followed by a one. It is only present when resync_marker_disable flag is set to '0'. A resync marker shall only be located immediately before a macroblock and aligned with a byte.

**macroblock_number**: This is a variable length code with length between 1 and 14 bits. It identifies the macroblock number within a VOP. The number of the top-left macroblock in a VOP shall be zero. The macroblock number increases from left to right and from top to bottom. The actual length of the code depends on the total number of macroblocks in the VOP calculated according to Table 6-27, the code itself is a binary representation of the macroblock number. If reduced_resolution_vop_enable is equal to one, the length of macroblock_number code shall be determined by as if the total number of macroblocks in a VOP in Table 6-27 be equal to ((video_object_layer_width+15)/16) * ((video_object_layer_height+15)/16).

© ISO/IEC 2004 – All rights reserved

**Table 6-27 — Length of macroblock_number code**

| length of macroblock_number code | ((vop_width+15)/16) * ((vop_height+15)/16) |
|---|---|
| 1 | 1-2 |
| 2 | 3-4 |
| 3 | 5-8 |
| 4 | 9-16 |
| 5 | 17-32 |
| 6 | 33-64 |
| 7 | 65-128 |
| 8 | 129-256 |
| 9 | 257-512 |
| 10 | 513-1024 |
| 11 | 1025-2048 |
| 12 | 2049-4096 |
| 13 | 4097-8192 |
| 14 | 8193-16384 |

**quant_scale**: This is an unsigned integer which specifies the absolute value of quantiser scale to be used for inverse quantising the macroblock of the video packet until updated by any subsequent dquant. The length of this field is specified by the value of the parameter quant_precision. The default length is 5-bits.

**header_extension_code**: This is a 1-bit flag which when set to '1' indicates the presence of additional fields in the header. When header_extension_code is set to '1', modulo_time_base, vop_time_increment and vop_coding_type are also included in the video packet header. If video_object_layer_shape is not "rectangular", VOP header fields used for the shape decoding (vop_width, vop_height, vop_horizontal_mc_spatial_ref, vop_vertical_mc_spatial_ref, change_conv_ratio_disable and vop_shape_coding_type) are also present. If video_object_layer_shape is not "binary only", intra_dc_vlc_thr is also present. Furthermore, if the vop_coding_type is equal to either a P, S or B VOP, the appropriate fcodes are also present. Additionally, if the current VOP is an S(GMC)-VOP, sprite_trajectory() is included. And if reduced_resolution_vop_enable is equal to one, vop_reduced_resolution is also present.

**use_intra_dc_vlc**: The value of this internal flag is set to 1 when the values of intra_dc_thr and the DCT quantiser indicate the usage of the intra DC VLCs as shown in Table 6-25. Otherwise, the value of this flag is set to 0.

**motion_marker**: This is a 17-bit binary string '1 1111 0000 0000 0001'. It is only present when the data_partitioned flag is set to '1'.In the data partitioning mode, a motion_marker is inserted after the motion data (prior to the texture data). The motion_marker is unique from the motion data and enables the decoder to determine when all the motion information has been received correctly.

**dc_marker**: This is a 19 bit binary string '110 1011 0000 0000 0001'. It is present when the data_partitioned flag is set to '1'. It is used for I-VOPs. In the data partitioning mode, a dc_marker is inserted into the bitstream after the mcbpc, dquant and DC data but before the ac_pred flag and remaining texture information.

**vop_id**: This indicates the ID of VOP which is incremented by 1 whenever a VOP is encoded. All '0' value is skipped in vop_id in order to prevent the resync_marker emulation. The bit length of vop_id is the smaller value between (the length of vop_time_increment) + 3 and 15.

**vop_id_for_prediction_indication**: This is a one-bit flag which indicates the existence of the following vop_id_for_prediction field.

0: vop_id_for_prediction does not exist.
1: vop_id_for_prediction does exist.


© ISO/IEC 2004 – All rights reserved

**vop_id_for_prediction**: This indicates the vop_id of the VOP which is used as the reference VOP of the decoding of the current VOP or Video Packet. When this field exists, the reference VOP shall be replaced with the indicated one. This VOP which is used for prediction may be changed according to the backward channel message. If this field does not exist, the previous VOP is used for prediction, or all MBs in the VOP or Video Packet are coded to Intra MB.

#### 6.3.5.1 Definition of DCECS variable values

The semantic of all complexity estimation parameters is defined at the VO syntax level. DCECS variables represent % values. The actual % values have been converted to 8 bit words by normalization to 256. To each 8 bit word a binary 1 is added to prevent start code emulation (i.e 0% = '00000001', 99.5% = '11111111' and is conventionally considered equal to one). The binary '00000000' string is a forbidden value. The only parameter expressed in their absolute value is the dcecs_vlc_bits parameter expressed as a 4 bit word.

**dcecs_opaque**: 8 bit number representing the % of luminance and chrominance blocks using opaque coding mode on the total number of blocks (bounding rectangle).

**dcecs_transparent**: 8 bit number representing the % of luminance and chrominance blocks using transparent coding mode on the total number of blocks (bounding rectangle).

**dcecs_intra_cae**: 8 bit number representing the % of luminance and chrominance blocks using IntraCAE coding mode on the total number of blocks (bounding rectangle).

**dcecs_inter_cae**: 8 bit number representing the % of luminance and chrominance blocks using InterCAE coding mode on the total number of blocks (bounding rectangle).

**dcecs_no_update**: 8 bit number representing the % of luminance and chrominance blocks using no update coding mode on the total number of blocks (bounding rectangle).

**dcecs_upsampling**: 8 bit number representing the % of luminance and chrominance blocks which need upsampling from 4-4- to 8-8 block dimensions on the total number of blocks (bounding rectangle).

**dcecs_intra_blocks**: 8 bit number representing the % of luminance and chrominance Intra or Intra+Q coded blocks on the total number of blocks (bounding rectangle).

**dcecs_not_coded_blocks**: 8 bit number representing the % of luminance and chrominance Non Coded blocks on the total number of blocks (bounding rectangle).

**dcecs_dct_coefs**: 8 bit number representing the % of the number of DCT coefficients on the maximum number of coefficients (coded blocks).

**dcecs_dct_lines**: 8 bit number representing the % of the number of DCT8x1 on the maximum number of DCT8x1 (coded blocks).

**dcecs_vlc_symbols**: 8 bit number representing the average number of VLC symbols for macroblock.

**dcecs_vlc_bits**: 4 bit number representing the average number of bits for each symbol.

**dcecs_inter_blocks**: 8 bit number representing the % of luminance and chrominance Inter and Inter+Q coded blocks on the total number of blocks (bounding rectangle).

**dcecs_inter4v_blocks**: 8 bit number representing the % of luminance and chrominance Inter4V coded blocks on the total number of blocks (bounding rectangle).

**dcecs_apm** (Advanced Prediction Mode): 8 bit number representing the % of the number of luminance block predicted using APM on the total number of blocks for VOP (bounding rectangle).

**dcecs_npm** (Normal Prediction Mode): 8 bit number representing the % of luminance and chrominance blocks predicted using NPM on the total number of luminance and chrominance blocks for VOP (bounding rectangle).

© ISO/IEC 2004 – All rights reserved

**dcecs_forw_back_mc_q**: 8 bit number representing the % of luminance and chrominance predicted blocks on the total number of blocks for VOP (bounding rectangle).

**dcecs_halfpel2**: 8 bit number representing the % of luminance and chrominance blocks predicted by a half-pel vector on one dimension (horizontal or vertical) on the total number of blocks (bounding rectangle).

**dcecs_halfpel4**: 8 bit number representing the % of luminance and chrominance blocks predicted by a half-pel vector on two dimensions (horizontal and vertical) on the total number of blocks (bounding rectangle).

**dcecs_interpolate_mc_q**: 8 bit number representing the % of luminance and chrominance interpolated blocks in % of the total number of blocks for VOP (bounding rectangle).

**dcecs_sadct**: 8 bit number representing the % of luminance and chrominance blocks coded using SADCT mode in % of the total number of blocks for VOP (bounding rectangle).

**dcecs_quarterpel**: 8 bit number representing the % of luminance and chrominance blocks predicted using quarter-pel vectors in % of the total number of blocks for VOP (bounding rectangle).

#### 6.3.5.2 Video Plane with Short Header

video_plane_with_short_header() – This data structure contains a video plane using an abbreviated header format. Certain values of parameters shall have pre-defined and fixed values for any video_plane_with_short_header, due to the limited capability of signaling information in the short header format. These parameters having fixed values are shown in Table 6-28.

**Table 6-28 — Fixed Settings for video_plane_with_short_header()**

| Parameter | Value |
|---|---|
| video_object_layer_shape | "rectangular" |
| obmc_disable | 1 |
| quant_type | 0 |
| resync_marker_disable | 1 |
| data_partitioned | 0 |
| block_count | 6 |
| reversible_vlc | 0 |
| vop_rounding_type | 0 |
| vop_fcode_forward | 1 |
| vop_coded | 1 |
| interlaced | 0 |
| complexity_estimation_disable | 1 |
| use_intra_dc_vlc | 0 |
| scalability | 0 |
| not_8_bit | 0 |
| bits_per_pixel | 8 |
| colour_primaries | 1 |
| transfer_characteristics | 1 |
| matrix_coefficients | 6 |

**short_video_start_marker**: This is a 22-bit start marker containing the value '0000 0000 0000 0000 1000 00'. It is used to mark the location of a video plane having the short header format. short_video_start_marker shall be byte aligned by the insertion of zero to seven zero-valued bits as necessary to achieve byte alignment prior to short_video_start_marker.


© ISO/IEC 2004 – All rights reserved

**temporal_reference**: This is an 8-bit number which can have 256 possible values. It is formed by incrementing its value in the previously transmitted video_plane_with_short_header() by one plus the number of non-transmitted pictures (at 30000/1001 Hz) since the previously transmitted picture. The arithmetic is performed with only the eight LSBs.

**split_screen_indicator**: This is a boolean signal that indicates that the upper and lower half of the decoded picture could be displayed side by side. This bit has no direct effect on the encoding or decoding of the video plane.

**document_camera_indicator**: This is a boolean signal that indicates that the video content of the VOP is sourced as a representation from a document camera or graphic representation, as opposed to a view of natural video content. This bit has no direct effect on the encoding or decoding of the video plane.

**full_picture_freeze_release**: This is a boolean signal that indicates that resumption of display updates should be activated if the display of the video content has been frozen due to errors, packet losses, or for some other reason such as the receipt of a external signal. This bit has no direct effect on the encoding or decoding of the video plane.

**source_format**: This is an indication of the width and height of the rectangular video plane represented by the video_plane_with_short_header. The meaning of this field is shown in Table 6-29. Each of these source formats has the same VOP time increment resolution which is equal to 30000/1001 (approximately 29.97) Hz and the same width:height pixel aspect ratio (288/3):(352/4), which equals 12:11 in relatively prime numbers and which defines a CIF picture as having a width:height picture aspect ratio of 4:3.

**Table 6-29 — Parameters Defined by source_format Field**

| source_format value | Source Format Meaning | vop_width | vop_height | num_macroblocks_in_gob | num_gobs_in_vop |
|---|---|---|---|---|---|
| 000 | reserved | reserved | reserved | reserved | reserved |
| 001 | sub-QCIF | 128 | 96 | 8 | 6 |
| 010 | QCIF | 176 | 144 | 11 | 9 |
| 011 | CIF | 352 | 288 | 22 | 18 |
| 100 | 4CIF | 704 | 576 | 88 | 18 |
| 101 | 16CIF | 1408 | 1152 | 352 | 18 |
| 110 | reserved | reserved | reserved | reserved | reserved |
| 111 | reserved | reserved | reserved | reserved | reserved |

**picture_coding_type**: This bit indicates the vop_coding_type. When equal to zero, the vop_coding_type is “I”, and when equal to one, the vop_coding_type is “P”.

**four_reserved_zero_bits**: This is a four-bit field containing bits which are reserved for future use and equal to zero.

**pei**: This is a single bit which, when equal to one, indicates the presence of a byte of psupp data following the pei bit.

**psupp**: This is an eight bit field which is present when pei is equal to one. The pei + psupp mechanism provides for a reserved method of later allowing the definition of backward-compatible data to be added to the bitstream. Decoders shall accept and discard psupp when pei is equal to one, with no effect on the decoding of the video data. The pei and psupp combination pair may be repeated if present. The ability for an encoder to add pei and psupp to the bitstream is reserved for future use.

**gob_number**: This is a five-bit number which indicates the location of video data within the video plane. A group of blocks (or GOB) contains a number of macroblocks in raster scanning order within the picture. For a given gob_number, the GOB contains the num_macroblocks_per_gob macroblocks starting with macroblock_number = gob_number * num_macroblocks_per_gob. The gob_number can either be read from the bitstream or inferred from the progress of macroblock decoding as shown in the syntax description pseudo-code.

© ISO/IEC 2004 – All rights reserved

**num_gobs_in_vop**: This is the number of GOBs in the VOP. This parameter is derived from the source_format as shown in Table 6-29.

gob_layer(): This is a layer containing a fixed number of macroblocks in the VOP. Which macroblocks which belong to each gob can be determined by gob_number and num_macroblocks_in_gob.

**gob_resync_marker**: This is a fixed length code of 17 bits having the value '0000 0000 0000 0000 1' which may optionally be inserted at the beginning of each gob_layer(). Its purpose is to serve as a type of resynchronization marker for error recovery in the bitstream. The gob_resync_marker codes may (and should) be byte aligned by inserting zero to seven zero-valued bits in the bitstream just prior to the gob_resync_marker in order to obtain byte alignment. The gob_resync_marker shall not be present for the first GOB (for which gob_number = 0).

**gob_number**: This is a five-bit number which indicates which GOB is being processed in the VOP. Its value may either be read following a gob_resync_marker or may be inferred from the progress of macroblock decoding. All GOBs shall appear in the bitstream of each video_plane_with_short_header(), and the GOBs shall appear in a strictly increasing order in the bitstream. In other words, if a gob_number is read from the bitstream after a gob_resync_marker, its value must be the same as the value that would have been inferred in the absence of the gob_resync_marker.

**gob_frame_id**: This is a two bit field which is intended to help determine whether the data following a gob_resync_marker can be used in cases for which the VOP header of the video_plane_with_short_header() may have been lost. gob_frame_id shall have the same value in every GOB header of a given video_plane_with_short_header(). Moreover, if any field among the split_screen_indicator or document_camera_indicator or full_picture_freeze_release or source_format or picture_coding_type as indicated in the header of a video_plane_with_short_header() is the same as for the previous transmitted picture in the same video object, gob_frame_id shall have the same value as in that previous video_plane_with_short_header(). However, if any of these fields in the header of a certain video_plane_with_short_header() differs from that in the previous transmitted video_plane_with_short_header() of the same video object, the value for gob_frame_id in that picture shall differ from the value in the previous picture.

**num_macroblocks_in_gob**: This is the number of macroblocks in each group of blocks (GOB) unit. This parameter is derived from the source_format as shown in Table 6-29. The count of stuffing macroblocks is not included in this value.

**short_video_end_marker**: This is a 22-bit end of sequence marker containing the value '0000 0000 0000 0000 1111 11'. It is used to mark the end of a sequence of video_plane_with_short_header(). short_video_end_marker may (and should) be byte aligned by the insertion of zero to seven zero-valued bits to achieve byte alignment prior to short_video_end_marker.

#### 6.3.5.3 Shape coding

**bab_type**: This is a variable length code between 1 and 7 bits. It indicates the coding mode used for the bab. There are seven bab_types as depicted in Table 6-30. The VLC tables used depend on the decoding context i.e. the bab_types of blocks already received. For I-VOPs, the context-switched VLC table of Table B.27 is used. For P-VOPs, B-VOPs, and S(GMC)-VOPs , the context switched table of Table B.28 is used.

**Table 6-30 — List of bab_types and usage**

| bab_type | Semantic | Used in |
|---|---|---|
| 0 | MVDs==0 && No Update | P,B, and S(GMC)- VOPs |
| 1 | MVDs!=0 && No Update | P,B, and S(GMC)-VOPs |
| 2 | transparent | All VOP types |
| 3 | opaque | All VOP types |
| 4 | intraCAE | All VOP types |
| 5 | MVDs==0 && interCAE | P,B, and S(GMC)-VOPs |
| 6 | MVDs!=0 && interCAE | P,B, and S(GMC)-VOPs |


© ISO/IEC 2004 – All rights reserved

The bab_type determines what other information fields will be present for the bab shape. No further shape information is present if the bab_type = 0, 2 or 3. Opaque means that all pixels of the bab are part of the object. Transparent means that none of the bab pixels belong to the object. IntraCAE means the intra-mode CAE decoding will be required to reconstruct the pixels of the bab. No_update means that motion compensation is used to copy the bab from the previous VOP's binary alpha map. InterCAE means the motion compensation and inter_mode CAE decoding are used to reconstruct the bab. MVDs refers to the motion vector difference for shape.

**mvds_x**: This is a VLC code between 1 and 18 bits. It represents the horizontal element of the motion vector difference for the bab. The motion vector difference is in full integer precision. The VLC table is shown is Table B.29.

**mvds_y**: This is a VLC code between 1 and 18 bits. It represents the vertical element of the motion vector difference for the bab. The motion vector difference is in full integer precision. If mvds_x is '0', then the VLC table of Table B.30, otherwise the VLC table of Table B.29 is used.

**conv_ratio**: This is VLC code of length 1-2 bits. It specifies the factor used for sub-sampling the 16x16 pixel bab. The decoder must up-sample the decoded bab by this factor. The possible values for this factor are 1, 2 and 4 and the VLC table used is given in Table B.31.

**scan_type**: This is a 1-bit flag where a value of '0' implies that the bab is in transposed form i.e. the BAB has been transposed prior to coding. The decoder must then transpose the bab back to its original form following decoding. If this flag is '1', then no transposition is performed.

binary_arithmetic_code(): This is a binary arithmetic decoder representing the pixel values of the bab. This code may be generated by intra cae or inter cae depending on the bab_type. Cae decoding relies on the knowledge of intra_prob[] and inter_prob[], probability tables given in Annex B.

**enh_bab_type**: This is a variable length code between 1 and 3 bits. It indicates the bab coding mode used in binary shape enhancement layer coding. There are four enh_bab_types as depicted in Table 6-31. The VLC tables used depend on the decoding context i.e. the bab_types of blocks in lower reference layer.

**Table 6-31 — List of enh_bab_types and usage**

| enh_bab_type | Semantic | Used in |
|---|---|---|
| 0 | intra NOT_CODED | P-,B-VOPs |
| 1 | intra CODED | P-,B-VOPs |
| 2 | inter NOT_CODED | B- VOPs |
| 3 | inter CODED | B- VOPs |

The enh_bab_type determines what other information fields will be present for the bab shape. No further shape information is present if the enh_bab_type = 0 or 2. NOT_CODED means that motion compensation is used to copy the bab from the reference bab's binary alpha map. In intra NOT_CODED mode, the upsampled bab from the collocated block in lower reference layer is used for the reference bab. In inter NOT_CODED mode, motion compensated bab in the previous VOP of the current layer is used for the reference bab. And the motion vector of the collocated block in the lower reference layer is used for the motion compensation. Each component of the lower layer shape motion vectors are scaled by the up-sampling ratio according to subclause 7.5.4.5 for referencing in the enhancement layer. Intra CODED means Scan Interleaving(SI) based CAE decoding will be required to reconstruct the pixels of the current bab. Inter CODED means the motion compensation and inter_mode CAE decoding are used to reconstruct the current bab.

enh_binary_arithmetic_code() – This is a binary arithmetic decoder representing the pixel values of the bab and bab type of Scan Interleaving (SI) method. This code may be generated by SI decoding method or inter cae depending on the enh_bab_type. When enh_bab_type==1, the first decoded value represents the bab type of SI decoding method ("0": transitional bab, "1": exceptional bab). And the other decoded values represent the pixel values of the bab. If the bab type of SI is "transitional bab", only transitional pixels in the coded-scan-lines are decoded. Otherwise, for "exceptional bab", all of the pixels in the coded-scan-lines are decoded. This binary value decoding relies on the knowledge of enh_bab_type_prob[], and enh_intra_v_prob[] and

© ISO/IEC 2004 – All rights reserved

enh_intra_h_prob[]probability tables given in Annex B. When enh_bab_type==3, this binary arithmetic decoder represents the pixel values of the bab. This code is generated by inter cae. This binary value decoding relies on the knowledge of inter_prob[], probability tables given in Annex B.

#### 6.3.5.4 Sprite coding

warping_mv_code(dmv : The codeword for each differential motion vector consists of a VLC indicating the length of the dmv code (dmv_length) and a FLC, dmv_code-, with dmv_length bits. The codewords are listed in Table B.33.

brightness_change_factor (): The codeword for brightness_change_factor consists of a variable length code denoting brightness_change_factor_size and a fix length code, brightness_change_factor, of brightness_change_factor_size bits (sign bit included). The codewords are listed in Table B.34.

send_mb():This function returns 1 if the current macroblock has already been transmitted. Otherwise it returns 0.

**piece_quant**: This is a 5-bit unsigned interger which indicates the quantiser scale to be used for a sprite-piece until updated by a subsequent dquant. The piece_quant carries the binary representation of quantiser scale values from 1 to 31 in steps of 1.

**piece_width**: This value specifies the width of the sprite piece measured in macroblock units.

**piece_height**: This value specifies the height of the sprite piece measured in macroblock units.

**piece_xoffset**: This value specifies the horizontal offset location, measured in macroblock units from the left edge of the sprite object, for the placement of the sprite piece into the sprite object buffer at the decoder.

**piece_yoffset**: This value specifies the vertical offset location, measured in macroblock units from the top edge of the sprite object.

decode_sprite_piece (): It decodes a selected region of the sprite object or its update. It also decodes the parameters required by the decoder to properly incorporate the pieces. All the static-sprite-object pieces will be encoded using a subset of the I-VOP syntax. And the static-sprite-update pieces use a subset of the P-VOP syntax. The sprite update is defined as the difference between the original sprite texture and the reconstructed sprite assembled from all the sprite object pieces.

sprite_shape_texture(): For the static-sprite-object pieces, shape and texture are coded using the macroblock layer structure in I-VOPs. And the static-sprite-update pieces use the P-VOP inter-macroblock syntax -- except that there are no motion vectors and shape information included in this syntax structure. Macroblocks raster scanning is employed to encode a sprite piece; however, whenever the scan encounters a macroblock which has been part of some previously sent sprite piece, then the block is not coded and the corresponding macroblock layer is empty.

### 6.3.6 Macroblock related

**not_coded**: This is a 1-bit flag which signals if a macroblock is coded or not. When set to'1' it indicates that a macroblock is not coded and no further data is included in the bitstream for this macroblock (with the exception of alpha data that may be present). The decoder shall treat this macroblock as 'inter' with motion vector equal to zero and no DCT coefficient data for P-VOPs, and the decoder shall treat this macroblock as 'GMC macroblock (i.e. prediction using the global motion compensated image)' with no motion vector data and no DCT coefficient data for S(GMC)-VOPs. When set to '0' it indicates that the macroblock is coded and its data is included in the bitstream.

**mcbpc**: This is a variable length code that is used to derive the macroblock type and the coded block pattern for chrominance. It is always included for coded macroblocks. Table B.6 and Table B.7 list all allowed codes for mcbpc in I-, P-, and S(GMC)- VOPs respectively. The values of the column "MB type" in these tables are used as the variable "derived_mb_type" which is used in the respective syntax part for motion and texture decoding. In P-VOPs using the short video header format (i.e., when short_video_header is 1), mcbpc codes indicating macroblock type 2 shall not be used.


© ISO/IEC 2004 – All rights reserved

**mcsel**: This is a 1-bit flag that specifies the reference image of each macroblock in S-VOPs. This flag is present only when sprite_enable == "GMC," vop_coding_type == "S", and the macroblock type specified by mcbpc is "inter" or "inter+q". mcsel indicates whether the global motion compensated image or the previous reconstructed VOP is referred to for interframe prediction. This flag is set to '1' when GMC is used for the macroblock, and is set to '0' if local MC is used. If mcsel = "1", local motion vectors are not transmitted. The default value for mcsel is '1' (i.e. prediction using GMC ) when sprite_enable == "GMC," vop_coding_type == "S", and not_coded == '1'.

**ac_pred_flag**: This is a 1-bit flag which when set to '1' indicates that either the first row or the first column of AC coefficients are differentially coded for intra coded macroblocks.

**cbpy**: This variable length code represents a pattern of non-transparent luminance blocks with at least one non intra DC transform coefficient, in a macroblock. Table B.8 to Table B.11 indicate the codes and the corresponding patterns they indicate for the respective cases of intra- and inter-MBs.

**dquant**: This is a 2-bit code which specifies the change in the quantiser scale, for I-, P-, and S(GMC)-VOPs. Table 6-32 lists the codes and the differential values they represent. The value of the quantiser scale lies in range of 1 to $2^{quant\_precision}$-1; if the value of quantiser scale after adding dquant value is less than 1 or exceeds $2^{quant\_precision}$-1, it shall be correspondingly clipped to 1 and $2^{quant\_precision}$-1. If quant_precision takes its default value of 5, the range of allowed values for the quantiser scale is [1:31].

**Table 6-32 — dquant codes and corresponding values**

| dquant code | value |
|---|---|
| 00 | -1 |
| 01 | -2 |
| 10 | 1 |
| 11 | 2 |

**co_located_not_coded**: The value of this internal flag is set to 1 when the current VOP is a B-VOP, the future reference VOP is a P-VOP, and the co-located macroblock in the future reference VOP is skipped (i.e. coded as not_coded = '1'). Otherwise the value of this flag is set to 0. The co-located macroblock is the macroblock which has the same horizontal and vertical index with the current macroblock in the B-VOP. If the co-located macroblock lies outside of the bounding rectangle, this macroblock is considered to be not skipped.

**modb**: This is a variable length code present only in coded macroblocks of B-VOPs. It indicates whether mb_type and/or cbpb information is present for a macroblock. The codes for modb are listed in Table B.3.

**mb_type**: This variable length code is present only in coded macroblocks of B-VOPs. Further, it is present only in those macroblocks for which one motion vector is included. The codes for mb_type are shown in Table B.4 for B-VOPs for no scalability and in Table B.5 for B-VOPs with scalability. When mb_type is not present (i.e. modb=='1') for a macroblock in a B-VOP, the macroblock type is set to the default type. The default macroblock type for the enhancement layer of spatially scalable bitstreams (i.e. ref_select_code == '00' && scalability = '1') is "forward mc + Q". Otherwise, the default macroblock type is "direct".

**cbpb**: This is a 3 to 6 bit code representing coded block pattern in B-VOPs, if indicated by modb. Each bit in the code represents a coded/no coded status of a block; the leftmost bit corresponds to the top left block in the macroblock. For each non-transparent block with coefficients, the corresponding bit in the code is set to '1'. In case no coefficients are coded for all the non-transparent blocks in the macroblock, modb shall be set to the value indicating cbpb is not present (i.e. modb=='1' or '01') and cbpb shall not be included in the bitstream for this macroblock.

**dbquant**: This is a variable length code which specifies the change in quantiser scale for B-VOPs. Table 6-33 lists the codes and the differential values they represent. If the value of the quantiser scale after adding dbquant value is less than 1 or exceeds $2^{quant\_precision}$-1, it shall be correspondingly clipped to 1 and $2^{quant\_precision}$-1. If quant_precision takes its default value of 5, the range of allowed values for the quantzer for B-VOP is [1:31].

© ISO/IEC 2004 – All rights reserved

**Table 6-33 — dbquant codes and corresponding values**

| dbquant code | value |
|---|---|
| 10 | -2 |
| 0 | 0 |
| 11 | 2 |

**coda_i**: This is a one-bit flag which is set to "1" to indicate that all the values in the grayscale alpha macroblock are equal to 255 (AlphaOpaqueValue). When set to "0", this flag indicates that one or more 8x8 blocks are coded according to cbpa.

**ac_pred_flag_alpha**: This is a one-bit flag which when set to '1' indicates that either the first row or the first column of AC coefficients are to be differentially decoded for intra alpha macroblocks. It has the same effect for alpha as the corresponding luminance flag.

**cbpa**: This is the coded block pattern for grayscale alpha texture data. For I, P, S(GMC), and B VOPs, this VLC is exactly the same as the inter (P or S(GMC)) cbpy VLC described in Table B.8 – Table B.11. cbpa is followed by the alpha block data which is coded in the same way as texture block data. Note that grayscale alpha blocks with alpha all equal to zero (transparent) are not included in the bitstream.

**coda_pb**: This is a VLC indicating the coding status for P, S(GMC), or B alpha macroblocks. The semantics are given in the table below (Table 6-34). When this VLC indicates that the alpha macroblock is all opaque, this means that all values are set to 255 (AlphaOpaqueValue).

**Table 6-34 — coda_pb codes and corresponding values**

| coda_pb | Meaning |
|---|---|
| 1 | alpha residue all zero |
| 01 | alpha macroblock all opaque |
| 00 | alpha residue coded |

#### 6.3.6.1 MB Binary Shape Coding

**bab_type**: This defines the coding type of the current bab according to Table B.27 and Table B.28 for intra and inter mode, respectively.

**mvds_x**: This defines the size of the x-component of the differential motion vector for the current bab according to Table B.29.

**mvds_y**: This defines the size of the y-component of the differential motion vector for the current bab according to Table B-29 if mvds_x!=0 and according to Table B.30 if mvds_x==0.

**conv_ratio**: This defines the upsampling factor according to Table B.31 to be applied after decoding the current shape information.

**scan_type**: This defines according to Table 6-35 whether the current bordered to be decoded bab and the eventual bordered motion compensated bab need to be transposed.

**Table 6-35 — scan_type**

| scan_type | meaning |
|---|---|
| 0 | transpose bab as in matrix transpose |
| 1 | do not transpose |


© ISO/IEC 2004 – All rights reserved

binary_arithmetic_code(): This is a binary arithmetic decoder that defines the context dependent arithmetically to be decoded binary shape information. The meaning of the bits is defined by the arithmetic decoder according to subclause 7.5.3

**enh_bab_type**: This defines the coding type of the current bab in the enhancement layer according to Table 7.7 and Table 7.8 for P-VOP and B-VOP coding, respectively.

enh_binary_arithmetic_code(): This is a binary arithmetic decoder that defines the context dependent arithmetically to be decoded binary shape information in the enhancement layer. The meaning of the bits is defined by the arithmetic decoder according to subclause 7.5.3.

#### 6.3.6.2 Motion vector

**horizontal_mv_data**: This is a variable length code, as defined in Table B.12, which is used in motion vector decoding as described in subclause 7.6.3.

**vertical_mv_data**: This is a variable length code, as defined in Table B.12, which is used in motion vector decoding as described in subclause 7.6.3.

**horizontal_mv_residual**: This is an unsigned integer which is used in motion vector decoding as described in subclause 7.6.3. The number of bits in the bitstream for horizontal_mv_residual, r_size, is derived from either vop_fcode_forward or vop_fcode_backward as follows;

r_size = vop_fcode_forward - 1 or r_size = vop_fcode_backward - 1

**vertical_mv_residual**: This is an unsigned integer which is used in motion vector decoding as described in subclause 7.6.3. The number of bits in the bitstream for vertical_mv_residual, r_size, is derived from either vop_fcode_forward or vop_fcode_backward as follows;

r_size = vop_fcode_forward - 1 or r_size = vop_fcode_backward - 1

#### 6.3.6.3 Interlaced Information

**dct_type**: This is a 1-bit flag indicating whether the macroblock is frame DCT coded or field DCT coded. If this flag is set to "1", the macroblock is field DCT coded; otherwise, the macroblock is frame DCT coded. This flag is only present in the bitstream if the interlaced flag is set to "1" and the macroblock is coded (coded blcok pattern is non-zero) or intra-coded. Boundary blocks are always coded in frame-based mode.

**field_prediction**: This is a 1-bit flag indicating whether the macroblock is field predicted or frame predicted. This flag is set to '1' when the macroblock is predicted using field motion vectors. If it is set to '0' then frame prediction (16x16 or 8x8) will be used. This flag is only present when interlaced == '1' for the following types of macrloblocks: a macroblock in a P-VOP with derived_mb_type < 2; a non-direct mode macrloblock in a B-VOP; or a macroblocks in an S (GMC)-VOP with mcsel == '0'.

**forward_top_field_reference**: This is a 1-bit flag which indicates the reference field for the forward motion compensation of the top field. When this flag is set to '0', the top field is used as the reference field. If it is set to '1' then the bottom field will be used as the reference field. This flag is only present in the bitstream if the field_prediction flag is set to "1" and the macroblock is not backward predicted.

**forward_bottom_field_reference**: This is a 1-bit flag which indicates the reference field for the forward motion compensation of the bottom field. When this flag is set to '0', the top field is used as the reference field. If it is set to '1' then the bottom field will be used as the reference field. This flag is only present in the bitstream if the field_prediction flag is set to "1" and the macroblock is not backward predicted.

**backward_top_field_reference**: This is a 1-bit flag which indicates the reference field for the backward motion compensation of the top field. When this flag is set to '0', the top field is used as the reference field. If it is set to '1' then the bottom field will be used as the reference field. This flag is only present in the bitstream if the field_prediction flag is set to "1" and the macroblock is not forward predicted.

© ISO/IEC 2004 – All rights reserved

**backward_bottom_field_reference**: This is a 1-bit flag which indicates the reference field for the backward motion compensation of the bottom field. When this flag is set to '0', the top field is used as the reference field. If it is set to '1' then the bottom field will be used as the reference field. This flag is only present in the bitstream if the field_prediction flag is set to "1" and the macroblock is not forward predicted.

### 6.3.7 Block related

**intra_dc_coefficient**: This is a fixed length code that defines the value of an intra DC coefficient when the short video header format is in use (i.e., when short_video_header is "1"). It is transmitted as a fixed length unsigned integer code of size 8 bits, unless this integer has the value 255. The values 0 and 128 shall not be used – they are reserved. If the integer value is 255, this is interpreted as a signalled value of 128. The integer value is then multiplied by a dc_scaler value of 8 to produce the reconstructed intra DC coefficient value.

**dct_dc_size_luminance**: This is a variable length code as defined in Table B.13 that is used to derive the value of the differential DC coefficients of luminance values in blocks in intra macroblocks. This value categorizes the coefficients according to their size.

**dct_dc_differential**: This is a variable length code as defined in Table B.15 that is used to derive the value of the differential DC coefficients in blocks in intra macroblocks. After identifying the category of the DC coefficient in size from dct_dc_size_luminance or dct_dc_size_chrominance, this value denotes which actual difference in that category occurred.

**dct_dc_size_chrominance**: This is a variable length code as defined in Table B.14 that is used to derive the value of the differential DC coefficients of chrominance values in blocks in intra macroblocks. This value categorizes the coefficients according to their size.

pattern_code[i]: The value of this internal flag is set to 1 if the block or alpha block with the index value i includes one or more DCT coefficients that are decoded using at least one of Table B.16 to Table B.25. Otherwise the value of this flag is set to 0.

#### 6.3.7.1 Alpha block related

**dct_dc_size_alpha**: This is a variable length code for coding the alpha block DC coefficient. Its semantics are the same as dct_dc_size_luminance in subclause 6.3.7.

### 6.3.8 Still texture object

**still_texture_object_start_code**: The still_texture_object_start_code is a string of 32 bits. The first 24 bits are '0000 0000 0000 0000 0000 0001' and the last 8 bits are defined in Table 6-3.

**texture_tile_start_code**: The texture_tile_start_code is a string of 32 bits. The 32 bits are '0000 0000 0000 0000 0000 0001 1100 0001' in binary. The texture_tile_start code marks the start of a new tile.

**tiling_disable**: This field indicates the succeeding bitstream contains a structure of tile when the field is '0'.

**tile_width**: This is a 15-bit unsigned integer which specifies horizontal size, in pexel unit, of the rectangle. When texture_object_layer_shape=='00', this value must be lower than texture_object_layer_width and a zero value is forbidden. When texture_object_layer_width is not a multilple of tile_width, the horizontal size of tile in the last column is derived by texture_object_layer_width%tile_width. When texture_object_layer_shape=='01', object_width is used instead of texture_object_layer_width. The value of tile_width shall be divisible by two.

**tile_height**: This is a 15-bit unsigned integer which specifies vertical size, in pexel unit, of the rectangle. When texture_object_layer_shape=='00', this value must be lower than texture_object_layer_height and a zero value is forbidden. When texture_object_layer_height is not a multilple of tile_height, the vertical size of tile in the last row is derived by texture_object_layer_height%tile_height. When texture_object_layer_shape=='01', object_height is used instead of texture_object_layer_height. The value of tile_height shall be divisible by two.

**number_of_tiles**: This is a 16-bit of unsigned integer specifying the number of tiles encoded in this bitstream. When texture_object_layer_shape=='00', the value is derived from CEIL(texture_object_layer_width÷tile_width) *


© ISO/IEC 2004 – All rights reserved

CEIL(texture_object_layer_height÷tile_height), where CEIL() rounds up to the nearest integer. When texture_object_layer_shape=='01', the value is derived from CEIL(object_width÷tile_width) * CEIL(object_height÷tile_height).

**tiling_jump_table_enable**: This field indicates the succeeding bitstream contains a size of bitstream for each tile when the field is '1'.

**tile_size_high**: This is a left part of 16-bit of a unsigned integer in 32-bit which indicates a size of bitstream containing the corresponding tile in byte unit.

**tile_size_low**: This is a right part of 16-bit of a unsigned integer in 32-bit which indicates a size of bitstream containing the corresponding tile in byte unit. The real size of bitstream for a certain tile is derived from 'tile_size_high<<16+tile_size_low'.

**tile_id**: This is given by 16-bits representing one of the values in the range of '0000' to 'FFFF' in hexadecimal starting from top-left ended to bottom-right. The field uniquely identifies each tile.

**texture_error_resilience_disable**: This is a one-bit flag which when set to '0' indicates that the Still Texture Object is operating in error resilience mode.

**target_segment_length**: This parameter specifies the minimum number of bits in a segment within a packet before adding a segment marker.

**decode_segment_marker()**: This function will decode the arithmetically encoded segment marker. The arithmetic model used is the initial model used in decoding type information for the color in which the segment marker is. When in error free case, a ZTR symbol will be decoded as the segment marker.

**texture_object_id**: This is given by 16-bits representing one of the values in the range of '0000 0000 0000 0000' to '1111 1111 1111 1111' in binary. The texture_object_id uniquely identifies a texture object layer.

**wavelet_filter_type**: This field indicates the arithmetic precision which is used for the wavelet decomposition as the following:

**Table 6-36 — Wavelet type**

| wavelet_filter_type | Meaning |
|---|---|
| 0 | integer |
| 1 | Double float |

**wavelet_download**: This field indicates if the 2-band filter bank is specificed in the bitstream:

**Table 6-37 — Wavelet downloading flag**

| wavelet_download | meaning |
|---|---|
| 0 | default filters |
| 1 | specified in bitstream |

The default filter banks are described in subclause B.2.2.

**wavelet_decomposition_levels**: This field indicates the number of levels in the wavelet decomposition of the texture.

**scan_direction**: This field indicates the scan order of AC coefficients. In single-quant and multi-quant mode, if this flag is `0', then the coefficients are scanned in the tree-depth fashion. If it is `1', then they are scanned in the subband by subband fashion. In bilevel_quant mode, if the flag is `0', then they are scanned in bitplane by bitplane

© ISO/IEC 2004 – All rights reserved

fashion. Within each bitplane, they are scanned in a subband by subband fashion. If it is "1", they are scanned from the low wavelet decomposition layer to high wavelet decomposition layer. Within each wavelet decomposition layer, they are scanned from most significant bitplane down to the least significant bitplane.

**start_code_enable**: If this flag is enabled ( disable =0; enabled = 1), the start code followed by an ID to be inserted in to each spatial scalability layer and/or each SNR scalability layer.

**texture_object_layer_shape**: This is a 2-bit integer defined in Table 6-38. It identifies the shape type of a texture object layer.

**Table 6-38 — Texture Object Layer Shape type**

| texture_object_layer_shape | Meaning |
|---|---|
| 00 | rectangular |
| 01 | binary |
| 10 | reserved |
| 11 | grayscale |

**quantisation_type**: This field indicates the type of quantisation as shown in Table 6-39.

**Table 6-39 — The quantisation type**

| quantisation_type | Code |
|---|---|
| single quantiser | 01 |
| multi quantiser | 10 |
| bi-level quantiser | 11 |

**spatial_scalability_levels**: This field indicates the number of spatial scalability layers supported in the bitstream. This number can be from 1 to wavelet_decomposition_levels.

**use_default_spatial_scalability**: This field indicates how the spatial scalability levels are formed. If its value is one, then default spatial scalability is used, starting from (¼)^(spatial_scalability_levels-1)-th of the full resolution up to the full resolution, where ^ is a power operation. If its value is zero, the spatial scalability is specified by wavelet_layer_index described below.

**wavelet_layer_index**: This field indicates the identification number of wavelet_decomposition layer used for spatial scalability. The index starts with 0 (i.e., root_band) and ends at (wavelet_decomposition_levels–1) (i.e., full resolution).

**uniform_wavelet_filter**: If this field is "1", then the same wavelet filter is applied for all wavelet layers. If this field is "0", then different wavelet filters may be applied for the wavelet decomposition. Note that the same filters are used for both luminance and chromanence. Since the chromanence's width and height is half that of the luminance, the last wavelet filter applied to the luminance is skipped when the chromanence is synthesized.

**wavelet_stuffing**: These 3 stuffing bits are reserved for future expansion. It is currently defined to be '111'.

**texture_object_layer_width**: The texture_object_layer_width is a 15-bit unsigned integer representing the width of the displayable part of the luminance component in pixel units. A zero value is forbidden.

**texture_object_layer_height**: The texture_object_layer_height is a 15-bit unsigned integer representing the height of the displayable part of the luminance component in pixel units. A zero value is forbidden.


© ISO/IEC 2004 – All rights reserved

**horizontal_ref**: This is a 15-bit integer which specifies, in pixel units, the horizontal position of the top left of the rectangle defined by horizontal size of object_width. The value of horizontal_ref shall be divisible by two. This is used for decoding and for picture composition.

**vertical_ref**: This is a 15-bit integer which specifies, in pixel units, the vertical position of the top left of the rectangle defined by vertical size of object_height. The value of vertical_ref shall be divisible by two. This is used for decoding and for picture composition.

**object_width**: This is a 15-bit unsigned integer which specifies the horizontal size, in pixel units, of the rectangle that includes the object. A zero value is forbidden.

**object_height**: This is a 15-bit unsigned integer which specifies the vertical size, in pixel units, of the rectangle that includes the object. A zero value is forbidden.

**quant_byte**: This field defines one byte of the quantisation step size for each scalability layer. A zero value is forbidden. The quantisation step size parameter, quant, is decoded using the function get_param( ): quant = get_param( 7 ).

**max_bitplanes**: This field indicates the number of maximum bitplanes in all three quantisation modes.

**texture_tile_type**: This is a 2-bit integer defined in Table 6-40. It identifies the shape type of a texture object in a tile when texture_object_layer_shape is "01".

**Table 6-40 — Texture Tile Shape**

| texture_tile_type | Meaning |
|---|---|
| 00 | Forbiden |
| 01 | opaque tile |
| 10 | Boundary tile |
| 11 | Transparent tile |

next_texture_marker ():This function performs a similar operation as next_start_code(), but for texture_marker.

**texture_marker** – This is a binary string of at least 16 zero's followed by a one '0 0000 0000 0000 0001'. It is only present when texture_error_resilience_disable flag is set to '0'. A texture marker shall only be located immediately before a texture packet and aligned with a byte.

**TU_first** – This parameter specifies the number of the first texture unit within the texture packet. This parameter is decoded by the function get_param().

**TU_last** – This parameter specifies the number of the last texture unit within the texture packet. This parameter is decoded by the function get_param().

**header_extention_code** – This is a one-bit flag which when set to '1' indicates that additional header information is sent in the texture packet. This bit must have value 1 in the first packet of a texture object.

**target_segment_length** – This parameter specifies the minimum number of bits in a segment within a packet before adding a segment marker.

#### 6.3.8.1 Texture Layer Decoding

**arith_decode_highbands_td()**: This is an arithmetic decoder for decoding the quantised coefficient values of the higher bands (all bands except DC band) within a single tree block. The bitstream is generated by an adaptive arithmetic encoder. The arithmetic decoding relies on the initialization of the uniform probability distribution models described in subclause B.2.2. This decoder uses only integer arithmetic. It also uses an adaptive probability model based on the frequency counts of the previously decoded symbols. The maximum range (or precision) specified is

© ISO/IEC 2004 – All rights reserved

(2^16) - 1 (16 bits). The maximum frequency count for the magnitude and residual models is 127, and for all other models it is 127. The arithmetic coder used is identical to the one used in arith_decode_highbands_bilevel_td().

**texture_spatial_layer_start_code**: The texture_spatial_layer_start_code is a string of 32 bits. The 32 bits are '0000 0000 0000 0000 0000 0001 1011 1111' in binary. The texture_spatial_layer_start_code marks the start of a new spatial layer.

**texture_spatial_layer_id**: This is given by 5-bits representing one of the values in the range of '00000' to '11111' in binary. The texture_spatial_layer_id uniquely identifies a spatial layer.

**arith_decode_highbands_bb()**: This is an arithmetic decoder for decoding the quantised coefficient values of the higher bands (all bands except DC band) within a single band. The bitstream is generated by an adaptive arithmetic encoder. The arithmetic decoding relies on the initialization of the uniform probability distribution models described in subclause B.2.2. This decoder uses arithmetic. It also uses an adaptive probability model based on the frequency counts of the previously decoded symbols. The maximum range (or precision) specified is (2^16) - 1 (16 bits). The maximum frequency count for the magnitude and residual models is 127, and for all other models it is 127.

**snr_scalability_levels**: This field indicates the number of levels of SNR scalability supported in this spatial scalability level.

**texture_snr_layer_start_code**: The texture_snr_layer_start_code is a string of 32 bits. The 32 bits are '0000 0000 0000 0000 0000 0001 1100 0000' in binary. The texture_snr_layer_start_code marks the start of a new snr layer.

**texture_snr_layer_id**: This is given by 5-bits representing one of the values in the range of '00000' to '11111' in binary. The texture_snr_layer_id uniquely identifies an SNR layer.

NOTE All the start codes start at the byte boundary. Appropriate number of bits is stuffed before any start code to byte-align the bitstream.

**all_nonzero**: This flag indicates whether some of the subbands of the current layer contain only zero coefficients. The value '0' for this flag indicates that one or more of the subbands contain only zero coefficients. The value '1' for this flag indicates the all the subbands contain some nonzero coefficients.

**all_zero**: This flag indicates whether all the coefficients in the current layer are zero or not. The value '0' for this flag indicates that the layer contains some nonzero coefficients. The value '1' for this flag indicates that the layer only contains zero coefficients, and therefore the layer is skipped.

**lh_zero, hl_zero, hh_zero**: This flag indicates whether the LH/HL/HH subband of the current layer contains only all zero coefficients. The value '1' for this flag indicates that the LH/HL/HH subband contains only zero coefficients, and therefore the subband is skipped. The value '0' for this flag indicates that the LH/HL/HH subband contains some nonzero coefficients

**arith_decode_highbands_bilevel_bb()**: This is an arithmetic decoder for decoding the quantised coefficient values of the higher bands in the bilevel_quant mode (all bands except DC band). The bitstream is generated by an adaptive arithmetic encoder. The arithmetic decoding relies on the initialization of the uniform probability distribution models described. The arith_decode_highbands_bilevel() function uses bitplane scanning, and a different probability model as described in subclause B.2.2. In this mode, The maximum range (or precision) specified is (2^16) - 1 (16 bits). The maximum frequency count is 127. It uses the lh/hl/hh_zero flags to see if any of the LH/HL/HH are all zero thus not decoded. For example if lh_zero=1 and hh_zero=1 only hl_zero is decoded.

**arith_decode_highbands_bilevel_td()**: This is an arithmetic decoder for decoding the quantised coefficient values of the higher bands in the bilevel_quant mode (all bands except DC band). The bitstream is generated by an adaptive arithmetic encoder. The arithmetic decoding relies on the initialization of the uniform probability distribution models described. The arith_decode_highbands_bilevel() function uses bitplane scanning, and a different probability model as described in subclause B.2.2. In this mode, The maximum range (or precision) specified is (2^16) - 1 (16 bits). The maximum frequency count is 127. It uses the lh/hl/ll_zero flags to see if any of the LH/HL/HH are all zero thus not decoded. For example if lh_zero=1 and hh_zero=1 only hl_zero is decoded.


© ISO/IEC 2004 – All rights reserved

**lowpass_filter_length**: This field defines the length of the low pass filter in binary ranging from "0001" (length of 1) to "1111" (length of 15.)

**highpass_filter_length**: This field defines the length of the high pass filter in binary ranging from "0001" (length of 1) to "1111" (length of 15.)

**filter_tap_integer**: This field defines an integer filter coefficient in a 16 bit signed integer. The filter coefficients are decoded from the left most tap to the right most tap order.

**filter_tap_float_high**: This field defines the left 16 bits of a floating filter coefficient which is defined in 32-bit IEEE floating format. The filter coefficients are decoded from the left most tap to the right most tap order.

**filter_tap_float_low**: This field defines the right 16 bits of a floating filter coefficient which is defined in 32-bit IEEE floating format. The filter coefficients are decoded from the left most tap to the right most tap order.

**integer_scale**: This field defines the scaling factor of the integer wavelet, by which the output of each composition level is divided by an integer division operation. A zero value is forbidden.

**mean**: This field indicates the mean value of one color component of the texture.

**quant_dc_byte**: This field indicates the quantisation step size for one color component of the DC subband. A zero value is forbidden. The quantisation step size parameter, quant_dc, is decoded using the function get_param( ): quant_dc = get_param(7); where get_param() function is defined in the description of band_offset_byte.

**band_offset_byte**: This field defines one byte of the absolute value of the parameter band_offset. This parameter is added to each DC band coefficient obtained by arithmetic decoding. The parameter band_offset is decoded using the function get_param( ):

```
band_offset = -get_param(7);
```

where function get_param() is defined as

```
int get_param(int  nbit)
{
int count = 0;
int word =0;
int value = 0;
int module = 1<<(nbit);

    do{
word= get_next_word_from_bitstream( nbit+1);
value += (word  & (module-1) ) << (count * nbit);
count ++;
} while( word>> nbit);
    return value;
}
```

The function get_next_word_from_bitstream( x ) reads the next x bits from the input bitstream.

**band_max_byte**: This field defines one byte of the maximum value of the DC band. The parameter band_max_value is decoded using function get_param(). The number of maximum bitplanes for DC band is derived from CEIL($\log_2$(band_max_value+1))

```
band_max_value =  get_param(7);
```

© ISO/IEC 2004 – All rights reserved

**arith_decode_dc()**: This is an arithmetic decoder for decoding the quantised coefficient values of DC band only. No zerotree symbol is decoded since the VAL is assumed for all DC coefficient values. This bitstream is generated by an adaptive arithmetic encoder. The arithmetic decoding relies on the initialization of a uniform probability distribution model described in subclause B.2.2. The arith_decode_dc() function uses the same arithmetic decoder as described in arith_decode_highbands_td() but it uses different scanning, and a different probability model (*DC*).

#### 6.3.8.2 Shape Object decoding

**change_conv_ratio_disable**: This specifies whether conv_ratio is encoded at the shape object decoding function. If it is set to "1" when disable.

**sto_constant_alpha**: This is a 1-bit flag when set to '1', the opaque alpha values of the binary mask are replaced with the alpha value specified by sto_constant_alpha_value.

**sto_constant_alpha_value**: This is an 8-bit code that gives the alpha value to replace the opaque pixels in the binary alpha mask. Value '0' is forbidden.

**marker_bit**: This is one-bit that shall be set to 1. This bit prevents emulation of start codes.

**sto_shape_coded_layers**: This is a 4-bit unsigned integer to indicate the number of enhancement layers to contained in the bitstream.

**texture_shape_layer_start_code**: This is a string of 32 bits. The first 24 bits are '0000 0000 0000 0000 0000 0001' and the last 8 bits '1100 0010' (0xC2). This texture_shape_layer_start_code marks the start of a new shape enhancement layer.

**texture_shape_layer_id**: This is given by a 5-bit number representing one of the values in the range of '00000' to '11111' in binary. The texture_shape_layer_id uniquely identifies a shape spatial layer.

**texture_spatial_layer_start_code**: This is a string of 32 bits. The first 24 bits are '0000 0000 0000 0000 0000 0001' and the last 8 bits '1011 1111' (0xBF). This texture_spatial_layer_start_code marks the start of texture decoding.

**texture_spatial_layer_id**: This is given by a 5-bit number representing one of the values in the range of '00000' to '11111' in binary. This texture_spatial_layer_id uniquely identifies the start of texture decoding.

shape_base_layer_height_blocks(): This is a function that returns the number of shape blocks in vertical directions in the base layer. In the case that tiling_disable is '1' object_height will be used for the number. The number is given by ((object_height>>wavelet_decomposition_levels)+15)/16.

If tiling_disable is '0' tiling_height will be used instead of object_height. The number is given by ((tile_height>>wavelet_decomposition_levels) +15)/16.

shape_base_layer_width_blocks(): This is a function that returns the number of shape blocks in horizontal directions in the base layer. In the case that tiling_disable is '1' object_width will be used for the number. The number is given by ((object_width>>wavelet_decomposition_levels) +15)/16.

If tiling_disable is '0' tiling_width will be used instead of object_width. The number is given by ((tile_width>>wavelet_decomposition_levels) +15)/16.

**bab_type**: This is a variable length code of 1-2 bits. It indicates the coding mode used for the bab. There are three bab_types as depicted in Table 6-41. The VLC tables used depend on the decoding context i.e. the bab_types of blocks already received.


© ISO/IEC 2004 – All rights reserved

**Table 6-41 — List of bab_types and usage**

| bab_type | Semantic | code |
|---|---|---|
| 2 | transparent | 10 |
| 3 | opaque | 0 |
| 4 | intraCAE | 11 |

The bab_type determines what other information fields will be present for the bab shape. No further shape information is present if the bab_type = 2 or 3. Opaque means that all pixels of the bab are part of the object. Transparent means that none of the bab pixels belong to the object. IntraCAE means the intra-mode CAE decoding will be required to reconstruct the pixels of the bab.

**conv_ratio**: This is VLC code of length 1-2 bits. It specifies the factor used for sub-sampling the 16x16 pixel bab. The decoder must up-sample the decoded bab by this factor. The possible values for this factor are 1, 2 and 4 and the VLC table used is given in Table B.31.

**scan_type**: This is a 1-bit flag where a value of '0' implies that the bab is in transposed form i.e. the bab has been transposed prior to coding. The decoder must then transpose the bab back to its original form following decoding. If this flag is '1', then no transposition is performed.

binary_arithmetic_decode(): This is a binary arithmetic decoder representing the pixel values of the bab. Cae decoding relies on the knowledge of intra_prob[], probability tables given in Annex B.

shape_enhanced_layer_height_blocks(): This is a function that returns the number of shape blocks in vertical directions in the enhancement layer. In the case that tiling_disable is '1', object_height will be used for the number. If the current coding layer is the *L*-th layer in the bitstream, the number of blocks is given by

((object_height>>(wavelet_decomposition_levels–*L*–1))+bab_size–1)/bab_size.

If tiling_disable is '0' tiling_height will be used instead of object_height. If the current coding layer is the *L*-th layer in the bitstream, the number of blocks is given by

((tile_height>>(wavelet_decomposition_levels–*L*–1))+bab_size–1)/bab_size.

The value of bab_size (size of the coded bab) is defined in subclause 7.10.6.2.1.

shape_enhanced_layer_width_blocks(): This is a function that returns the number of shape blocks in horizontal directions in the enhancement layer. In the case that tiling_disable is '1', object_width will be used for the number. If the current coding layer is the *L*-th layer in the bitstream, the number of blocks is given by

((object_width>>(wavelet_decomposition_levels–*L*–1))+bab_size–1)/bab_size.

If tiling_disable is '0' tiling_width will be used instead of object_width. If the current coding layer is the *L*-th layer in the bitstream, the number of blocks is given by

((tile_width>>(wavelet_decomposition_levels–*L*–1))+bab_size–1)/bab_size.

enh_binary_arithmetic_decode(): The first decoded value denotes BAB type of Scan Interleaving (SI) method (SI_bab_type: "0": transitional BAB, "1": exceptional BAB). And the other decoded values represent the pixel values of the current BAB. If the BAB is a transitional BAB, only transitional pixels are decoded. Otherwise all of the pixels are decoded. This binary value decoding relies on the knowledge of SI_bab_type_prob[], enh_intra_v_prob[] and enh_intra_h_prob[], sto_SI_bab_type_prob_even[], sto_enh_odd_prob0[], sto_enh_even_prob0[], sto_enh_odd_prob1[], and sto_enh_even_prob1[] probability tables given in Annex B.

© ISO/IEC 2004 – All rights reserved

### 6.3.9 Mesh object

**mesh_object_start_code**: The mesh_object_start_code is the bit string '000001BC' in hexadecimal. It initiates a mesh object.

#### 6.3.9.1 Mesh object plane

**mesh_object_plane_start_code**: The mesh_object_plane_start_code is the bit string '000001BD' in hexadecimal. It initiates a mesh object plane.

**is_intra**: This is a 1-bit flag which when set to '1' indicates that the mesh object is coded in intra mode. When set to '0' it indicates that the mesh object is coded in predictive mode.

#### 6.3.9.2 Mesh geometry

**mesh_type_code**: This is a 2-bit integer defined in Table 6-42. It indicates the type of initial mesh geometry to be decoded.

**Table 6-42 — Mesh type code**

| mesh type code | mesh geometry |
|---|---|
| 00 | forbidden |
| 01 | uniform |
| 10 | Delaunay |
| 11 | reserved |

**nr_of_mesh_nodes_hor**: This is a 10-bit unsigned integer specifying the number of nodes in one row of a uniform mesh.

**nr_of_mesh_nodes_vert**: This is a 10-bit unsigned integer specifying the number of nodes in one column of a uniform mesh.

**mesh_rect_size_hor**: This is a 8-bit unsigned integer specifying the width of a rectangle of a uniform mesh (containing two triangles) in half pixel units.

**mesh_rect_size_vert**: This is a 8-bit unsigned integer specifying the height of a rectangle of a uniform mesh (containing two triangles) in half pixel units.

**triangle_split_code**: This is a 2-bit integer defined in Table 6-43. It specifies how rectangles of a uniform mesh are split to form triangles.

**Table 6-43 — Specification of the triangulation type**

| triangle split code | Split |
|---|---|
| 00 | top-left to right bottom |
| 01 | bottom-left to top right |
| 10 | alternately top-left to bottom-right and bottom-left to top-right |
| 11 | alternately bottom-left to top-right and top-left to bottom-right |

**nr_of_mesh_nodes**: This is a 16-bit unsigned integer defining the total number of nodes (vertices) of a (non-uniform) Delaunay mesh. These nodes include both interior nodes as well as boundary nodes.


© ISO/IEC 2004 – All rights reserved

**nr_of_boundary_nodes**: This is a 10-bit unsigned integer defining the number of nodes (vertices) on the boundary of a (non-uniform) Delaunay mesh.

**node0_x**: This is a 13-bit signed integer specifying the x-coordinate of the first boundary node (vertex) of a mesh in half-pixel units with respect to a local coordinate system.

**node0_y**: This is a 13-bit signed integer specifying the y-coordinate of the first boundary node (vertex) of a mesh in half-pixel units with respect to a local coordinate system.

**delta_x_len_vlc**: This is a variable-length code specifying the length of the delta_x code that follows. The delta_x_len_vlc and delta_x codes together specify the difference between the x-coordinates of a node (vertex) and the previously encoded node (vertex). The definition of the delta_x_len_vlc and delta_x codes are given in Table B.33, the table for sprite motion trajectory coding.

**delta_x**: This is an integer that defines the value of the difference between the x-coordinates of a node (vertex) and the previously encoded node (vertex) in half pixel units. The number of bits in the bitstream for delta_x is delta_x_len_vlc.

**delta_y_len_vlc**: This is a variable-length code specifying the length of the delta_y code that follows. The delta_y_len_vlc and delta_y codes together specify the difference between the y-coordinates of a node (vertex) and the previously encoded node (vertex). The definition of the delta_y_len_vlc and delta_y codes are given in Table B.33, the table for sprite motion trajectory coding.

**delta_y**: This is an integer that defines the value of the difference between the y-coordinates of a node (vertex) and the previously encoded node (vertex) in half pixel units. The number of bits in the bitstream for delta_y is delta_y_len_vlc.

#### 6.3.9.3 Mesh motion

**motion_range_code**: This is a 3-bit integer defined in Table 6-44. It specifies the dynamic range of motion vectors in half pel units.

**Table 6-44 — motion range code**

| motion range code | motion vector range |
|---|---|
| 1 | [-32, 31] |
| 2 | [-64, 63] |
| 3 | [-128, 127] |
| 4 | [-256, 255] |
| 5 | [-512, 511] |
| 6 | [-1024, 1023] |
| 7 | [-2048, 2047] |

**node_motion_vector_flag**: This is a 1 bit code specifying whether a node has a zero motion vector. When set to '1' it indicates that a node has a zero motion vector, in which case the motion vector is not encoded. When set to '0', it indicates the node has a nonzero motion vector and that motion vector data shall follow.

**delta_mv_x_vlc**: This is a variable-length code defining (together with delta_mv_x_res) the value of the difference in the x-component of the motion vector of a node compared to the x-component of a predicting motion vector. The definition of the delta_mv_x_vlc codes are given in Table B.12, the table for motion vector coding (MVD). The value delta_mv_x_vlc is given in half pixel units.

**delta_mv_x_res**: This is an integer which is used in mesh node motion vector decoding using an algorithm equivalent to that described in the section on video motion vector decoding, subclause 7.6.3. The number of bits in the bitstream for delta_mv_x_res is motion_range_code-1.

© ISO/IEC 2004 – All rights reserved

**delta_mv_y_vlc**: This is a variable-length code defining (together with delta_mv_y_res) the value of the difference in the y-component of the motion vector of a node compared to the y-component of a predicting motion vector. The definition of the delta_mv_y_vlc codes are given in Table B.12, the table for motion vector coding (MVD). The value delta_mv_y_vlc is given in half pixel units.

**delta_mv_y_res**: This is an integer which is used in mesh node motion vector decoding using an algorithm equivalent to that described in the section on video motion vector decoding, subclause 7.6.3. The number of bits in the bitstream for delta_mv_y_res is motion_range_code-1.

### 6.3.10 FBA object

**fba_object_start_code**: The fba_object_start_code is the bit string '000001BA' in hexadecimal. It initiates a FBA object.

#### 6.3.10.1 FBA object plane header

**fba_object_plane_start_code**: The fba_frame_start_code is the bit string '000001BB' in hexadecimal. It initiates a FBA object plane.

**is_intra**: This is a 1-bit flag which when set to '1' indicates that the FBA object is coded in intra mode. When set to '0' it indicates that the FBA object is coded in predictive mode.

**fba_object_mask**: This is a 2-bit integer defined in Table 6-45. It indicates whether FBA and BAP data are present in the FBA_frame.

**Table 6-45 — FBA object mask**

| mask value | Meaning |
|---|---|
| 00 | unused |
| 01 | FAP present |
| 10 | BAP present |
| 11 | both FAP and BAP present |

#### 6.3.10.2 FBA object plane data

**fap_quant**: This is a 5-bit unsigned integer which is the quantiser scale used to compute the FAPi table step size or DCT fap_scale depending on the fba_object_coding_type. If the fba_object_coding_type is DCT this is a 5-bit unsigned integer used as the index to a fap_scale table for computing the quantisation step size of DCT coefficients. The value of fap_scale is specified in the following list:

fap_scale[0 - 31] = { 1, 1, 2, 3, 5, 7, 8, 10, 12, 15, 18, 21, 25, 30, 35, 42,
50, 60, 72, 87, 105, 128, 156, 191, 234, 288, 355, 439, 543, 674, 836, 1039}

**fap_mask_type**: This is a 2-bit integer. It indicates if the group mask will be present for the specified fap group, or if the complete faps will be present; its meaning is described in Table 6-46. In the case the type is '10' the '0' bit in the group mask indicates interpolate fap.


© ISO/IEC 2004 – All rights reserved

**Table 6-46 — fap mask type**

| mask type | Meaning |
|---|---|
| 00 | no mask nor fap |
| 01 | group mask for copying |
| 10 | group mask for interpolation |
| 11 | fap |

**fap_group_mask**[group_number]: This is a variable length bit entity that indicates, for a particular group_number which fap is represented in the bitstream. The value is interpreted as a mask of 1-bit fields. A 1-bit field in the mask that is set to '1' indicates that the corresponding fap is present in the bitstream. When that 1-bit field is set to '0' it indicates that the fap is not present in the bitstream. The number of bits used for the fap_group_mask depends on the group_number, and is given in Table 6-47.

**Table 6-47 — fap group mask bits**

| group_number | No. of bits |
|---|---|
| 1 | 2 |
| 2 | 16 |
| 3 | 12 |
| 4 | 8 |
| 5 | 4 |
| 6 | 5 |
| 7 | 3 |
| 8 | 10 |
| 9 | 4 |
| 10 | 4 |

NFAP[group_number] : This indicates the number of FAPs in each FAP group. Its values are specified in the following table:

**Table 6-48 — NFAP definition**

| group_number | NFAP[group_number] |
|---|---|
| 1 | 2 |
| 2 | 16 |
| 3 | 12 |
| 4 | 8 |
| 5 | 4 |
| 6 | 5 |
| 7 | 3 |
| 8 | 10 |
| 9 | 4 |
| 10 | 4 |

**fba_suggested_gender**: This is a 1-bit integer indicating the suggested gender for the face model. It does not bind the decoder to display a facial model of suggested gender, but indicates that the content would be more suitable for display with the facial model of indicated gender, if the decoder can provide one. If fba_suggested_gender is 1, the suggested gender is male, otherwise it is female.

© ISO/IEC 2004 – All rights reserved

**fba_object_coding_type**: This is a 1-bit integer indicating which coding method is used. Its meaning is described in Table 6-49.

**Table 6-49 — fba_object_coding_type**

| type value | Meaning |
|---|---|
| 0 | predictive coding |
| 1 | DCT |

**is_i_new_max**: This is a 1-bit flag which when set to '1' indicates that a new set of maximum range values for I frame follows these 4, 1-bit fields.

**is_i_new_min**: This is a 1-bit flag which when set to '1' indicates that a new set of minimum range values for I frame follows these 4, 1-bit fields.

**is_p_new_max**: This is a 1-bit flag which when set to '1' indicates that a new set of maximum range values for P frame follows these 4, 1-bit fields.

**is_p_new_min**: This is a 1-bit flag which when set to '1' indicates that a new set of minimum range values for P frame follows these 4, 1-bit fields.

**bap_pred_quant_index**: This is a 5-bit unsigned integer used as the index to a bap_pred_scale table for computing the quantisation step size of BAP values for predictive and DCT coding. If fba_object_coding_type is predictive, the value of bap_pred_scale is specified in the following list:

bap_pred_scale[0-31]=
{ 0, 1, 2, 3, 5, 7, 9, 11, 14, 17, 20, 23, 27, 31, 35, 39, 43, 47, 52, 57, 62, 67, 72, 77, 82, 88, 94, 100, 106, 113, 120, 127}

If the fba_object_coding_type is DCT this is a 5-bit unsigned integer used as the index to a bap_scale table for computing the quantisation step size of DCT coefficients. The value of bap_scale is specified in the following list:

bap_scale[0 – 31] = { 1, 1, 2, 3, 5, 7, 8, 10, 12, 15, 18, 21, 25, 30, 35, 42,
50, 60, 72, 87, 105, 128, 156, 191, 234, 288, 355, 439, 543, 674, 836, 1039}

**bap_mask_type**: This 2-bit value determines whether BAPs are transmitted individually or in groups.

| bap_mask_type | Meaning |
|---|---|
| 00 | No mask nor BAPS |
| 01 | BAP group mask transmitted |
| 10 | reserved |
| 11 | All BAPs transmitted –no group mask |

**bap_group_mask**: this is a variable-length mask indicating which BAPs in a group are present in the fba_object_plane.

| group number | group name | No. of. bits |
|---|---|---|
| 1 | Pelvis | 3 |
| 2 | Left leg1 | 4 |
| 3 | Right leg1 | 4 |
| 4 | Left leg2 | 6 |


© ISO/IEC 2004 – All rights reserved

| | | |
|---|---|---|
| 5 | Right leg2 | 6 |
| 6 | Left arm1 | 5 |
| 7 | Right arm1 | 5 |
| 8 | Left arm2 | 7 |
| 9 | Right arm2 | 7 |
| 10 | Spine1 | 12 |
| 11 | Spine2 | 15 |
| 12 | Spine3 | 18 |
| 13 | Spine4 | 18 |
| 14 | Spine5 | 12 |
| 15 | Left hand1 | 16 |
| 16 | Right hand1 | 16 |
| 17 | Left hand2 | 13 |
| 18 | Right hand2 | 13 |
| 19 | Global positioning | 6 |
| 20 | Extension1 | 22 |
| 21 | Extension2 | 22 |
| 22 | Extension3 | 22 |
| 23 | Extension4 | 22 |
| 24 | Extension5 | 22 |

**bap_is_i_new_max**: This is a 1-bit flag which when set to '1' indicates that a new set of maximum range values for I frame follows these 4, 1-bit fields.

**bap_is_i_new_min**: This is a 1-bit flag which when set to '1' indicates that a new set of minimum range values for I frame follows these 4, 1-bit fields.

**bap_is_p_new_max**: This is a 1-bit flag which when set to '1' indicates that a new set of maximum range values for P frame follows these 4, 1-bit fields.

**bap_is_p_new_min**: This is a 1-bit flag which when set to '1' indicates that a new set of minimum range values for P frame follows these 4, 1-bit fields.

#### 6.3.10.3 Temporal Header

**is_frame_rate**: This is a 1-bit flag which when set to '1' indicates that frame rate information follows this bit field. When set to '0' no frame rate information follows this bit field.

**is_time_code**: This is a 1-bit flag which when set to '1' indicates that time code information follows this bit field. When set to '0' no time code information follows this bit field.

**time_code**: This is a 18-bit integer containing the following: time_code_hours, time_code_minutes, marker_bit and time_code_seconds as shown in Table 6-23. The parameters represent the hours, minutes, and seconds of the time base for the current object plane. The time code specifies the modulo part (i.e. the full second units) of the time base for the first object plane (in display order) after the GOV header. The time_code values in MPEG-4 are meant to be taken literally as precise multiples of exact seconds. It is not the same as SMPTE 12M time code.

© ISO/IEC 2004 – All rights reserved

**Table 6-50 — Meaning of time_code**

| time_code | range of value | No. of bits | Mnemonic |
|---|---|---|---|
| time_code_hours | 0 - 23 | 5 | uimsbf |
| time_code_minutes | 0 - 59 | 6 | uimsbf |
| marker_bit | 1 | 1 | bslbf |
| time_code_seconds | 0 - 59 | 6 | uimsbf |

**skip_frames**: This is a 1-bit flag which when set to '1' indicates that information follows this bit field that indicates the number of skipped frames. When set to '0' no such information follows this bit field.

#### 6.3.10.4 Decode frame rate and frame skip

**frame_rate**: This is an 8 bit unsigned integer indicating the reference frame rate of the sequence.

**seconds**: This is a 4 bit unsigned integer indicating the fractional reference frame rate. The frame rate is computed as follows frame rate = (frame_rate + seconds/16).

**frequency_offset**: This is a 1-bit flag which when set to '1' indicates that the frame rate uses the NTSC frequency offset of 1000/1001. This bit would typically be set when frame_rate = 24, 30 or 60, in which case the resulting frame rate would be 23.97, 29.94 or 59.97 respectively. When set to '0' no frequency offset is present. I.e. if (frequency_offset ==1) frame rate = (1000/1001) * (frame_rate + seconds/16).

**number_of_frames_to_skip**: This is a 4-bit unsigned integer indicating the number of frames skipped. If the number_of_frames_to skip is equal to 15 (pattern "1111") then another 4-bit word follows allowing to skip up to 29 frames(pattern "11111110"). If the 8-bits pattern equals "11111111", then another 4-bits word will follow and so on, and the number of frames skipped is incremented by 30. Each 4-bit pattern of '1111' increments the total number of frames to skip with 15.

#### 6.3.10.5 Decode new minmax

**i_new_max[j]**: This is a 5-bit unsigned integer used to scale the maximum value of the arithmetic decoder used in the I frame. Arithmetic decoder max values must be non-negative and min values must be non-positive. Min values are positive in the bitstream and negated in decoder.

**i_new_min[j]**: This is a 5-bit unsigned integer used to scale the minimum value of the arithmetic decoder used in the I frame. Arithmetic decoder max values must be non-negative and min values must be non-positive. Min values are positive in the bitstream and negated in decoder.

**p_new_max[j]**: This is a 5-bit unsigned integer used to scale the maximum value of the arithmetic decoder used in the P frame. Arithmetic decoder max values must be non-negative and min values must be non-positive. Min values are positive in the bitstream and negated in decoder.

**p_new_min[j]**: This is a 5-bit unsigned integer used to scale the minimum value of the arithmetic decoder used in the P frame. Arithmetic decoder max values must be non-negative and min values must be non-positive. Min values are positive in the bitstream and negated in decoder.

#### 6.3.10.6 Decode viseme and expression

**viseme_def**: This is a 1-bit flag which when set to '1' indicates that the mouth FAPs sent with the viseme FAP may be stored in the decoder to help with FAP interpolation in the future.

**init_face**: This is a 1-bit flag which when set to '1' indicates that the neutral face may be modified within the neutral face constraints.


© ISO/IEC 2004 – All rights reserved

**expression_def**: This is a 1-bit flag which when set to '1' indicates that the FAPs sent with the expression FAP may be stored in the decoder to help with FAP interpolation in the future.

#### 6.3.10.7 Decode viseme_segment and expression_segment

**viseme_segment_select1q[k]**: This is the quantised value of viseme_select1 at frame k of a viseme FAP segment.

**viseme_segment_select2q[k]**: This is the quantised value of viseme_select2 at frame k of a viseme FAP segment.

**viseme_segment_blendq[k]**: This is the quantised value of viseme_blend at frame k of a viseme FAP segment.

**viseme_segment_def[k]**: This is a 1-bit flag which when set to '1' indicates that the mouth FAPs sent with the viseme FAP at frame k of a viseme FAP segment may be stored in the decoder to help with FAP interpolation in the future.

**viseme_segment_select1q_diff[k]**: This is the prediction error of viseme_select1 at frame k of a viseme FAP segment.

**viseme_segment_select2q_diff[k]**: This is the prediction error of viseme_select2 at frame k of a viseme FAP segment.

**viseme_segment_blendq_diff[k]**: This is the prediction error of viseme_blend at frame k of a viseme FAP segment.

**expression_segment_select1q[k]**: This is the quantised value of expression_select1 at frame k of an expression FAP segment.

**expression_segment_select2q[k]**: This is the quantised value of expression_select2 at frame k of an expression FAP segment.

**expression_segment_intensity1q[k]**: This is the quantised value of expression_intensity1 at frame k of an expression FAP segment

**expression_segment_intensity2q[k]**: This is the quantised value of expression_intensity2 at frame k of an expression FAP segment

**expression_segment_select1q_diff[k]**: This is the prediction error of expression_select1 at frame k of an expression FAP segment.

**expression_segment_select2q_diff[k]**: This is the prediction error of expression_select2 at frame k of an expression FAP segment.

**expression_segment_intensity1q_diff[k]**: This is the prediction error of expression_intensity1 at frame k of an expression FAP segment.

**expression_segment_intensity2q_diff[k]**: This is the prediction error of expression_intensity2 at frame k of an expression FAP segment.

**expression_segment_init_face[k]**: This is a 1-bit flag which indicates the value of init_face at frame k of an expression FAP segment.

**expression_segment_def[k]**: This is a 1-bit flag which when set to '1' indicates that the FAPs sent with the expression FAP at frame k of a viseme FAP segment may be stored in the decoder to help with FAP interpolation in the future.

© ISO/IEC 2004 – All rights reserved

#### 6.3.10.8 Decode i_dc, p_dc, and ac

**dc_q**: This is the quantised DC component of the DCT coefficients. For an intra FAP segment, this component is coded as a signed integer of either 16 bits or 31 bits. The DCT quantiser scales of the 68 FAPs are specified in the following list:

```
DCTQP[1 - 68] = {1,    1,    7.5,   7.5,   7.5,   7.5,   7.5,   7.5,   7.5,   7.5,
                 7.5,  7.5,  7.5,   15,    15,    15,    15,    5,     10,    10,
                 10,   10,   425,   425,   425,   425,   5,     5,     5,     5,
                 7.5,  7.5,  7.5,   7.5,   7.5,   7.5,   7.5,   7.5,   20,    20,
                 20,   20,   10,    10,    10,    10,    255,   170,   255,   255,
                 7.5,  7.5,  7.5,   7.5,   7.5,   7.5,   7.5,   7.5,   7.5,   7.5,
                 15,   15,   15,    15,    10,    10,    10,    10}
```

For DC coefficients, the quantisation step size is obtained as follows:

qstep[i] = fap_scale[fap_quant_inex] * DCTQP[i] ÷ 3.0

**dc_q_diff**: This is the quantised prediction error of a DC coefficient of an inter FAP segment. Its value is computed by subtracting the decoded DC coefficient of the previous FAP segment from the DC coefficient of the current FAP segment. It is coded by a variable length code if its value is within [-255, +255]. Outside this range, its value is coded by a signed integer of 16 or 32 bits.

**count_of_runs**: This is the run length of zeros preceding a non-zero AC coefficient.

**ac_q[i][next]**: This is a quantised AC coefficients of a segment of FAPi. For AC coefficients, the quantiser step size is three times larger than the DC quantisation step size and is obtained as follows:

qstep[i] = fap_scale[fap_quant_inex] * DCTQP[i]

BAPs are decoded using the same process. For BAPs, DCT quantiserscales (DCTQP[i]) have the same value as BAP predictive coding step sizes as defined in Annex C.

#### 6.3.10.9 Decode bap min max

**bap_i_new_max[i]** – This is a 5-bit unsigned integer used to scale the maximum value of the arithmetic decoder used in the I frame.

**bap_i_new_min[i]** – This is a 5-bit unsigned integer used to scale the minimum value of the arithmetic decoder used in the I frame.

**bap_p_new_max[i]** – This is a 5-bit unsigned integer used to scale the maximum value of the arithmetic decoder used in the P frame.

**bap_p_new_min[i]** – This is a 5-bit unsigned integer used to scale the minimum value of the arithmetic decoder used in the P frame.

### 6.3.11 3D Mesh Object

#### 6.3.11.1 3D_Mesh_Object

**3D_MO_start_code**: This is a unique 16-bit code that is used for synchronization purpose. The value of this code is always '0000 0000 0010 0000'.


© ISO/IEC 2004 – All rights reserved

#### 6.3.11.2 3D_Mesh_Object_Layer

**3D_MOL_start_code**: This is a unique 16-bit code that is used for synchronization purposes. The value of this code is always '0000 0000 0011 0000'.

**mol_id**: This 8-bit unsigned integer specifies a unique id for the mesh object layer. Value 0 indicates a base layer, and value larger than 0 a refinement layer. The first 3D_Mesh_Object_Layer immediately after a 3D_Mesh_Object_Header must have mold_id=0, and subsequent 3DMesh_Object_Layer's within the same 3D_Mesh_Object must have mold_id>0.

**ce_SNHC_n_vertices**: This is the number of vertices in the current resolution of the 3D mesh. Used to support computational graceful degradation.

**ce_SNHC_n_triangles**: This is the number of triangles in the current resolution of the 3D mesh. Used to support computational graceful degradation.

**ce_SNHC_n_edges**: This is the number of edges in the current resolution of the 3D mesh. Used to support computational graceful degradation.

#### 6.3.11.3 3DMesh_Object_Base_Layer

**3D_MOBL_start_code**: This is a code of length 16 that is used for synchronization purposes. It also indicates three different partition types for error resilience.

**Table 6-51 — Definition of partition type information**

| 3D_MOBL_start_code | partition type | Meaning |
|---|---|---|
| '0000 0000 0011 0001' | partition_type_0 | One or more groups of vg, tt and td. |
| '0000 0000 0011 0011' | partition_type_1 | One or more vgs |
| '0000 0000 0011 0100' | partition_type_2 | One pair of tt and td. |

**mobl_id**: This 8-bit unsigned integer specifies a unique id for the mesh object component.

**one_bit**: This boolean value is always true. This value is used for byte alignment.

**last_component**: This boolean value indicates if there are more connected components to be decoded. If **last_component** is '1', then the last component has been decoded. Otherwise there are more components to be decoded. This field is arithmetic coded

**codap_last_vg**: This boolean value indicates if the current vg is the last one in the partition. The value is false if there are more vgs to be decoded in the partition.

**codap_vg_id**: This unsigned integer indicates the id of the vertex graph corresponding to the current simple polygon in partition_type_2. The length of this value is a log scaled value of the vg_number of vg decoded from the previous partition_type_1. If there is only one vg in the previous partition_type_1.

**codap_left_bloop_idx**: This unsigned integer indicates the left starting index, within the bounding loop table of a connected component, for the triangles that are to be reconstructed in a partition. The length of this value is the log scaled value of the size of the bounding loop table.

**codap_right_bloop_idx**: This unsigned integer indicates the right starting index, within the bounding loop table of a connected component, for the triangles that are to be reconstructed in a partition. The length of this value is the log scaled value of the size of the bounding loop table.

**codap_bdry_pred**: This boolean value denotes how to predict geometry and photometry information that are in common with two or more partitions. If **codap_bdry_pred** is '1', the restricted boundary prediction mode is used, otherwise, the extended boundary prediction mode is used.

© ISO/IEC 2004 – All rights reserved

#### 6.3.11.4 3DMesh_Object_Header

**ccw**: This boolean value indicates if the vertex ordering of the decoded faces follows a counter clock-wise order.

**convex**: This boolean value indicates if the model is convex.

**solid**: This boolean value indicates if the model is solid.

**creaseAngle**: This 6-bit unsigned integer indicates the crease angle.

#### 6.3.11.5 coord_header

**coord_binding**: This 2 bit unsigned integer indicates the binding of vertex coordinates to the 3D mesh. Table 6-52 shows the admissible values for **coord_binding**.

**Table 6-52 — Admissible values for coord_binding**

| coord_binding | binding |
|---|---|
| 00 | forbidden |
| 01 | bound_per_vertex |
| 10 | forbidden |
| 11 | forbidden |

**coord_bbox**: This boolean value indicates whether a bounding box is provided for the geometry. If no bounding box is provided, a default bounding box is used. The default bounding box is defined as **coord_xmin**=0, **coord_ymin**=0, **coord_zmin**=0, and **coord_size**=1.

**coord_xmin**, **coord_ymin**, **coord_zmin**: These floating point values indicates the lower left corner of the bounding box in which the geometry lies.

**coord_size**: This floating point value indicates the size of the bounding box.

**coord_quant**: This 5-bit unsigned integer indicates the quantisation step used for geometry. The minimum value of coord_quant is 1 and the maximum is 24.

**coord_pred_type**: This 2-bit unsigned integer indicates the type of prediction used to reconstruct the vertex coordinates of the mesh. Table 6-53 shows the admissible values for coord_pred_type.

**Table 6-53 — Admissible values for coord_pred_type**

| coord_pred_type | prediction type |
|---|---|
| 00 | no_prediction |
| 01 | forbidden |
| 10 | parallelogram_prediction |
| 11 | reserved |

**coord_nlambda**: This 2-bit unsigned integer indicates the number of ancestors used to predict geometry. The only admissible value of **coord_nlambda** is 3. Table 6-54 shows the admissible values as a function of **coord_pred_type.**


© ISO/IEC 2004 – All rights reserved

**Table 6-54 — Admissible values for coord_nlambda as a function of coord_prediction type**

| coord_pred_type | coord_nlambda |
|---|---|
| 00 | not coded |
| 10 | 3 |

**coord_lambda**: This signed fixed-point number indicates the weight given to an ancestor for prediction. The number of bits used for this field is equal to **coord_quant** + 3. The 3 leading bits represent the integer part, and the **coord_quant** remaining bits the fractional part.

#### 6.3.11.6 normal_header

**normal_binding**: This 2 bit unsigned integer indicates the binding of normals to the 3D mesh. The admissible values are described in Table 6-55.

**Table 6-55 — Admissible values for normal_binding**

| normal_binding | binding |
|---|---|
| 00 | not_bound |
| 01 | bound_per_vertex |
| 10 | bound_per_face |
| 11 | bound_per_corner |

**normal_bbox:** This boolean value should always be false ('0').

**normal_quant**: This 5-bit unsigned integer indicates the quantisation step used for normals. The minimum value of normal_quant is 3 and the maximum is 31.

**normal_pred_type**: This 2-bit unsigned integer indicates how normal values are predicted. Table 6-56 shows the admissible values, and Table 6-57 shows admissible values as a function of normal_binding.

**Table 6-56 — Admissible values for normal_pred_type**

| normal_pred_type | prediction type |
|---|---|
| 00 | no_prediction |
| 01 | tree_prediction |
| 10 | parallelogram_prediction |
| 11 | reserved |

**Table 6-57 — Admissible combinations of normal_binding and normal_pred_type**

| normal_binding | normal_pred_type |
|---|---|
| not_bound | not coded |
| bound_per_vertex | no_prediction, parallelogram_prediction |
| bound_per_face | no_prediction, tree_prediction |
| bound_per_corner | no_prediction, tree_prediction |

**normal_nlambda**: This 2-bit unsigned integer indicates the number of ancestors used to predict normals. Admissible values of **normal_nlambda** are 1, 2, and 3. Table 6-58 shows admissible values as a function of **normal_pred_type.**

© ISO/IEC 2004 – All rights reserved

**Table 6-58 — Admissible values for normal_nlambda as a function of normal_prediction type**

| normal_pred_type | normal_nlambda |
|---|---|
| no_prediction | not coded |
| tree_prediction | 1, 2, 3 |
| parallelogram_prediction | 3 |

**normal_lambda**: This signed fixed-point indicates the weight given to an ancestor for prediction. The number of bits used for normal_lambda is (normal_quant-3)/2+3. The 3 leading bits represent the integer part, and the **normal_quant** remaining bits the fractional part.

#### 6.3.11.7 color_header

**color_binding**: This 2 bit unsigned integer indicates the binding of colors to the 3D mesh. Table 6-59 shows the admissible values.

**Table 6-59 — Admissible values for color_binding**

| color_binding | Binding |
|---|---|
| 00 | not_bound |
| 01 | bound_per_vertex |
| 10 | bound_per_face |
| 11 | bound_per_corner |

**color_bbox**: This boolean indicates if a bouding box for colors is given. If no bounding box is provided, a default bounding box is used. The default bounding box is defined as **color_rmin**=0, **color_gmin**=0, **color_bmin**=0, and **color_size**=1.

**color_rmin**, **color_gmin**, **color_bmin**: These floating point values give the position of the lower left corner of the bounding box in RGB space.

**color_size**: This floating point value gives the size of the color bounding box.

**color_quant**: This 5-bit unsigned integer indicates the quantisation step used for colors. The minimum value of color_quant is 1 and the maximum is 16.

**color_pred_type**: This 2-bit unsigned integer indicates how colors are predicted. Table 6-60 shows the admissible values, and Table 6-61 shows admissible values as a function of color_binding.

**Table 6-60 — Admissible values for color_pred_type**

| color_pred_type | prediction type |
|---|---|
| 00 | no_prediction |
| 01 | tree_prediction |
| 10 | parallelogram_prediction |
| 11 | reserved |


© ISO/IEC 2004 – All rights reserved

**Table 6-61 — Admissible combinations of color_binding and color_pred_type**

| color_binding | color_pred_type |
|---|---|
| not_bound | not coded |
| bound_per_vertex | no_prediction, parallelogram_prediction |
| bound_per_face | no_prediction, tree_prediction |
| bound_per_corner | no_prediction, tree_prediction |

**color_nlambda**: This 2-bit unsigned integer indicates the number of ancestors used to predict normals. Admissible values of **color_nlambda** are 1, 2, and 3. Table 6-62 shows admissible values as a function of **normal_pred_type.**

**Table 6-62 — Admissible values for color_nlambda as a function of color_prediction type**

| color_pred_type | color_nlambda |
|---|---|
| no_prediction | not coded |
| tree_prediction | 1, 2, 3 |
| parallelogram_prediction | 3 |

**color_lambda**: This signed fixed-point indicates the weight given to an ancestor for prediction. The number of bits used for this field is equal to **color_quant** + 3. The 3 leading bits represent the integer part, and the **normal_quant** remaining bits the fractional part.

#### 6.3.11.8 texCoord_header

**texCoord_binding**: This 2 bit unsigned integer indicates the binding of texture coordinates to the 3D mesh. Table 6-63 describes the admissible values.

**Table 6-63 — Admissible values for texCoord_binding**

| texCoord_binding | Binding |
|---|---|
| 00 | not_bound |
| 01 | bound_per_vertex |
| 10 | forbidden |
| 11 | bound_per_corner |

**texCoord_bbox**: This boolean value indicates if a bounding box for texture coordinates is given. If no bounding box is provided, a default bounding box is used. The default bounding box is defined as **texCoord_umin**=0, **texCoord_vmin**=0, and **texCoord_size**=1.

**texCoord_umin**, **texCoord_vmin**: These floating point values give the position of the lower left corner of the bounding box in 2D space.

**texCoord_size**: This floating point value gives the size of the texture coordinate bounding box.

**texCoord_quant**: This 5-bit unsigned integer indicates the quantisation step used for texture coordinates. The minimum value of texCoord_quant is 1 and the maximum is 16.

**texCoord_pred_type**: This 2-bit unsigned integer indicates how colors are predicted. Table 6-64 shows the admissible values, and Table 6-65 shows admissible values as a function of texCoord_binding.

© ISO/IEC 2004 – All rights reserved

**Table 6-64 — Admissible values for texCoord_pred_type**

| texCoord_pred_type | prediction type |
|---|---|
| 00 | no_prediction |
| 01 | forbidden |
| 10 | parallelogram_prediction |
| 11 | reserved |

**Table 6-65 — Admissible combinations of texCoord_binding and texCoord_pred_type**

| texCoord_binding | texCoord_pred_type |
|---|---|
| not_bound | not coded |
| bound_per_vertex | no_prediction, parallelogram_prediction |
| bound_per_corner | no_prediction, tree_prediction |

**texCoord_nlambda**: This 2-bit unsigned integer indicates the number of ancestors used to predict normals. Admissible values of **texCoord_nlambda** are 1, 2, and 3. Table 6-66 shows admissible values as a function of **texCoord_pred_type.**

**Table 6-66 — Admissible values for texCoord_nlambda as a function of texCoord_prediction type**

| texCoord_pred_type | texCoord_nlambda |
|---|---|
| not_prediction | not coded |
| tree_prediction | 1, 2, 3 |
| parallelogram_prediction | 3 |

**texCoord_lambda**: This signed fixed-point indicates the weight given to an ancestor for prediction. The number of bits used for this field is equal to **texCoord_quant** + 3. The 3 leading bits represent the integer part, and the **texCoord_quant** remaining bits the fractional part.

#### 6.3.11.9 ce_SNHC_header

**ce_SNHC_n_proj_surface_spheres**: The number of Projected Surface Spheres. Typically, this number is equal to 1.

**ce_SNHC_x_coord_center_point**: The x-coordinate (in 32-bit IEEE floating point format) of the center point (typically the gravity point of the object) of the Projected Surface Sphere.

**ce_SNHC_y_coord_center_point**: The y-coordinate (in 32-bit IEEE floating point format) of the center point (typically the gravity point of the object) of the Projected Surface Sphere.

**ce_SNHC_z_coord_center_point**: The z-coordinate (in 32-bit IEEE floating point format) of the center point (typically the gravity point of the object) of the Projected Surface Sphere.

**ce_SNHC_normalized_screen_distance_factor**: This indicates where the virtual screen is placed, compared to the radius of the Projected Surface Sphere. The distance between the center point of the Projected Surface Sphere and the virtual screen is equal to *ce_SNHC_radius/( ce_SNHC_normalized_screen_distance_factor+1)*. Note that *ce_SNHC_radius* is specified for each Projected Surface Sphere, while *ce_SNHC_normalized_screen_distance_factor* is specified only once.

**ce_SNHC_radius**: The radius (in 32-bit IEEE floating point format) of the Projected Surface Sphere.


© ISO/IEC 2004 – All rights reserved

**ce_SNHC_min_proj_surface**: The minimal projected surface value (in 32-bit IEEE floating point format) on the corresponding Projected Surf ace Sphere. This value is often (but not necessarily) equal to one of the *ce_SNHC_proj_surface* values.

**ce_SNHC_n_proj_points**: The number of points on the Projected Surface Sphere in which the projected surface will be transmitted. For all other points, the projected surface is determined by linear interpolation. *ce_SNHC_n_proj_points* is typically small (e.g. 20) for the first Projected Surface Sphere and very small (e.g. 3) for additional Projected Surface Spheres.

**ce_SNHC_sphere_point_coord**: This indicates the index of the point position in a octahedron, as explained in "inverse quantisation" section (see subclause 7.13.8.4).

**ce_SNHC_proj_surface**: The projected surface (in 32-bit IEEE floating point format) in the point specified by *ce_SNHC_sphere_point_coord*.

#### 6.3.11.10 connected component

**has_stitches**: This boolean value indicates if stitches are applied for the current connected component (within itself or between the current component and connected components previously decoded). This field is arithmetic coded.

#### 6.3.11.11 vertex_graph

**vg_simple**: This boolean value indicates if the current vertex graph is simple. A simple vertex graph does not contain any loop. This field is arithmetic coded.

**vg_last**: This boolean value indicates if the current run is the last run starting from the current branching vertex. This field is not coded for the first run of each branching vertex, i.e. when the skip_last variable is true. When not coded the value of **vg_last** for the current vertex run is considered to be false. This field is arithmetic coded.

**vg_forward_run**: This boolean value indicates if the current run is a new run. If it is not a new run, it is a previously traversed run, indicating a loop in the graph. This field is arithmetic coded.

**vg_loop_index**: This unsigned integer indicates the index of run to which the current loop connects. Its unary representation (see Table 6-67) is arithmetic coded. If the variable openloops is equal to **vg_loop_index**, the trailing '1' in the unary representation is omitted.

**Table 6-67 — Unary representation of the vg_loop_index field**

| vg_loop_index | unary representation |
|---|---|
| 0 | 1 |
| 1 | 01 |
| 2 | 001 |
| 3 | 0001 |
| 4 | 00001 |
| 5 | 000001 |
| 6 | 0000001 |
| ... | |
| openloops-1 | openloops-1 0's |

**vg_run_length**: This unsigned integer indicates the length of the current vertex run. Its unary representation (see Table 6-68) is arithmetic coded.

© ISO/IEC 2004 – All rights reserved

**Table 6-68 — Unary representation of the vg_run_length field**

| vg_run_length | unary representation |
|---|---|
| 1 | 1 |
| 2 | 01 |
| 3 | 001 |
| 4 | 0001 |
| 5 | 00001 |
| 6 | 000001 |
| 7 | 0000001 |
| 8 | 00000001 |
| N | n-1 0's followed by 1 |

**vg_leaf**: This boolean value indicates if the last vertex of the current run is a leaf vertex. If it is not a leaf vertex, it is a branching vertex. This field is arithmetic coded.

**vg_loop**: This boolean value indicates if the leaf of the current run connects to a branching vertex of the graph, indicating a loop. This field is arithmetic coded.

#### 6.3.11.12 stitches

**stitch_cmd**: This boolean value indicates if a stitching command of the type PUSH, POP or GET is associated to the current vertex. This field is arithmetic coded.

**stitch_pop_or_get**: This boolean value indicates if a stitching command of the type POP or GET is associated to the current vertex. This field is arithmetic coded.

**stitch_pop**: This boolean value indicates if a stitching command of the type POP is associated to the current vertex. This field is arithmetic coded.

**stitch_stack_index**: This unsigned integer value indicates the depth in the anchor stack where the anchor which the current vertex will be stitched to is located. This field is arithmetic coded.

**stitch_incr_length**: This integer value indicates the incremental length of the current stitch that must be added or subtracted to the length that is currently stored at the anchor. This field is arithmetic coded.

**stitch_incr_length_sign**: This boolean value indicates if the stitch_incr_length is negative. This field is arithmetic coded.

**stitch_push**: This boolean value indicates if the current vertex must be pushed into the stack of anchors. This field is arithmetic coded.

**stitch_reverse**: This boolean value indicates whether the current vertex must be stitched to its anchor using a reverse stitch as opposed to a forward stitch which is the default behavior. This field is arithmetic coded.

**stitch_length**: This unsigned integer value. This field is arithmetic coded.

#### 6.3.11.13 triangle_tree

branch_position: This integer variable is used to store the last branching triangle in a partition.

**tt_run_length**: This unsigned integer indicates the length of the current triangle run. Its unary representation (see Table 6-69) is arithmetic coded.


© ISO/IEC 2004 – All rights reserved

**Table 6-69 — Unary representation of the tt_run_length field**

| tt_run_length | unary representation |
|---|---|
| 1 | 1 |
| 2 | 01 |
| 3 | 001 |
| 4 | 0001 |
| 5 | 00001 |
| 6 | 000001 |
| 7 | 0000001 |
| 8 | 00000001 |
| N | n-1 0's followed by 1 |

**tt_leaf**: This boolean value indicates if the last triangle of the current run is a leaf triangle. If it is not a leaf triangle, it is a branching triangle. This field is arithmetic coded.

**triangulated**: This boolean value indicates if the current component contains triangles only. This field is arithmetic coded.

marching_triangle: This boolean value is determined by the position of the triangle in the triangle tree. If **marching_triangle** is 0, the triangle is a leaf or a branch. Otherwise, the triangle is a run.

**marching_edge**: This boolean value indicates the marching edge of an edge inside a triangle run. If **marching_edge** is false, it stands for a march to the left, otherwise it stands for a march to the right. This field is arithmetic coded.

**polygon_edge**: This boolean value indicates whether the base of the current triangle is an edge that should be kept when reconstructing the 3D mesh object. If the base of the current triangle is not kept, the edge is discarded. This field is arithmetic coded.

**codap_branch_len**: This unsigned integer indicates the length of the next branch to be traversed. The length of this value is the log scaled value of the size of the bounding loop table.

#### 6.3.11.14 triangle

**td_orientation**: This boolean value informs the decoder the traversal order of tt/td pair at a branch. This field is arithmetic coded. Table 6-70 shows the admissible values.

**Table 6-70 — Admissible values for td_orientation**

| td_orientation | traversal order |
|---|---|
| 0 | right branch first |
| 1 | left branch first |

visited: This variable indicates if the current vertex has been visited or not. When **codap_bdry_pred** is '1', visited is true for the vertices visited in the current partition. However, when **codap_bdry_pred** is '0', visited is true for the vertices visited in the previous partitions as well as in the current partition.

vertex_index: This variable indicates the index of the current vertex in the vertex array.

**no_ancestors**: This boolean value is true if there are no ancestors to use for prediction of the current vertex.

**coord_bit**: This boolean value indicates the value of a geometry bit. This field is arithmetic coded.

© ISO/IEC 2004 – All rights reserved

**coord_leading_bit**: This boolean value indicates the value of a leading geometry bit. This field is arithmetic coded.

**coord_sign_bit**: This boolean value indicates the sign of a geometry sample. This field is arithmetic coded.

**coord_trailing_bit**: This boolean value indicates the value of a trailing geometry bit. This field is arithmetic coded.

**normal_bit**: This boolean value indicates the value of a normal bit. This field is arithmetic coded.

**normal_leading_bit**: This boolean value indicates the value of a leading normal bit. This field is arithmetic coded.

**normal_sign_bit**: This boolean value indicates the sign of a normal sample. This field is arithmetic coded.

**normal_trailing_bit**: This boolean value indicates the value of a trailing normal bit. This field is arithmetic coded.

**color_bit**: This boolean value indicates the value of a color bit. This field is arithmetic coded.

**color_leading_bit**: This boolean value indicates the value of a leading color bit. This field is arithmetic coded.

**color_sign_bit**: This boolean value indicates the sign of a color sample. This field is arithmetic coded.

**color_trailing_bit**: This boolean value indicates the value of a trailing color bit. This field is arithmetic coded.

**texCoord_bit**: This boolean value indicates the value of a texture bit. This field is arithmetic coded.

**texCoord_leading_bit**: This boolean value indicates the value of a leading texture bit. This field is arithmetic coded.

**texCoord_sign_bit**: This boolean value indicates the sign of a texture sample. This field is arithmetic coded.

**texCoord_trailing_bit**: This boolean value indicates the value of a trailing texture bit. This field is arithmetic coded.

#### 6.3.11.15 3DMeshObject_Refinement_Layer

**3D_MORL_start_code**: This is a unique 16-bit code that is used for synchronization purpose. The value of this code is always '0000 0000 0011 0010'.

**morl_id**: This 8-bit unsigned integer specifies a unique id for the forest split component.

**connectivity_update**: This 2-bit variable indicates whether the forest split operation results in a refinement of the connectivity of the mesh or not.

**Table 6-71 — Admissible values for connectivity_update**

| connectivity_update | meaning |
|---|---|
| 00 | not_updated |
| 01 | fs_update |
| 10 | reserved |
| 11 | reserved |

**pre_smoothing**: This boolean value indicates whether the current forest split operation uses a pre-smoothing step to globally predict vertex positions.

**post_smoothing**: This boolean value indicates whether the current forest split operation uses a post-smoothing step to remove quantisation artifacts.


© ISO/IEC 2004 – All rights reserved

**stuffing_bit**: This boolean value is always true.

**other_update**: This boolean value indicates whether updates for vertex coordinates and properties associated with faces and corners not incident to any tree of the forest follow in the bitstream or not.

**other_update**: this boolean value indicates whether updates for vertex coordinates and properties associated with faces and corners not incident to any tree of the forest follow in the bitstream or not.

#### 6.3.11.15.1 pre_smoothing_parameters

**pre_smoothing_n**: This integer value indicates the number of iterations of the pre-smoothing filter.

**pre_smoothing_lambda**: This float value is the first parameter of the pre-smoothing filter.

**pre_smoothing_mu**: This float value is the second parameter of the pre-smoothing filter.

#### 6.3.11.15.2 post_smoothing_parameters

**post_smoothing_n**: This integer value indicates the number of iterations of the pre-smoothing filter.

**post_smoothing_lambda**: This float value is the first parameter of the pre-smoothing filter.

**post_smoothing_mu**: This float value is the second parameter of the pre-smoothing filter.

#### 6.3.11.15.3 fs_pre_update

**pfs_forest_edge**: This boolean value indicates if an edge should be added to the forest built so far.

#### 6.3.11.15.4 smoothing_constraints

**smooth_with_sharp_edges**: This boolean value indicates if data is included in the bitstream to mark smoothing discontinuity edges or not. If **sharp_edges**==0 no edge is marked as a smoothing discontinuity edge. If smoothing discontinuity edges are marked, then both the pre-smoothing and post-smoothing filters take them into account.

**smooth_with_fixed_vertices**: This boolean value indicates if data is included in the bitstream to mark vertices which do not move during the smoothing process. If **smooth_with_fixed_vertices**==0 all vertices are allowed to move. If fixed vertices are marked, then both the pre-smoothing and post-smoothing filters take them into account.

**smooth_sharp_edge**: This boolean value indicates if a corresponding edge is marked as a smoothing discontinuity edge.

**smooth_fixed_vertex**: This boolean value indicates if a corresponding vertex is marked as a fixed vertex or not.

### 6.3.12 Upstream message

#### 6.3.12.1 upstream_message

**upstream_message_type**: This 3-bit value indicates the type of the upstream information as shown in Table 6-72. The meaning of each upstream type is as follows:

video_newpred: This upstream message conveys the decoding status of the receiver (decoder) for the NEWPRED mode. The NEWPRED is the error resilience tool by selecting the reference picture of the inter-frame coding according to the error condition of the network. This upstream message shows whether the decoder decode the forward video data correctly or not. This message returns corresponding to the VOP or Video Packet of the forward video data. Which type of message is required for the encoder is indicated in requested_upstream_message_type in the VOL header of the downstream data. The decoder definitions of NEWPRED are described in subclauses 7.14 and E.1.6.

© ISO/IEC 2004 – All rights reserved

SNHC_QoS: This upstream message conveys some information that reveals to the encoder (server) the performances of the decoder w.r.t. decoding and rendering 3D objects with different parameter settings, i.e. with a varying number of triangles, different screen coverages of the rendered objects and different rendering modes. Using this information, the encoder can restrict the 3D content to the capabilities of the decoder, using for instance mesh simplification and/or rendering mode selection.

**Table 6-72 — Meaning of upstream_message_type**

| upstream_message_type | meaning |
|---|---|
| 000 | reserved |
| 001 | video_newpred |
| 010 | SNHC_QoS |
| 011-111 | reserved |

#### 6.3.12.2 upstream_video_newpred

**newpred_upstream_message_type**: This indicates whether the corresponding NP segment is correctly decoded or not. Which type of message is required for the encoder is indicated in requested_upstream_message_type in the VOL header of the downstreamdata. In the other case, this indicates requesting intra refresh.

00: NP_NACK. It indicates the erroneous decoding of the NP segment.
01: NP_ACK. It indicates the correct decoding of the NP segment.
10: Intra refresh command.
11: Reserved.

**unreliable_flag**: This field presents only if newpred_upstream_message_type is 'NP_NACK'. The unreliable_flag is set to 1 when a reliable value of vop_id is not available at the decoder. (When the NP segment is erred, a reliable vop_id may not be available at the decoder. On the other hand, a reliable vop_id is available, when the decoder cannot decode due to the luck of the reference picture.)

0: reliable
1: unreliable

**vop_id**: When the newpred_upstream_message_type is 'NP_NACK' or 'NP_ACK', this indicates the ID of VOP which is incremented by 1 whenever a VOP is encoded. The vop_id is copied from the vop_id field of the NP segment header in the corresponding forward channel data when the reliable vop_id is available. Otherwise, it may happen in the case of NP_NACK that the vop_id is incremented by 1 from the reliable vop_id of the previously received NP segment in the same location of the current NP segment.

When the newpred_upstream_message_type is 'Intra refresh command', this indicates the ID of Intra refresh which is incremented by 1 whenever new refresh is required. The length of this field is 4 bits in the Intra refresh case. In the case that Intra refresh command is repeatedly returned for the same error until the proper action corresponding to the previous Intra refresh command reaches, this ID is set to the same number as the previous Intra refresh command.

**macroblock_number**: The macroblock_number is the macroblock address of the start of the corresponding NP segment or the refresh area.

**end_macroblock_number**: This field is present only when the newpred_upstream_message_type is 'Intra refresh command'. The end_macroblock_number is the macroblock address of the end of the refresh area.

**requested_vop_id_for_prediction**: This field is present only if newpred_upstream_message_type is 'NP_NACK'. This indicates the requested vop_id of the NP segment for reference by the decoder. Typically it is the vop_id of the last correctly decoded NP segment in the same location of the current NP segment.


© ISO/IEC 2004 – All rights reserved

#### 6.3.12.3 upstream_SNHC_QoS

**screen_width**: Screen width used during the calibration process. screen_width is expressed in number of pixels.

**screen_height**: Screen height used during the calibration process. screen_height is expressed in number of pixels.

**n_rendering_modes**: n_rendering_modes is the number of rendering modes for which information is transmitted.

**rendering_mode_type**: rendering_mode_type is the kind of rendering used during the calibration process for a particular performance curve. The different rendering types are coded according to the following table.

**Table 6-73 — Meaning of rendering_mode_type**

| rendering_mode_type | Rendering mode |
|---|---|
| 0000 | Wire-framed |
| 0001 | Flat shading |
| 0010 | Smooth shading |
| 0011 | Texture rendering |
| 0100-1111 | Reserved for later use |

**n_curves**: n_curves is the number of performance curves transmitted for one particular rendering mode.

**triangle_parameter**: triangle_parameter is the number of triangles in units of 64 triangles.

**n_points_on_curve**: n_points_on_curve is the number of points specified for one particular performance curve.

**screen_coverage_parameter**: screen_coverage_parameter is the number of pixels, expressed in percentage of the screen size. 0x00 corresponds to 0%, while 0xFF corresponds to 100%. All other points are determined by linear interpolation.

**frame_rate_value**: frame_rate_value is the frame rate for one particular point on one particular curve. For achieving enough precision, 12 bits are used. The 8 Most Significant Bits represent the integer value, the 4 Least Significant Bits represent the fractional value.

### 6.3.13 Studio Video Object

#### 6.3.13.1 Studio Visual Object

**visual_object_start_code**: The visual_object_start_code is the bit string '000001B5' in hexadecimal. It identifies the beginning of a visual object header.

**visual_object_verid**: This is a 4-bit code which identifies the version number of the visual object. Its meaning is defined in Table 6-74.

**Table 6-74 — Meaning of visual_object_verid**

| visual_object_verid | Meaning |
|---|---|
| 0000 | reserved |
| 0001 | object type listed in Table 9.1 |
| 0010 | object type listed in Table 9.2 |
| 0011 | reserved |
| 0100 | object type listed in Table 9.3 |
| 0101 – 1111 | reserved |

© ISO/IEC 2004 – All rights reserved

**visual_object_type**: The visual_object_type is a 4-bit code given in Table 6-75 which identifies the type of the visual object.

**Table 6-75 — Meaning of visual object type**

| code | visual object type |
|---|---|
| 0000 | reserved |
| 0001 | video ID |
| 0010 | still texture ID |
| 0011 | mesh ID |
| 0100 | face ID |
| 0101 | reserved |
| : | : |
| : | : |
| 1111 | reserved |

**video_object_start_code**: The video_object_start_code is a string of 32 bits. The first 27 bits are '0000 0000 0000 0000 0000 0001 000' in binary and the last 5-bits represent one of the values in the range of '00000' to '11111' in binary. The video_object_start_code identifies the beginning of a video object header.

#### 6.3.13.2 Extension and user data Studio

##### 6.3.13.2.1 Extension data

**extension_start_code**: The extension_start_code is the bit string '000001B8' in hexadecimal. It identifies the beginning of extensions.

##### 6.3.13.2.2 User data Studio

**user_data_start_code**: The user_data_start_code is the bit string '000001B2' in hexadecimal. It identifies the beginning of user data. The user data continues until receipt of another start code.

**user_data**: This is an 8 bit integer, an arbitrary number of which may follow one another. User data is defined by users for their specific applications. In the series of consecutive user_data bytes there shall not be a string of 23 or more consecutive zero bits.

##### 6.3.13.2.3 Sequence display extension

This specification does not define the display process. The information in this extension does not affect the decoding process and may be ignored by decoders that conform to this specification.

**extension_start_code_identifier**: This is a 4-bit integer which identifies the extension. See Table 6-4.

**video_format**: This is a three bit integer indicating the representation of the pictures before being coded in accordance with this specification. Its meaning is defined in Table 6-76. If the sequence_display_extension() is not present in the bitstream then the video format may be assumed to be "Unspecified video format".


© ISO/IEC 2004 – All rights reserved

**Table 6-76 — Meaning of video_format**

| video_format | Meaning |
|---|---|
| 000 | component |
| 001 | PAL |
| 010 | NTSC |
| 011 | SECAM |
| 100 | MAC |
| 101 | Unspecified video format |
| 110 | reserved |
| 111 | reserved |

**video_range**: This one-bit flag indicates the black level and range of the luminance and chrominance signals. In the case that sequence_display_extension() is not present in the bitstream, video_range is assumed to have the value 0 (a range of Y from 16 to 235 for 8-bit video).

**colour_description**: A flag which if set to '1' indicates the presence of colour_primaries, transfer_characteristics and matrix_coefficients in the bitstream.

**colour_primaries**: This 8-bit integer describes the chromaticity coordinates of the source primaries, and is defined in Table 6-77.

**Table 6-77 — Colour Primaries**

| Value | Primaries |
|---|---|
| 0 | (forbidden) |
| 1 | Recommendation ITU-R BT.709<br>primary x y<br>green 0,300 0,600<br>blue 0,150 0,060<br>red 0,640 0,330<br>white D65 0,3127 0,3290 |
| 2 | Unspecified Video<br>Image characteristics are unknown. |
| 3 | reserved |
| 4 | Recommendation ITU-R BT.470-2 System M<br>primary x y<br>green 0,210,71<br>blue 0,140,08<br>red 0,670,33<br>white C 0,310 0,316 |
| 5 | Recommendation ITU-R BT.470-2 System B, G<br>primary x y<br>green 0,290,60<br>blue 0,150,06<br>red 0,640,33<br>white D65 0,313 0,329 |

© ISO/IEC 2004 – All rights reserved

| | |
|---|---|
| 6 | SMPTE 170M<br>primary x y<br>green 0,310 0,595<br>blue 0,155 0,070<br>red 0,630 0,340<br>white D65 0,3127 0,3290 |
| 7 | SMPTE 240M (1987)<br>primary x y<br>green 0,310 0,595<br>blue 0,155 0,070<br>red 0,630 0,340<br>white D65 0,3127 0,3291 |
| 8 | Generic film (colour filters using Illuminant C)<br>primary x y<br>green 0,243 0,692 (Wratten 58)<br>blue 0,145 0,049 (Wratten 47)<br>red 0,681 0,319 (Wratten 25) |
| 9-255 | reserved |

In the case that sequence_display_extension() is not present in the bitstream or colour_description is zero the chromaticity is assumed to be that corresponding to colour_primaries having the value 1.

**transfer_characteristics**: This 8-bit integer describes the opto-electronic transfer characteristic of the source picture, and is defined in Table 6-78.

**Table 6-78 — Transfer Characteristics**

| Value | Transfer Characteristic |
|---|---|
| 0 | (forbidden) |
| 1 | Recommendation ITU-R BT.709<br>$V = 1,099\ L_c^{0,45} - 0,099$<br>for $1 \geq L_c \geq 0,018$<br>$V = 4,500\ L_c$<br>for $0,018 > L_c \geq 0$ |
| 2 | Unspecified Video<br>Image characteristics are unknown. |
| 3 | reserved |
| 4 | Recommendation ITU-R BT.470-2 System M<br>Assumed display gamma 2,2 |
| 5 | Recommendation ITU-R BT.470-2 System B, G<br>Assumed display gamma 2,8 |
| 6 | SMPTE 170M<br>$V = 1,099\ L_c^{0,45} - 0,099$<br>for $1 \geq L_c \geq 0,018$<br>$V = 4,500\ L_c$<br>for $0,018 > L_c \geq 0$ |


© ISO/IEC 2004 – All rights reserved

| | |
|---|---|
| 7 | SMPTE 240M (1987)<br>$V = 1{,}1115\ L_c^{0{,}45} - 0{,}1115$<br>for $L_c \geq 0{,}0228$<br>$V = 4{,}0\ L_c$<br>for $0{,}0228 > L_c$ |
| 8 | Linear transfer characteristics<br>i.e. $V = L_c$ |
| 9 | Logarithmic transfer characteristic (100:1 range)<br>$V = 1.0\text{-}\mathrm{Log}_{10}(Lc)/2$<br>for $1 = L_c = 0.01$<br>$V = 0.0$<br>for $0.01 > L_c$ |
| 10 | Logarithmic transfer characteristic (316.22777:1 range)<br>$V = 1.0\text{-}\mathrm{Log}_{10}(Lc)/2.5$<br>for $1 = L_c = 0.0031622777$<br>$V = 0.0$<br>for $0.0031622777 > L_c$ |
| 11-255 | reserved |

In the case that sequence_display_extension() is not present in the bitstream or colour_description is zero the transfer characteristics are assumed to be those corresponding to transfer_characteristics having the value 1.

**matrix_coefficients**: This 8-bit integer describes the matrix coefficients used in deriving luminance and chrominance signals from the green, blue, and red primaries, and is defined in Table 6-79.

In this table:

$E'_Y$ is analogue with values between 0 and 1

$E'_{PB}$ and $E'_{PR}$ are analogue between the values -0,5 and 0,5

$E'_R$, $E'_G$ and $E'_B$ are analogue with values between 0 and 1

White is defined as $E'_y = 1$, $E'_{PB} = 0$, $E'_{PR} = 0$; $E'_R = E'_G = E'_B = 1$.

Y, Cb and Cr are related to $E'_Y$, $E'_{PB}$ and $E'_{PR}$ by the following formulae:

if **video_range=0:**

$$Y = ( 219 * 2^{n-8} * E'_Y ) + 2^{n-4}$$
$$Cb = ( 224 * 2^{n-8} * E'_{PB} ) + 2^{n-1}$$
$$Cr = ( 224 * 2^{n-8} * E'_{PR} ) + 2^{n-1}$$

if **video_range=1:**

$$Y = ((2^n -1) * E'_Y )$$
$$Cb = ((2^n -1) * E'_{PB} ) + 2^{n-1}$$
$$Cr = ((2^n -1) * E'_{PR} ) + 2^{n-1}$$

for n bit video.

© ISO/IEC 2004 – All rights reserved

For example, for 8 bit video,

**video_range**=0 gives a range of Y from 16 to 235, Cb and Cr from 16 to 240;

**video_range**=1 gives a range of Y from 0 to 255, Cb and Cr from 0 to 255.

NOTE The decoding process given by this specification limits output sample values for Y, Cr and Cb to the range [0:255]. Thus sample values outside the range implied by the above equations may occasionally occur at the output of the decoding process. In particular the sample values 0 and 255 may occur.

**Table 6-79 — Matrix Coefficients**

| Value | Matrix |
|---|---|
| 0 | (forbidden) |
| 1 | Recommendation ITU-R BT.709<br>$E'_Y = 0{,}7154\ E'_G + 0{,}0721\ E'_B + 0{,}2125\ E'_R$<br>$E'_{PB} = -0{,}386\ E'_G + 0{,}500\ E'_B - 0{,}115\ E'_R$<br>$E'_{PR} = -0{,}454\ E'_G - 0{,}046\ E'_B + 0{,}500\ E'_R$ |
| 2 | Unspecified Video<br>Image characteristics are unknown. |
| 3 | reserved |
| 4 | FCC<br>$E'_Y = 0{,}59\ E'_G + 0{,}11\ E'_B + 0{,}30\ E'_R$<br>$E'_{PB} = -0{,}331\ E'_G + 0{,}500\ E'_B - 0{,}169\ E'_R$<br>$E'_{PR} = -0{,}421\ E'_G - 0{,}079\ E'_B + 0{,}500\ E'_R$ |
| 5 | Recommendation ITU-R BT.470-2 System B, G<br>$E'_Y = 0{,}587\ E'_G + 0{,}114\ E'_B + 0{,}299\ E'_R$<br>$E'_{PB} = -0{,}331\ E'_G + 0{,}500\ E'_B - 0{,}169\ E'_R$<br>$E'_{PR} = -0{,}419\ E'_G - 0{,}081\ E'_B + 0{,}500\ E'_R$ |
| 6 | SMPTE 170M<br>$E'_Y = 0{,}587\ E'_G + 0{,}114\ E'_B + 0{,}299\ E'_R$<br>$E'_{PB} = -0{,}331\ E'_G + 0{,}500\ E'_B - 0{,}169\ E'_R$<br>$E'_{PR} = -0{,}419\ E'_G - 0{,}081\ E'_B + 0{,}500\ E'_R$ |
| 7 | SMPTE 240M (1987)<br>$E'_Y = 0{,}701\ E'_G + 0{,}087\ E'_B + 0{,}212\ E'_R$<br>$E'_{PB} = -0{,}384\ E'_G + 0{,}500\ E'_B - 0{,}116\ E'_R$<br>$E'_{PR} = -0{,}445\ E'_G - 0{,}055\ E'_B + 0{,}500\ E'_R$ |
| 8-255 | reserved |

In the case that sequence_display_extension() is not present in the bitstream or colour_description is zero the matrix coefficients are assumed to be those corresponding to matrix_coefficients having the value 1.

**display_horizontal_size**: See display_vertical_size.

**display_vertical_size**: display_horizontal_size and display_vertical_size together define a rectangle which may be considered as the "intended display's" active region. If this rectangle is smaller than the encoded frame size then the display process may be expected to display only a portion of the encoded frame. Conversely if the display


© ISO/IEC 2004 – All rights reserved

rectangle is larger than the encoded frame size then the display process may be expected to display the reconstructed frames on a portion of the display device rather than on the whole display device.

display_horizontal_size shall be in the same units as horizontal_size (samples of the encoded frames).

display_vertical_size shall be in the same units as vertical_size (lines of the encoded frames).

display_horizontal_size and display_vertical_size do not affect the decoding process but may be used by the display process that is not standardised in this specification.

#### 6.3.13.2.4 Quant matrix extension

Each quantisation matrix has a default set of values. When a video_object_start_code is decoded all matrices shall be reset to their default values. User defined matrices may be downloaded and this can occur in a quant_matrix_extension().

With 4:2:0 data only two matrices are used, one for intra blocks the other for non-intra blocks.

With 4:2:2 or 4:4:4 data four matrices are used. Both an intra and a non-intra matrix are provided for both luminance blocks and for chrominance blocks. Note however that it is possible to download the same user defined matrix into both the luminance and chrominance matrix at the same time.

The default matrix for intra blocks (both luminance and chrominance) is:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 8 | 17 | 18 | 19 | 21 | 23 | 25 | 27 |
| 17 | 18 | 19 | 21 | 23 | 25 | 27 | 28 |
| 20 | 21 | 22 | 23 | 24 | 26 | 28 | 30 |
| 21 | 22 | 23 | 24 | 26 | 28 | 30 | 32 |
| 22 | 23 | 24 | 26 | 28 | 30 | 32 | 35 |
| 23 | 24 | 26 | 28 | 30 | 32 | 35 | 38 |
| 25 | 26 | 28 | 30 | 32 | 35 | 38 | 41 |
| 27 | 28 | 30 | 32 | 35 | 38 | 41 | 45 |

The default matrix for non-intra blocks (both luminance and chrominance) is:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| 19 | 20 | 21 | 22 | 23 | 24 | 26 | 27 |
| 20 | 21 | 22 | 23 | 25 | 26 | 27 | 28 |
| 21 | 22 | 23 | 24 | 26 | 27 | 28 | 30 |
| 22 | 23 | 24 | 26 | 27 | 28 | 30 | 31 |
| 23 | 24 | 25 | 27 | 28 | 30 | 31 | 33 |

**load_intra_quantiser_matrix**: This is a one-bit flag which is set to '1' if intra_quantiser_matrix follows. If it is set to '0' then there is no change in the values that shall be used.

**intra_quantiser_matrix**: This is a list of sixty-four 8-bit unsigned integers. The new values, encoded in the default zigzag scanning order as described in 7.16.4.2.1, replace the previous values. The first value shall always be 8 (values 1 to 7 and 9 to 255 are reserved). For all of the 8-bit unsigned integers, the value zero is forbidden. With 4:2:2 and 4:4:4 data the new values shall be used for both the luminance intra matrix and the chrominance intra matrix. However the chrominance intra matrix may subsequently be loaded with a different matrix except when rgb_component is set to '1'.

© ISO/IEC 2004 – All rights reserved

**load_non_intra_quantiser_matrix**: This is a one-bit flag which is set to '1' if non_intra_quantiser_matrix follows. If it is set to '0' then there is no change in the values that shall be used.

**non_intra_quantiser_matrix**: This is a list of sixty-four 8-bit unsigned integers. The new values, encoded in the default zigzag scanning order as described in 7.16.3.2.1, replace the previous values. For all the 8-bit unsigned integers, the value zero is forbidden. With 4:2:2 and 4:4:4 data the new values shall be used for both the luminance non-intra matrix and the chrominance non-intra matrix. However the chrominance non-intra matrix may subsequently be loaded with a different matrix.

**load_chroma_intra_quantiser_matrix**: This is a one-bit flag which is set to '1' if chroma_intra_quantiser_matrix follows. If it is set to '0' then there is no change in the values that shall be used. If chroma_format is "4:2:0" or rgb_components is set to '1', this flag shall take the value '0'.

**chroma_intra_quantiser_matrix**: This is a list of sixty-four 8-bit unsigned integers. The new values, encoded in the default zigzag scanning order as described in 7.16.4.2.1, replace the previous values. The first value shall always be 8 (values 1 to 7 and 9 to 255 are reserved). For all of the 8-bit unsigned integers, the value zero is forbidden.

**load_chroma_non_intra_quantiser_matrix**: This is a one-bit flag which is set to '1' if chroma_non_intra_quantiser_matrix follows. If it is set to '0' then there is no change in the values that shall be used. If chroma_format is "4:2:0" or rgb_components is set to '1' this flag shall take the value '0'.

**chroma_non_intra_quantiser_matrix**: This is a list of sixty-four 8-bit unsigned integers. The new values, encoded in the default zigzag scanning order as described in 7.16.4.2.1, replace the previous values. For all the 8-bit unsigned integers, the value zero is forbidden.

**load_intra_quantiser_matrix_grayscale[i]**: This is a one-bit flag which is set to '1' if intra_quantiser_matrix_grayscale[i] follows. If it is set to '0' then there is no change in the values that shall be used.

**intra_quantiser_matrix_grayscale[i]**: This is a list of sixty-four 8-bit unsigned integers defining the grayscale intra alpha quantisation matrix to be used. The semantics and the default quantisation matrix are identical to those of intra_quantiser_matrix.

**load_non_intra_quantiser_matrix_grayscale[i]**: This is a one-bit flag which is set to '1' if non_intra_quantiser_matrix_grayscale[i] follows. If it is set to '0' then there is no change in the values that shall be used.

**non_intra_quantiser_matrix_grayscale[i]**: This is a list of sixty-four 8-bit unsigned integers defining the grayscale nonintra alpha quantisation matrix to be used. The semantics and the default quantisation matrix are identical to those of nonintra_quantiser_matrix.

#### 6.3.13.2.5 Picture display extension

This specification does not define the display process. The information in this extension does not affect the decoding process and may be ignored by decoders that conform to this specification.

The picture display extension allows the position of the display rectangle whose size is specified in sequence_display_extension() to be moved on a picture-by-picture basis. One application for this is the implementation of pan-scan.

**frame_centre_horizontal_offset**: This is a 16-bit signed integer giving the horizontal offset in units of 1/16th sample. A positive value shall indicate that the centre of the reconstructed frame lies to the right of the centre of the display rectangle.

**frame_centre_vertical_offset**: This is a 16-bit signed integer giving the vertical offset in units of 1/16th sample. A positive value shall indicate that the centre of the reconstructed frame lies below the centre of the display rectangle.

The dimensions of the display rectangular region are defined in the sequence_display_extension(). The coordinates of the region within the coded picture are defined in the picture_display_extension().


© ISO/IEC 2004 – All rights reserved

The centre of the reconstructed frame is the centre of the rectangle defined by video_object_layer_width and video_object_layer_height.

Since (in the case of an interlaced sequence) a coded VOP may relate to one, two or three decoded fields the picture_display_extension() may contain up to three offsets.

The number of frame centre offsets in the picture_display_extension() shall be defined as follows:

```
if ( progressive_sequence == 1) {
    if ( repeat_first_field == '1' ) {
        if ( top_field_first == '1' )
            number_of_frame_centre_offsets = 3
        else
            number_of_frame_centre_offsets = 2
    } else {
        number_of_frame_centre_offsets = 1
    }
} else {
    if (vop_structure == "field") {
        number_of_frame_centre_offsets = 1
    } else {
        if (repeat_first_field == '1' )
            number_of_frame_centre_offsets = 3
        else
            number_of_frame_centre_offsets = 2
    }
}
```

A picture_display_extension() shall not occur unless a sequence_display_extension() followed the StudioVideoObject().

In the case that a given picture does not have a picture_display_extension() then the most recently decoded frame centre offset shall be used. Note that each of the missing frame centre offsets have the same value (even if two or three frame centre offsets would have been contained in the picture_display_extension() had been present). Following a StudioVisualObject() the value zero shall be used for all frame centre offsets until a picture_display_extension() defines non-zero values.

Figure 6-22 illustrates the picture display parameters. As shown the frame centre offsets contained in the picture_display_extension() shall specify the position of the centre of the reconstructed frame from the centre of the display rectangle.

NOTES -

1 The display rectangle may also be larger than the reconstructed frame.

2 Even in a field VOP the frame_centre_vertical_offset still represents the offset of the centre of the frame in $1/16^{th}$s of a **frame** line (not a line in the field).

3 In the example of Figure 6-22 both frame_centre_horizontal_offset and frame_centre_vertical_offset have negative values.

© ISO/IEC 2004 – All rights reserved

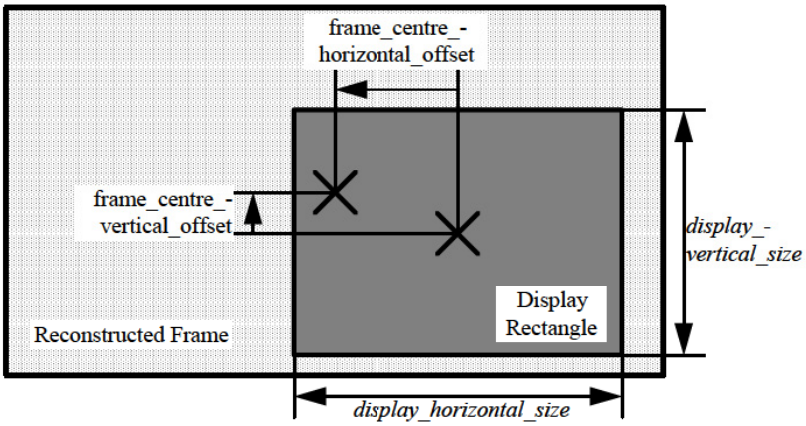



**Figure 6-22 — Frame centre offset parameters**

#### 6.3.13.2.5.1 Pan-scan

The frame centre offsets may be used to implement pan-scan in which a rectangular region is defined which may be panned around the entire reconstructed frame.

By way of example only; this facility may be used to identify a 3/4 aspect ratio window in a 9/16 coded VOP format. This would allow a decoder to produce usable VOPs for a conventional definition television set from an encoded format intended for enhanced definition. The 3/4 aspect ratio region is intended to contain the "most interesting" region of the VOP.

The 3/4 region is defined by display_horizontal_size and display_vertical_size. The 9/16 frame size is defined by video_object_later_width and video_object_layer_height.

### 6.3.13.2.6 Copyright extension

**copyright_flag**: This is a one bit flag. When copyright_flag is set to '1', it indicates that the source video material encoded in all the coded pictures following the copyright extension, in coding order, up to the next copyright extension or end of sequence code, is copyrighted. The copyright_identifier and copyright_number identify the copyrighted work. When copyright_flag is set to '0', it does not indicate whether the source video material encoded in all the coded pictures following the copyright extension, in coding order, is copyrighted or not.

**copyright_identifier**: This is a 8-bit integer given by a Registration Authority as designated by ISO/IEC JTC1/SC29. Value zero indicates that this information is not available. The value of copyright_number shall be zero when copyright_identifier is equal to zero.

When copyright_flag is set to '0', copyright_identifier has no meaning and shall have the value 0.

**original_or_copy**: This is a one bit flag. It is set to '1' to indicate that the material is an original, and set to '0' to indicate that it is a copy.

**reserved**: This is a 7-bit integer, reserved for future extension. It shall have the value zero.

**copyright_number_1**: This is a 20-bit integer, representing bits 44 to 63 of copyright_number.

**copyright_number_2**: This is a 22-bit integer, representing bits 22 to 43 of copyright_number.

**copyright_number_3**: This is a 22-bit integer. representing bits 0 to 21 of copyright_number.


© ISO/IEC 2004 – All rights reserved

**copyright_number**: This is a 64-bit integer, derived from copyright_number_1, copyright_number_2, and copyright_number_3 as follows:

copyright_number **= (**copyright_number_1 << 44) + (copyright_number_2 << 22) + copyright_number_3.

The meaning of copyright_number is defined only when copyright_flag is set to '1'. In this case, the value of copyright_number identifies uniquely the copyrighted work marked by the copyrighted extension. The value 0 for copyright_number indicates that the identification number of the copyrighted work is not available.

When copyright_flag is set to '0', copyright_number has no meaning and shall have the value 0.

#### 6.3.13.2.7 Camera parameters extension

**camera_id** – The number in camera_id identifies a camera.

**height_of_image_device** – This is a 22-bit unsigned integer which specifies the height of image device. Its value shall be measured to a resolution of 0,001 millimeter and having a range of zero to 4 194,303 mm.

**focal_length** – This is a 22-bit unsigned integer which specifies the focal length. Its value shall be measured to a resolution of 0,001 millimeter and having a range of zero to 4 194,303 mm.

**f_number** – This is a 22-bit unsigned integer which specifies the F-number. F-number is defined by (focal_length)/(effective aperture of lens). Its value shall be measured to a resolution of 0,001and having a range of zero to 4 194,303.

**vertical_angle_of_view** – This is a 22-bit unsigned integer which specifies the vertical angle of the field of view as determined between the top and bottom edges of the image device. Its value shall be measured to a resolution of 0,0001 degree and having a range of zero to 180 degrees.

**camera_position_x_upper, camera_position_y_upper, camera_position_z_upper** – These words constitute the 16 most significant bits of camera_position_x, camera_position_y and camera_position_z respectively.

**camera_position_x_lower, camera_position_y_lower, camera_position_z_lower** – These words constitute the 16 least significant bits of camera_position_x, camera_position_y and camera_position_z respectively.

**camera_position_x, camera_position_y, camera_position_z** – A set of these values specifies the position of the optical principal point of the camera in a user-specified world coordinate system. Each of these values shall be measured to a resolution of 0,001 millimeter and having a range of +2 147 483,647 mm to –2 147 483,648 mm. The camera_position_x is a 32-bit signed (two's complement) integer, the 16 least significant bits are defined in camera_position_x_lower, the 16 most significant bits are defined in camera_position_x_upper. The camera_position_y is a 32-bit signed (two's complement) integer, the 16 least significant bits are defined in camera_position_y_lower, the 16 most significant bits are defined in camera_position_y_upper. The camera_position_z is a 32-bit signed (two's complement) integer, the 16 least significant bits are defined in camera_position_z_lower, the 16 most significant bits are defined in camera_position_z_upper.

**camera_direction_x, camera_direction_y, camera_direction_z** – A set of these values specifies the direction of the camera. The direction of the camera is defined by using the vector from optical principal point to a point which is in front of the camera and is on the optical axis of the camera. Each of these values is a 22-bit signed (two's complement) integer and having a range of +2 097 151 to –2 097 152.

**image_plane_vertical_x, image_plane_vertical_y, image_plane_vertical_z** – A set of these values specifies the upper direction of the camera. The upper direction of the camera is defined by using the vector which is parallel to the side edge of the image device and is from bottom edge to top edge. Each of these values is a 22-bit signed (two's complement) integer and is having a range of +2 097 151 to –2 097 152.

Figure 6-23 explains these terms pictorially.

© ISO/IEC 2004 – All rights reserved

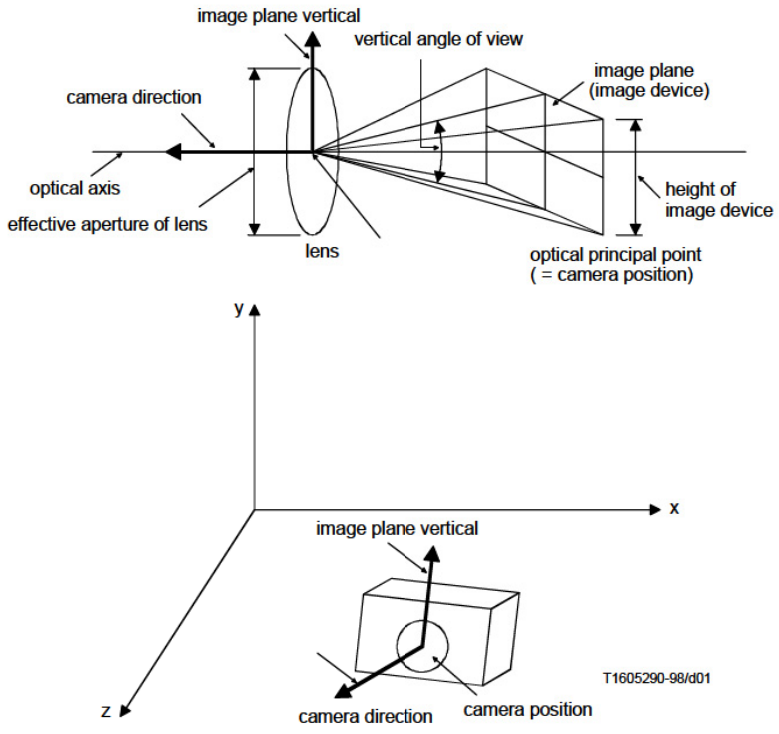



**Figure 6-23 — Camera parameters**

#### 6.3.13.2.8 ITU-T extension

The use of this extension is defined in Annex A of ITU-T Recommendation H.320.

#### 6.3.13.2.9 VLC code extension

In order to prevent start code emulation in StudioMacroblock() or StudioBlock(), a set of VLC codes that does not cause start code emulation for any combination of syntax elements shall be encoded.

**load_vlc_code**: This is a four bit integer indicating the presence of downloadable VLC codes for quantised AC coefficients. The value of load_intra_vlc_code and load_inter_vlc_code is derived from this code. Semantics are defined in the Table 6-80.

**Table 6-80 — Meaning of load_vlc_code**

| load_vlc_code | Meaning |
|---|---|
| 0000 | forbidden |
| 0001 | load_intra_vlc_code = 1<br>load_inter_vlc_code = 0 |
| 0010 | load_intra_vlc_code = 0<br>load_inter_vlc_code = 1 |
| 0011 | load_intra_vlc_code = 1<br>load_inter_vlc_code = 1 |
| 0100 – 1111 | reserved |


© ISO/IEC 2004 – All rights reserved

If load_intra_vlc_code is set to '1', the VLC codes defined in Table B.50 to Table 6-61 shall be replaced with the codes indicated by intra_vlc_length[j][i] and intra_vlc_code[j][i]. If it is set to '0', there is no change in Table B.50 to Table 6-61.

If load_inter_vlc_code is set to '1', the VLC codes defined in Table B.62 to Table B.73 shall be replaced with the codes indicated by inter_vlc_length[j][i] and inter_vlc_code[j][i]. If it is set to '0', there is no change in Table B.62 to Table B.73.

**intra_vlc_length[j][i]**: This is a four bit unsigned integer indicating the length of the VLC code of the entry No. i in the Table No. j, which is defined in Table B.50 to Table 6-61. The actual length is intra_vlc_length[j][i]+1. The length of each VLC code in each table shall be limited to less than 17. The length of the VLC code of entry No.21 in each table shall also be limited less than 14.

**intra_vlc_code[j][i]**: This is a 16 bit unsigned integer indicating the VLC code of the entry No. i in the Table No. j, which is defined in Table B.50 to Table 6-61. The most significant bits of the length indicated by intra_vlc_length[j][i] is the actual VLC code, and the remaining bits shall be filled with '1'.

**inter_vlc_length[j][i]**: This is a four bit unsigned integer indicating the length of the VLC code of the entry No. i in the Table No. j, which is defined in Table B.62 to Table B.73. The actual length is inter_vlc_length[j][i]+1. The length of each VLC code in each table shall be limited to less than 17. The length of the VLC code of entry No.21 in each table shall also be limited less than 14.

**inter_vlc_code[j][i]**: This is a 16 bit unsigned integer indicating the VLC code of the entry No. i in the Table No. j, which is defined in Table B.62 to Table B.73. The most significant bits of the length indicated by intra_vlc_length[j][i] is the actual VLC code, and the remaining bits shall be filled with '1'.

#### 6.3.13.3 Studio Video Object Layer

**video_object_layer_start_code**: The video_object_layer_start_code is a string of 32 bits. The first 28 bits are '0000 0000 0000 0000 0000 0001 0010' in binary and the last 4-bits represent one of the values in the range of '0000' to '1111' in binary. The video_object_layer_start_code identifies the beginning of a video object layer header.

**random_accessible_vol**: This flag may be set to "1" to indicate that every VOP in this VOL is individually decodable. If all of the VOPs in this VOL are intra-coded VOPs and some more conditions are satisfied then random_accessible_vol may be set to "1". The flag random_accessible_vol is not used by the decoding process. random_accessible_vol is intended to aid random access or editing capability. This shall be set to "0" if any of the VOPs in the VOL are non-intra coded or certain other conditions are not fulfilled.

**video_object_type_indication**: Constrains the following bitstream to use tools from the indicated object type only, e.g. Simple Object or Core Object, as shown in Table 6-81.

**Table 6-81 — FLC table for video_object_type indication**

| Video Object Type | Code |
|---|---|
| Reserved | 00000000 |
| Simple Object Type | 00000001 |
| Simple Scalable Object Type | 00000010 |
| Core Object Type | 00000011 |
| Main Object Type | 00000100 |
| N-bit Object Type | 00000101 |
| Basic Anim. 2D Texture | 00000110 |
| Anim. 2D Mesh | 00000111 |
| Simple Face | 00001000 |

© ISO/IEC 2004 – All rights reserved

| | |
|---|---|
| Still Scalable Texture | 00001001 |
| Advanced Real Time Simple | 00001010 |
| Core Scalable | 00001011 |
| Advanced Coding Efficiency | 00001100 |
| Advanced Scalable Texture | 00001101 |
| Simple Studio Object Type | 00001110 |
| Core Studio Object Type | 00001111 |
| Reserved | 00010000 - 11111111 |

**video_object_layer_verid**: This is a 4-bit code which identifies the version number of the video object layer. Its meaning is defined in Table 6-82. If both visual_object_verid and video_object_layer_verid exist, the semantics of video_object_layer_verid supersedes the other.

**Table 6-82 — Meaning of video_object_layer_verid**

| video_object_layer_verid | Meaning |
|---|---|
| 0000 | reserved |
| 0001 | object type listed in Table 9-1 |
| 0010 | object type listed in Table 9-2 |
| 0011 | reserved |
| 0100 | object type listed in Table 9-3 |
| 0101 – 1111 | reserved |

**video_object_layer_shape**: This is a 2-bit integer defined in Table 6-83. It identifies the shape type of a video object layer.

**Table 6-83 — Video Object Layer shape type**

| Shape format | Meaning |
|---|---|
| 00 | rectangular |
| 01 | binary |
| 10 | binary only |
| 11 | grayscale |

**video_object_layer_shape_extension**: This is a 4-bit integer defined in Table 6-84. It identifies the number (up to 3) and type of auxiliary components that can be used, including the grayscale shape (ALPHA) component. Only a limited number of types and combinations are defined in Table 6-83. More applications are possible by selection of the USER DEFINED type. If the video_object_layer_shape does not indicate grayscale shape, this code shall be set to '1111'.


© ISO/IEC 2004 – All rights reserved

**Table 6-84 — Semantic meaning of video_object_layer_shape_extension**

| video_object_layer_shape_extension | aux_comp_type[0] | aux_comp_type[1] | aux_comp_type[2] | aux_comp_count |
|---|---|---|---|---|
| 0000 | ALPHA | NO | NO | 1 |
| 0001 | DISPARITY | NO | NO | 1 |
| 0010 | ALPHA | DISPARITY | NO | 2 |
| 0011 | DISPARITY | DISPARITY | NO | 2 |
| 0100 | ALPHA | DISPARITY | DISPARITY | 3 |
| 0101 | DEPTH | NO | NO | 1 |
| 0110 | ALPHA | DEPTH | NO | 2 |
| 0111 | TEXTURE | NO | NO | 1 |
| 1000 | USER DEFINED | NO | NO | 1 |
| 1001 | USER DEFINED | USER DEFINED | NO | 2 |
| 1010 | USER DEFINED | USER DEFINED | USER DEFINED | 3 |
| 1011 | ALPHA | USER DEFINED | NO | 2 |
| 1100 | ALPHA | USER DEFINED | USER DEFINED | 3 |
| 1101-1110 | t.b.d. | t.b.d. | t.b.d. | t.b.d. |
| 1111 | NO | NO | NO | |

**progressive_sequence**: When set to '1' the coded video sequence contains only progressive frame-VOPs. When progressive_sequence is set to '0' the coded video sequence may contain both frame-VOPs and field-VOPs, and frame-VOP may be progressive or interlaced frames.

**rgb_components**: this is a one bit flag indicating if the components of the sequence are the RGB format or not. If it is set to '1', the components are RGB and the component representing green color is coded in the same way as the luminance component and the blue and red components are coded in the same way as the chrominance components. That is, the green component shall be treated as the Y component, the blue component shall be as the Cb component, and the red component shall be as the Cr component. This flag does not affect the decoding process except the restriction on the value of chroma_format and parameters related to the quantiser matrices.

**chroma_format**: This is a two bit integer indicating the chrominance format as defined in the Table 6-85. If rgb_components is set to '1', this shall be set to '11'.

**Table 6-85 — Meaning of chroma_format**

| chroma_format | Meaning |
|---|---|
| 00 | reserved |
| 01 | 4:2:0 |
| 10 | 4:2:2 |
| 11 | 4:4:4 |

**bits_per_pixel**: This field specifies the video data precision in bits per pixel. It may take different values for different video object layers within a single video object. A value of 12 in this field would indicate 12 bits per pixel. This field may take values between 4 and 12. The same number of bits per pixel is used in the luminance and two chrominance planes.

© ISO/IEC 2004 – All rights reserved

**video_object_layer_width**: The video_object_layer_width is a 14-bit unsigned integer representing the width of the displayable part of the luminance component of VOPs in pixel units. The width of the encoded luminance component of VOPs in macroblocks is (video_object_layer_width+15)/16. The displayable part is left-aligned in the encoded VOPs.

**video_object_layer_height**: The video_object_layer_height is a 14-bit unsigned integer representing the height of the displayable part of the luminance component of the frame in lines.

In the case that progressive_sequence is '1' the height of the encoded luminance component of frames in macroblocks, mb_height, is (video_object_layer_height+15)/16.

In the case that progressive_sequence is '0' the height of the encoded luminance component of frame VOPs in macroblocks, mb_height, is 2*((video_object_layer_height + 31)/32). The height of the encoded luminance component of field VOPs in macroblocks, mb_height, is ((video_object_layer_height + 31)/32).

The displayable part is top-aligned in the encoded VOPs.

**aspect_ratio_info**: This is a four-bit integer which defines the value of pixel aspect ratio. Table 6-86 shows the meaning of the code. If aspect_ratio_info indicates extended PAR, pixel_aspect_ratio is represented by par_width and par_height. The par_width and par_height shall be relatively prime.

**Table 6-86 — Meaning of pixel aspect ratio**

| aspect_ratio_info | pixel aspect ratios |
|---|---|
| 0000 | Forbidden |
| 0001 | 1:1 (Square) |
| 0010 | 12:11 (625-type for 4:3 picture) |
| 0011 | 10:11 (525-type for 4:3 picture) |
| 0100 | 16:11 (625-type stretched for 16:9 picture) |
| 0101 | 40:33 (525-type stretched for 16:9 picture) |
| 0110-1110 | Reserved |
| 1111 | extended PAR |

**par_width**: This is an 8-bit unsigned integer which indicates the horizontal size of pixel aspect ratio. A zero value is forbidden.

**par_height**: This is an 8-bit unsigned integer which indicates the vertical size of pixel aspect ratio. A zero value is forbidden.

**frame_rate_code**: This is a four-bit integer used to define frame_rate as shown in Table 6-87.

If progressive_sequence is '1' the period between two successive frames at the output of the decoding process is the reciprocal of the frame_rate. If progressive_sequence is '0' the period between two successive fields at the output of the decoding process is half of the reciprocal of the frame_rate.


© ISO/IEC 2004 – All rights reserved

**Table 6-87 — frame_rate**

| frame_rate_code | frame_rate |
|---|---|
| 0000 | forbidden |
| 0001 | 24 000÷1001 (23,976…) |
| 0010 | 24 |
| 0011 | 25 |
| 0100 | 30 000÷1001 (29,97…) |
| 0101 | 30 |
| 0110 | 50 |
| 0111 | 60 000÷1001 (59,94…) |
| 1000 | 60 |
| … | reserved |
| 1111 | reserved |

The VBV constraint is defined in Annex D.

**first_half_bit_rate, latter_half_bit_rate**: The bit rate is a 30-bit unsigned integer which specifies the bitrate of the bitstream measured in units of 400 bits/second, rounded upwards. The value zero is forbidden. This value is divided to two parts. The most significant bits are in first_half_bit_rate (15 bits) and the least significant bits are in latter_half_bit_rate (15 bits). The marker_bit is inserted between the first_half_bit_rate and the latter_half_bit_rate in order to avoid the start code emulation. The instantaneous video object layer channel bit rate seen by the encoder is denoted by $R_{vol}(t)$ in bits per second. If the bit_rate (i.e. first_half_bit_rate and latter_half_bit_rate) field in the VOL header is present, it defines a peak rate (in units of 400 bits per second; a value of 0 is forbidden) such that $R_{vol}(t) <= 400 \times bit\_rate$ Note that $R_{vol}(t)$ counts only visual syntax for the current elementary stream (also see Annex D).

**first_half_vbv_buffer_size, latter_half_vbv_buffer_size**: vbv_buffer_size is an 18-bit unsigned integer. This value is divided into two parts. The most significant bits are in first_half_vbv_buffer_size (15 bits) and the least significant bits are in latter_half_vbv_buffer_size (3 bits), The VBV buffer size is specified in units of 16384 bits. The value 0 for vbv_buffer_size is forbidden. Define B = 16384 × vbv_buffer_size to be the VBV buffer size in bits.

**first_half_vbv_occupancy, latter_half_vbv_occupancy**: The vbv_occupancy is a 26-bit unsigned integer. This value is divided to two parts. The most significant bits are in first_half_vbv_occupancy (11 bits) and the least significant bits are in latter_half_vbv_occupancy (15 bits). The marker_bit is inserted between the first_vbv_buffer_size and the latter_half_vbv_buffer_size in order to avoid the start code emulation. The value of this integer is the VBV occupancy in 64-bit units just before the removal of the first VOP following the VOL header. The purpose for the quantity is to provide the initial condition for VBV buffer fullness.

**low_delay**: This is a one-bit flag which when set to '1' indicates the VOL contains no B-VOPs.

**mpeg2_stream**: This is a one bit flag which when set to '1' indicates if the decoded bitstream is transcoded from one which conforms to ISO/IEC 13818-2:2000 according to information specified in ANNEX Q. This flag affects the inverse quantisation process as defined in 7.16.4.3 in order to decode the bitstream accurately.

**alpha_bits_per_pixel[i]**: This field specifies the video data precision of alpha planes_for grayscale alpha or auxiliary component i=0,1,2 in bits per pixel. It may take different values for different video object layers within a single video object. A value of 12 in this field would indicate 12 bits per pixel. This field may take values between 4 and 12.

**minimum_alpha_level[i]**: T his is a 12-bit unsigned integer which specifies the level for complete transparency of alpha signals for grayscale alpha or auxiliary component i=0,1,2.

© ISO/IEC 2004 – All rights reserved

**maximum_alpha_level[i]**: This is a 12-bit unsigned integer which specifies the level for complete opacity of alpha signals for grayscale alpha or auxiliary component i=0,1,2. maximum_alpha_level shall be grater than minimum_alpha_level.

**composition_method**: This is a one bit flag which indicates which blending method is to be applied to the video object in the compositor. When set to '0', cross-fading shall be used. When set to '1', additive mixing shall be used. See subclause 7.16.6.3.5.

**linear_composition**: This is a one bit flag which indicates the type of signal used by the compositing process. When set to '0', the video signal in the format from which it was produced by the video decoder is used. When set to '1', linear signals are used. See subclause 7.16.6.3.5.

**sprite_enable**: This is a one-bit flag which when set to '1' indicates the presence of sprites.

**sprite_width**: This is a 20-bit unsigned integer which identifies the horizontal dimension of the sprite.

**sprite_height**: This is a 20-bit unsigned integer which identifies the vertical dimension of the sprite.

**sprite_left_coordinate**: This is a 20-bit signed integer which defines the left-edge of the sprite. The value of sprite_left_coordinate shall be divisible by two.

**sprite_top_coordinate**: This is a 20-bit signed integer which defines the top edge of the sprite. The value of sprite_left_coordinate shall be divisible by two.

**no_of_sprite_warping_points**: This is a 6-bit unsigned integer which represents the number of points used in sprite warping. When its value is 0 and when sprite_enable is set to '1', warping is identity (stationary sprite) and no coordinates need to be coded. When its value is 4, a perspective transform is used. When its value is 1,2 or 3, an affine transform is used. Further, the case of value 1 is separated as a special case from that of values 2 or 3. Table 6-88 shows the various choices.

**Table 6-88 — Number of point and implied warping function**

| Number of points | warping function |
|---|---|
| 0 | Stationary |
| 1 | Translation |
| 2,3 | Affine |
| 4 | Perspective |

**sprite_warping_accuracy**: This is a 2-bit code which indicates the quantisation accuracy of motion vectors used in the warping process for sprites. Table 6-89 shows the meaning of various codewords.

**Table 6-89 — Meaning of sprite warping accuracy codewords**

| code | sprite_warping_accuracy |
|---|---|
| 00 | 1/2 pixel |
| 01 | 1/4 pixel |
| 10 | 1/8 pixel |
| 11 | 1/16 pixel |

**sprite_brightness_change**: This is a one-bit flag which when set to '1' indicates a change in brightness during sprite warping ; alternatively, a value of '0' means no change in brightness.


© ISO/IEC 2004 – All rights reserved

**sprite_defocusing**: This is a one-bit flag which when set to `1` indicates a change in defocusing during sprite warping; alternatively, a value of `0` means no change in defocusing.

**sprite_lens_distortion**: This is a one-bit flag which when set to `1` indicates that the lens distortion exists during sprite warping; alternatively, a value of `0` means there is no lens distortion.

#### 6.3.13.4 Group of Studio Video Object Plane

**group_vop_start_code**: The group_vop_start_code is the bit string '000001B3' in hexadecimal. It identifies the beginning of a Group of Studio VOP header.

**closed_gov**: This is a one-bit flag which indicates the nature of the predictions used in the first consecutive B-VOPs (if any) immediately following the first coded I-VOP after the group of studio VOP header. The closed_gov is set to '1' to indicate that these B-VOPs have been encoded using only backward prediction or intra coding. This bit is provided for use during any editing which occurs after encoding. If the previous VOPs have been removed by editing, broken_link may be set to '1' so that a decoder may avoid displaying these B-VOPs following the first I-VOP following the group of studio VOP header. However if the closed_gov bit is set to '1', then the editor may choose not to set the broken_link bit as these B-VOPs can be correctly decoded.

**broken_link**: This is a one-bit flag which shall be set to '0' during encoding. It is set to '1' to indicate that the first consecutive B-VOPs (if any) immediately following the first coded I-frame following the group of studio VOP header may not be correctly decoded because the reference frame which is used for prediction is not available (because of the action of editing). A decoder may use this flag to avoid displaying frames that cannot be correctly decoded.

##### 6.3.13.4.1 Time Code SMPTE12M

**Time_code[63...0]** (64bit): Time-code format defined in Annex P shall be transparent with SMPTE 12M.

#### 6.3.13.5 Studio Video Object Plane

**vop_start_code**: This is the bit string '000001B6' in hexadecimal. It marks the start of a video object plane.

**temporal_reference**: The temporal_reference is a 10-bit unsigned integer associated with each coded VOP.

The following simple specification applies only when low_delay is equal to zero.

When a coded frame is in the form of two field VOPs, the temporal_reference associated with each VOP shall be the same (it is called the temporal_reference of the coded frame). The temporal_reference of each coded frame shall increment by one modulo 1024 when examined in display order at the output of the decoding process, except when a group of studio VOP header occurs. Among the frames coded after a group of studio VOP header, the temporal_reference of the coded frame that is displayed first shall be set to zero.

The following more general specification applies when low_delay is equal to zero or one.

If VOP A is not a big VOP, i.e., the VBV buffer is only examined once before the coded VOP A is removed from the VBV buffer, and if N is the temporal_reference of VOP A, then the temporal_reference of VOP B immediately following VOP A in display order is equal to :

- 0 if there is a group of studio VOP header present between VOP A and VOP B ( in coded order).
- (N+1) % 1024 if VOP B is a frame VOP or is the first of a pair of field VOPs.
- N if VOP B is the second field of a pair of field VOPs.

When low_delay is equal to one, there may be situations where the VBV buffer shall be re-examined several times before removing a coded VOP (referred to as a big VOP) from the VBV buffer.

© ISO/IEC 2004 – All rights reserved

If VOP A is a big VOP and if K is the number of times that the VBV buffer is re-examined as defined in Annex D (K>0), if N is the temporal_reference of VOP A, then the temporal_reference of VOP B immediately following VOP A in display order is equal to :

- K % 1024 if there is a group of studio VOP header present between VOP A and VOP B (in coded order).
- (N+K+1) % 1024 if VOP B is a frame VOP or is the first field of a pair of field VOPs.
- (N+K) % 1024 if VOP B is the second field of a pair of field VOPs.

NOTE - If the big VOP is the first field of a frame coded with field VOPs, then the temporal_reference of the two field VOPs of that coded frame are not identical.

**vop_structure**: This is a 2-bit integer defined in the Table 6-90.

**Table 6-90 — Meaning of vop_structure**

| vop_structure | Meaning |
|---|---|
| 00 | reserved |
| 01 | Top Field |
| 10 | Bottom Field |
| 11 | Frame picture |

When a frame is encoded in the form of two field VOPs both fields must be of the same vop_coding_type, except where the first encoded field is an I-VOP in which case the second may be either an I-VOP or a P-VOP.

The first encoded field of a frame may be a top-field or a bottom field, and the next field must be of opposite parity.

When a frame is encoded in the form of two field VOPs the following syntax elements may be set independently in each field VOP:

- vop_fcode[0][0], vop_fcode[0][1]
- vop_fcode[1][0], vop_fcode[1][1]
- dct_precision, alpha_dct_precision[i]
- intra_dc_precision, q_scale_type, alpha_intra_dc_precision[i], alpha_q_scale_type[i]
- alternate_scan
  - temporal_reference

**vop_coding_type**: The vop_coding_type identifies whether a VOP is an intra-coded VOP (I), predictive-coded VOP (P), bidirectionally predictive-coded VOP (B) or sprite VOP (S). The meaning of vop_coding_type is defined in Table 6-91.

**Table 6-91 — Meaning of vop_coding_type**

| vop_coding_type | coding method |
|---|---|
| 00 | intra-coded (I) |
| 01 | predictive-coded (P) |
| 10 | bidirectionally-predictive-coded (B) |
| 11 | sprite (S) |


© ISO/IEC 2004 – All rights reserved

**vop_coded**: This is a 1-bit flag which when set to '0' indicates that no subsequent data exists for the VOP. In this case, the following decoding rule applies: For an arbitrarily shaped VO (i.e. when the shape type of the VO is either 'binary' or 'binary only'), the alpha plane of the reconstructed VOP shall be completely transparent. For a rectangular VO (i.e. when the shape type of the VO is 'rectangular'), the corresponding rectangular alpha plane of the VOP, having the same size as its luminance component, shall be completely transparent. If there is no alpha plane being used in the decoding and composition process of a rectangular VO, the reconstructed VOP is filled with the respective content of the immediately preceding VOP for which vop_coded!=0.

**vop_width**: This is a 14-bit unsigned integer which specifies the horizontal size, in pixel units, of the rectangle that includes the VOP. The width of the encoded luminance component of VOP in macroblocks is (vop_width+15)/16. The rectangle part is left-aligned in the encoded VOP. A zero value is forbidden.

**vop_height**: This is a 14-bit unsigned integer which specifies the vertical size, in pixel units, of the rectangle that includes the VOP. The height of the encoded luminance component of VOP in macroblocks is (vop_height+15)/16. The rectangle part is top-aligned in the encoded VOP. A zero value is forbidden.

**vop_horizontal_mc_spatial_ref**: This is a 14-bit signed integer which specifies, in pixel units, the horizontal position of the top left of the rectangle defined by horizontal size of vop_width. The value of vop_horizontal_mc_spatial_ref shall be divisible by two. This is used for decoding and for picture composition.

**vop_vertical_mc_spatial_ref**: This is a 14-bit signed integer which specifies, in pixel units, the vertical position of the top left of the rectangle defined by vertical size of vop_height. The value of vop_vertical_mc_spatial_ref shall be divisible by two for progressive and divisible by four for interlaced motion compensation. This is used for decoding and for picture composition.

**top_field_first**: The meaning of this element depends upon vop_structure, progressive_sequence and repeat_first_field.

If progressive_sequence is equal to '0', this flag indicates what field of a reconstructed frame is output first by the decoding process:

In a field VOP top_field_first shall have the value '0', and the only field output by the decoding process is the decoded field VOP.

In a frame VOP top_field_first being set to '1' indicates that the top field of the reconstructed frame is the first field output by the decoding process. top_field_first being set to '0' indicates that the bottom field of the reconstructed frame is the first field output by decoding process

If progressive_sequence is equal to '1', this flag, combined with repeat_first_field, indicates how many times (one, two or three) the reconstructed frame is output by the decoding process.

If repeat_first_field is set to 0, top_field_first shall be set to '0'. In this case the output of the decoding process corresponding to this reconstructed frame consists of one progressive frame.

If top_field_first is set to 0 and repeat_first_field is set to '1', the output of the decoding process corresponding to this reconstructed frame consists of two identical progressive frames.

If top_field_first is set to 1 and repeat_first_field is set to '1', the output of the decoding process corresponding to this reconstructed frame consists of three identical progressive frames.

**repeat_first_field**: This flag is applicable only in a frame VOP, in a field VOP it shall be set to zero and does not affect the decoding process.

If progressive_sequence is equal to 0 and progressive_frame is equal to 0, repeat_first_field shall be zero, and the output of the decoding process corresponding to this reconstructed frame consists of two fields.

If progressive_sequence is equal to 0 and progressive_frame is equal to 1:

© ISO/IEC 2004 – All rights reserved

If this flag is set to 0, the output of the decoding process corresponding to this reconstructed frame consists of two fields. The first field (top or bottom field as identified by top_field_first) is followed by the other field.

If it is set to 1, the output of the decoding process corresponding to this reconstructed frame consists of three fields. The first field (top or bottom field as identified by top_field_first) is followed by the other field, then the first field is repeated.

If progressive_sequence is equal to 1:

If this flag is set to 0, the output of the decoding process corresponding to this reconstructed frame consists of one frame.

If it is set to 1, the output of the decoding process corresponding to this reconstructed frame consists of two or three frames, depending on the value of top_field_first.

**progressive_frame**: If progressive_frame is set to 0 it indicates that the two fields of the frame are interlaced fields in which an interval of time of the field period exists between (corresponding spatial samples) of the two fields. In this case the following restriction applies:

- repeat_first_field shall be zero (two field duration).

If progressive_frame is set to 1 it indicates that the two fields (of the frame) are actually from the same time instant as one another. In this case a number of restrictions to other parameters and flags in the bitstream apply:

- vop_structure shall be “Frame”
- if progressive_sequence is equal to one, frame_pred_frame_dct shall be 1

**intra_predictors_reset**: This flag indicates whether the macroblock to be decoded shall be decoded independently of any parameters of the previous macroblocks of the current VOP even if the macroblock is not located at the start of a slice. If this flag is set to 1, each predictor, such as for intra dc coefficient, is reset in the same way as done at the start of a sliece. Otherwise, predictions are used between neighbouring macroblocks in the same slice.

**alternate_scan**: This flag affects the decoding of transform coefficient data as described in 7.16.4.2.

**frame_pred_frame_dct**: If this flag is set to ‘1’ then only frame-DCT and frame prediction are used. In a field VOP it shall be ‘0’. frame_pred_frame_dct shall be ‘1’ if progressive_sequence is ‘1’. This flag affects the syntax of the bitstream.

**dct_precision**: This is a 2-bit integer defining the value of base_quantiser scale according to Table 6-92. This flag affects the inverse quantisation process as defined in 7.16.4.3.

**Table 6-92 — DCT precision**

| dct_precision | base_quantiser scale |
|---|---|
| 00 | 1.000 |
| 01 | 0.500 |
| 10 | 0.250 |
| 11 | 0.125 |

In case of mpeg2_stream = 1, dct_precision shall be set to ‘00’.

**intra_dc_precision**: This is a 2-bit integer defined in the Table 6-93.


© ISO/IEC 2004 – All rights reserved

**Table 6-93 — Intra DC precision**

| intra_dc_precision | Precision (bits) |
|---|---|
| 00 | bits_per_pixel |
| 01 | bits_per_pixel+1 |
| 10 | bits_per_pixel+2 |
| 11 | bits_per_pixel+3 |

The inverse quantisation process for the Intra DC coefficients is modified by this parameter as explained in 7.16.3.3.1.

**q_scale_type**: This flag affects the inverse quantisation process as described in 7.16.4.3.2.2.

**vop_fcode[s][t]**: A 4 bit unsigned integer taking values 1 through 9, or 15. The value zero is forbidden and the values 10 through 14 are reserved. It is used in the decoding of motion vectors, see 7.16.7.4.1.

In an I-VOP vop_fcode[s][t] is not used (since motion vectors are not used) and shall take the value 15 (all ones).

Similarly, in an I-VOP or a P-VOP vop_fcode[1][t] is not used in the decoding process (since it refers to backwards motion vectors) and shall take the value 15 (all ones).

See Table 7-38 for the meaning of the indices; s and t.

**dead_zone_disable**: This is a one bit flag which affect the inverse quantisation process as described in 7.16.4.2.3.

**alpha_dct_precision[i]**: This is a 2-bit integer defined in the Table 6-94. This flag affects the inverse quantisation process of the alpha macroblock in the same way as dct_precision does for that of the luminance macroblock.

**Table 6-94 — alpha base quantiser scale**

| alpha_dct_precision | alpha_base_quantiser scale |
|---|---|
| 00 | 1.000 |
| 01 | 0.500 |
| 10 | 0.250 |
| 11 | 0.125 |

**alpha_intra_dc_precision[i]**: This is a 2-bit integer defined in the Table 6-95.

**Table 6-95 — Alpha Intra DC precision**

| alpha_intra_dc_precision | Precision (bits) |
|---|---|
| 00 | bits_per_pixel |
| 01 | bits_per_pixel+1 |
| 10 | bits_per_pixel+2 |
| 11 | bits_per_pixel+3 |

© ISO/IEC 2004 – All rights reserved

The inverse quantisation process for the Intra DC coefficients of alpha channel is modified by this parameter in the same way as the luminance component.

**alpha_q_scale_type[i]**: This flag affects the inverse quantisation process for alpha channel in the same way as the luminance components.

**vop_sprite_brightness_change**: This is a one-bit flag which when set to '1' indicates a change in brightness during sprite warping, alternatively, a value of '0' means no change in brightness.

**composite_display_flag**: This flag is set to 1 to indicate that the following fields are of use when the input pictures have been coded as (analogue) composite video prior to encoding into a bitstream that complies with this specification. If it is set to 0 then these parameters do not occur in the bitstream.

The information relates to the picture that immediately follows the extension. In the case that this picture is a frame picture the information relates to the first field of that frame. The equivalent information for the second field may be derived (there is no way to represent it in the bitstream).

NOTES

1 The various syntactic elements that are included in the bitstream if composite_display_flag is '1' are not used in the decoding process.

2 repeat_first_field will cause a composite video field to be repeated out of the 4-field or 8-field sequence. It is recommended that repeat_first_field and composite_display_flag are not both set simultaneously.

**v_axis**: A 1-bit integer used only when the bitstream represents a signal that had previously been encoded according to PAL systems. v_axis is set to 1 on a positive sign, v_axis is set to 0 otherwise.

**field_sequence**: A 3-bit integer which defines the number of the field in the eight field sequence used in PAL systems or the four field sequence used in NTSC systems as defined in the Table 6-96.

**Table 6-96 — Definition of field_sequence**

| field sequence | frame | field |
|---|---|---|
| 000 | 1 | 1 |
| 001 | 1 | 2 |
| 010 | 2 | 3 |
| 011 | 2 | 4 |
| 100 | 3 | 5 |
| 101 | 3 | 6 |
| 110 | 4 | 7 |
| 111 | 4 | 8 |

**sub_carrier**: This is a 1-bit integer. When set to zero it means the sub-carrier/line frequency relationship is correct. When set to 1 the relationship is not correct.

**burst_amplitude**: This is a 7-bit integer defining the burst amplitude (for PAL and NTSC only). The amplitude of the sub-carrier burst is quantised as a Recommendation ITU-R BT.601 luminance signal, with the MSB omitted.

**sub_carrier_phase**: This is an 8-bit integer defining the phase of the reference sub-carrier at the field-synchronisation datum with respect, to field start as defined in Recommendation ITU-R BT.470. See Table 6-97.


© ISO/IEC 2004 – All rights reserved

**Table 6-97 — Definition of sub_carrier_phase**

| sub_carrier_phase | Phase |
|---|---|
| 0 | ([360°÷256] * 0) |
| 1 | ([360°÷256] * 1) |
| ... | ... |
| 255 | ([360°÷256] * 255) |

**extra_bit_picture**: A bit indicates the presence of the following extra information. If extra_bit_picture is set to '1', extra_information_picture will follow it. If it is set to '0', there are no data following it. extra_bit_picture shall be set to '0', the value '1' is reserved for possible future extensions defined by ITU-T|ISO/IEC.

**extra_information_picture**: Reserved. A decoder conforming to this specification that encounters extra_information_picture in a bitstream shall ignore it (i.e. remove from the bitstream and discard). A bitstream conforming to this specification shall not contain this syntax element.

#### 6.3.13.6 Studio Sprite coding

warping_mv_code(dmv): The codeword for each differential motion vector consists of a VLC indicating the length of the dmv code (dmv_length) and a FLC, dmv_code-, with dmv_length bits. The codewords are listed in Table B.88.

brightness_change_factor (): The codeword for brightness_change_factor consists of a variable length code denoting brightness_change_factor_size and a fixed length code, brightness_change_factor, of brightness_change_factor_size bits (sign bit included). The codewords are listed in Table B.89.

**defocusing_control_parameter**: This specifies the defocusing control parameter for luminance and chrominance sprite images by the following defocusing process.

The defocusing is performed by a filtering process. The applied filter $P_s(x, y)$ is shown in Figure 6-24.

$$P_s(x,y) = \begin{cases} 1/4a^2 & |x| \le a, |y| \le a \\ 0 & else \end{cases}$$

The value "a" is used as the defocusing control parameter.

The defocusing control parameter "a" for chrominance signals is divided by two in vertical direction or horizontal direction according to the chroma format.

- 4:4:4 format"a" is not divided.

- 4:2:2 format"a" is divided by two in horizontal direction.

- 4:2:0 format"a" is divided by two both in horizontal direction and in vertical direction.

In case the filter taps lie outside the image, outer pixels shall be obtained by the padding process as defined in subclause 7.16.5.9.1.

© ISO/IEC 2004 – All rights reserved

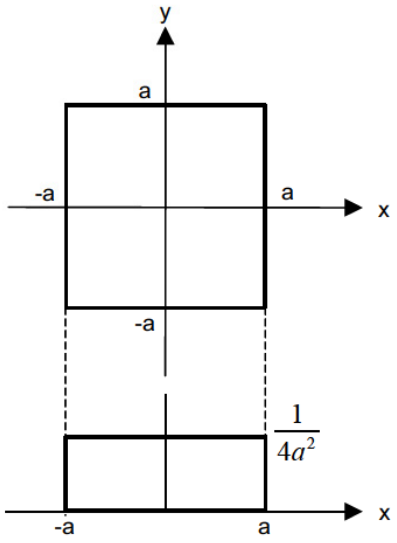



**Figure 6-24 — Defocusing filter**

The length of this value varies according to the sprite warping accuracy. Table 6-98 shows the length of the defocusing control parameter.

**Table 6-98 — Length of defocusing control parameter**

| sprite warping accuracy | length of defocusing control parameter | defocusing control parameter |
|---|---|---|
| ½ pixel | 9 bits | 0-511 |
| ¼ pixel | 10 bits | 0-1023 |
| 1/8 pixel | 11 bits | 0-2047 |
| 1/16 pixel | 12 bits | 0-4095 |

lens_distortion_parameter(): The codeword for lens_distortion_parameter consists of the following 4 syntax elements. **lens_distortion_parameter_1** and **lens_distortion_parameter_2** are 16-bit signed (two's complement) integer and **lens_center_horizontal** and **lens_center_vertical** are 14-bit unsigned integer. These parameters represent camera lens distortion of the following equation.

$$x' = u(\tilde{x} - c_x) + c_x$$
$$y' = u(\tilde{y} - c_y) + c_y$$

with

$$u = 1 + R(k_1 \cdot \tfrac{1}{4W^3} + Rk_2 \cdot \tfrac{1}{2W^5})$$
$$R = (\tilde{x} - c_x)^2 + s^2(\tilde{y} - c_y)^2$$


© ISO/IEC 2004 – All rights reserved

where

$(\tilde{x}, \tilde{y})$ is the position of a distorted point

$(x', y')$ is the position of no radial lens distortion point

$(c_x, c_y)$ is the lens center position in the image coordinates

(**lens_center_horizontal, lens_center_vertical**)

$k_1$ is the first lens distortion coefficient

**lens_distortion_parameter_1**

$k_2$ is the second lens distortion coefficient

**lens_distortion_parameter_2**

$W$ is the maximum image size

the larger one of **video_object_layer_width** or **video_object_layer_height** in StudioVideoObjectLayer()

$s$ is the aspect ratio of each pixel

**aspect_ratio_info** in StudioVideoObjectLayer()

*Note: These syntax elements provide no normative decoding procedure.*

#### 6.3.13.7 Studio Slice

**slice_start_code**: This is the bit string '000001B7' in hexadecimal. It marks the start of a slice.

**macroblock_number**: This is a variable length code with length between 1 and 14 bits. It identifies the macroblock number within a VOP. The number of the top-left macroblock in a VOP shall be zero. The macroblock number increases from left to right and from top to bottom. The actual length of the code depends on the total number of macroblocks in the VOP calculated according to Table 6-99, the code itself is a binary representation of the macroblock number.

**Table 6-99 — Length of macroblock_number code**

| length of macroblock_number code | ((vop_width+15)/16) * ((vop_height+15)/16) |
|---|---|
| 1 | 1-2 |
| 2 | 3-4 |
| 3 | 5-8 |
| 4 | 9-16 |
| 5 | 17-32 |
| 6 | 33-64 |
| 7 | 65-128 |
| 8 | 129-256 |
| 9 | 257-512 |
| 10 | 513-1024 |

© ISO/IEC 2004 – All rights reserved

| | |
|---|---|
| 11 | 1025-2048 |
| 12 | 2049-4096 |
| 13 | 4097-8192 |
| 14 | 8193-16384 |

**quantiser_scale_code**: A 5 bit unsigned integer in the range 1 to 31. The decoder shall use this value until another quantiser_scale_code is encountered either in StudioSlice() or StudioMacroblock(). The value zero is forbidden.

**alpha_quantiser_scale_code[i]**: A 5 bit unsigned integer in the range 1 to 31. The decoder shall use this value until another alpha_quantiser_scale_code[i] is encountered either in StudioSlice() or StudioMacroblock(). The value zero is forbidden.

**slice_extension_flag**: This flag shall be set to '1' to indicate the presence of intra_slice, slice_VOP_enable and slice_VOP_id in the bitstream.

**intra_slice**: This flag shall be set to '0' if any of the macroblocks in the slice are non-intra macroblocks. If all of the macroblocks are intra macroblocks then intra_slice may be set to '1'. intra_slice may be omitted from the bitstream (by setting intra_slice_flag to '0') in which case it shall be assumed to have the value zero.

intra_slice is not used by the decoding process. intra_slice is intended to aid a DSM application in performing FF/FR.

**slice_VOP_id_enable**: This flag controls the semantics of slice_VOP_id. If slice_VOP_id_enable is set to "0", slice_VOP_id is not used by this specification and shall have the value zero. If slice_VOP_id_enable is set to "1", slice_VOP_id may have a value different from zero.

slice_VOP_id_enable must have the same value in all the slices of a VOP. slice_VOP_id_enable may be omitted from the bitstream (by setting slice_extension_flag to '0') in which case it shall be assumed to have the value zero.

slice_VOP_id_enable is not used by the decoding process.

**slice_VOP_id**: This is a 6 bit integer. If slice_VOP_id_enable is set to "0", slice_VOP_id is not used by this specification and shall have the value zero. If slice_VOP_id_enable is set to "1", slice_VOP_id is application defined and may have any value, with the constraint that slice_VOP_id shall have the same value in all the slices of a VOP.

slice_VOP_id is not used by the decoding process. slice_VOP_id is intended to aid recovery on severe bursts of errors for certain types of applications. For example the application may increment slice_VOP_id with each transmitted VOP, so that in case of severe burst error, when several slices are lost, the decoder can know if the slice following the burst error belongs to the current VOP or to another VOP, which may be the case if at least a VOP header has been lost.

**extra_bit_slice**: This flag indicates the presence of the following extra information. If extra_bit_slice is set to '1', extra_information_slice will follow it. If it is set to '0', there are no data following it. extra_bit_slice shall be set to '0', the value '1' is reserved for possible future extensions defined by ITU-T|ISO/IEC.

**extra_information_slice**: Reserved. A decoder conforming to this specification that encounters extra_information_slice in a bitstream shall ignore it (i.e. remove from the bitstream and discard). A bitstream conforming to this specification shall not contain this syntax element.

#### 6.3.13.8 Studio Macroblock

**not_coded**: This is a 1-bit flag which signals if a macroblock is coded or not. When set to'1' it indicates that a macroblock is not coded and no further data is included in the bitstream for this macroblock (with the exception of alpha data that may be present). The decoder shall treat this macroblock as 'inter' with motion vector equal to zero


© ISO/IEC 2004 – All rights reserved

and no DCT coefficient data for P-VOPs. When set to '0' it indicates that the macroblock is coded and its data is included in the bitstream.

**compression_mode**: This is a flag which is set to '0' to indicate that the texture component of a current macroblock shall be coded following the DPCM block syntax. Otherwise, the macroblock is coded as DCT data.

**dpcm_scan_order**: This is a flag that indicates the scanning order of blocks for DPCM coding. If set to value '0' the block is scanned from top line to bottom line, and from left to right. If set to value '1' the block is scanned from bottom line to top line, and from right to left.

**coda_i[j]**: This is a one-bit flag which is set to "1" to indicate that all the values in the grayscale alpha macroblock are equal to maximum_alpha_level[j] (AlphaOpaqueValue[j]). When set to "0", this flag indicates that one or more 8x8 blocks are coded according to alpha_pattern_code[j].

**alpha_compression_mode[j]**: This is a flag which is set to '0' to indicate that the alpha component 'j' of a current macroblock shall be coded following the DPCM block syntax. Otherwise, the macroblock is coded as DCT data.

**alpha_macroblock_quant[j]**: This is set to 1 to indicate that alpha_quantiser_scale_code[j] is present in the bitstream.

**alpha_quantiser_scale_code[j]**: A 5 bit unsigned integer in the range 1 to 31. The decoder shall use this value until another alpha_quantiser_scale_code[j] is encountered either in StudioSlice() or StudioMacroblock(). The value zero is forbidden.

**alpha_dpcm_scan_order[j]**: This is a flag that indicates the scanning order of a alpha block for DPCM coding. If set to value '0' the block is scanned from top line to bottom line. If set to value '1' the block is scanned from bottom line to top line.

**coda_pb[j]**: This is a VLC indicating the coding status for P alpha macroblocks. The semantics are given in the table below (Table 6-100). When this VLC indicates that the alpha macroblock is all opaque, this means that all values are set to maximum_alpha_level[j] (AlphaOpaqueValue[j]).

**Table 6-100 — coda_pb[j] codes and corresponding values**

| coda_pb[j] | Meaning |
|---|---|
| 1 | alpha residue all zero |
| 01 | alpha macroblock all opaque |
| 00 | alpha residue coded |

**cbpa[j]**: This is the coded block pattern for an inter macroblock of grayscale alpha texture data. This VLC is defined in Table B.83 - Table B.86. cbpa is followed by the alpha block data which is coded in the same way as texture block data. Note that grayscale alpha blocks in an inter macroblock with alpha all equal to 0 (alpha residual all zero) are not included in the bitstream.

alpha_pattern_code[i]: The value of this internal flag is set to 1 if the alpha block with the index value i (i=0,...,3) includes one or more DCT coefficients that are decoded through the same process as the luminance components. Otherwise, the value of this flag shall be set to 0. For an alpha intra macroblock, alpha_pattern_code[i] shall be set to 1, and for an alpha inter macroblock alpha_pattern_code[i] shall be derived from cbpa[j].

#### 6.3.13.8.1 Studio MB Binary Shape Coding

**bab_type**: This is a variable length code between 1 and 6 bits. It indicates the coding mode used for the bab. There are five bab_types as depicted in Table 6-101. The VLC tables used depend on the decoding context i.e. the bab_types of blocks already received. For I-VOPs, the context-switched VLC table of Table B.75 is used. For P-VOPs, the context-switched table of Table B.76 is used.

© ISO/IEC 2004 – All rights reserved

**Table 6-101 — List of bab_types and usage**

| bab_type | Semantic | Used in |
|---|---|---|
| 0 | MVs==0 && No Update | P- VOPs |
| 1 | MVs!=0 && No Update | P- VOPs |
| 2 | transparent | All VOP types |
| 3 | Opaque | All VOP types |
| 4 | HHC | All VOP types |

The bab_type determines what other information fields will be present for the bab shape. No further shape information is present if the bab_type = 0, 2 or 3. Opaque means that all pixels of the bab are part of the object. Transparent means that none of the bab pixels belong to the object. HHC means the Hierarchical Huffman decoding will be required to reconstruct the pixel of the bab. No_update means that motion compensation is used to copy the bab from the previous VOP's binary alpha map.

**mvs_x**: This is a variable-length code between 1 and 18 bits. It represents the horizontal element of the motion vector difference for the bab. The motion vector is in full integer precision. The VLC table is shown in Table B.77.

**mvs_y**: This is a variable-length code between 1 and 18 bits. It represents the vertical element of the motion vector difference for the bab. The motion vector is in full integer precision. If mvs_x is '0', then the VLC table of Table B.78, otherwise the VLC table of Table B.77 is used.

**inferior_symbol_macroblock**: This is a 1-bit flag indicating less frequent pixel between the opaque and transparent pixel in the macroblock. If this flag is set to "0", the transparent symbol is inferior and the opaque symbol is superior. If this flag is set to "1", the opaque symbol is inferior and the transparent is superior.

**inferior_symbol_block**: This is a 1-bit flag indicating less frequent pixel between the opaque and transparent pixel in the block. If this flag is set to "0", the transparent symbol is inferior and the opaque symbol is superior. If this flag is set to "1", the opaque symbol is inferior and the transparent is superior.

**cbbp**: This is a variable-length code between 3 to 7 bits. It indicates the existence of inferior symbol in each block. The VLC table is shown in Table B.79.

**scan_direction**: This is a 1-bit flag indicating whether the block is divided into horizontal rows or vertical columns. If this flag is set to "0", the block is divided into horizontal rows. If this flag is set to "1", the block is divided into vertical columns.

**backward_load_flag**: This is a 1-bit flag indicating whether the order of the line bit pattern is changed. If this flag is set to "0", the order of the line bit pattern isn't changed. If this flag is set to "1", the order of the line bit pattern is turned right side left in scan_direction=="0" or upside down in scan_direction=='1'.

**clp**: This is a variable-length code between 2 to 24 bits. It indicates the existence of an inferior symbol in each pixel line. The VLC table is shown in Table B.80.

**lbp**: This is a variable-length code between 2 to 23 bits. It indicates the bit pattern of an inferior symbol in the pixel line. The VLC table is shown in Table B.81.

#### 6.3.13.8.2 Studio Macroblock modes

**macroblock_type**: Variable length coded indicator which indicates the method of coding and content of the macroblock according to the Table B.42 and Table B.43, selected by vop_coding_type.

**macroblock_quant**: Derived from macroblock_type according to the Table B.42 and Table B.43. This is set to 1 to indicate that quantiser_scale_code is present in the bitstream.


© ISO/IEC 2004 – All rights reserved

**macroblock_motion_forward**: Derived from macroblock_type according to the Table B.42 and Table B.43. This flag affects the bitstream syntax and is used by the decoding process.

**macroblock_pattern**: Derived from macroblock_type according to the Table B.42 and Table B.43. This is set to 1 to indicate that coded_block_pattern() is present in the bitstream.

**macroblock_intra**: Derived from macroblock_type according to the Table B.42 and Table B.43. This flag affects the bitstream syntax and is used by the decoding process.

**frame_motion_type**: This is a two bit code indicating the macroblock prediction type, defined in Table 6-102.

If frame_pred_frame_dct is equal to 1 then frame_motion_type is omitted from the bitstream. In this case motion vector decoding and prediction formation shall be performed as if frame_motion_type had indicated "Frame-based prediction".

**Table 6-102 — Meaning of frame_motion_type**

| code | prediction type | motion_vector _count | mv_format |
|---|---|---|---|
| 00 | reserved | | |
| 01 | Field-based | 2 | field |
| 10 | Frame-based | 1 | frame |
| 11 | reserved | | |

**field_motion_type**: This is a two bit code indicating the macroblock prediction type, defined in Table 6-103.

**Table 6-103 — Meaning of field_motion_type**

| code | prediction type | motion_vector _count | mv_format |
|---|---|---|---|
| 00 | reserved | | |
| 01 | Field-based | 1 | field |
| 10 | 16x8 MC | 2 | field |
| 11 | reserved | | |

**dct_type**: This is a flag indicating whether the macroblock is frame DCT coded or field DCT coded. If this is set to '1', the macroblock is field DCT coded; otherwise, the macroblock is frame DCT coded. Boundary blocks are always coded in frame-based mode.

In the case that **dct_type** is not present in the bitstream then the value of dct_type (used in the remainder of the decoding process) shall be derived as shown in Table 6-104.

© ISO/IEC 2004 – All rights reserved

**Table 6-104 — Value of dct_type if dct_type is not in the bitstream.**

| Condition | dct_type |
|---|---|
| vop_structure == “field” | unused because there is no frame/field distinction in a field vop. |
| frame_pred_frame_dct == 1 | 0 (“frame”) |
| !(macroblock_intra \|\| macroblock_pattern) | unused - macroblock is not coded |
| macroblock is skipped | unused - macroblock is not coded |
| compression_mode == "DPCM" | 0("frame") - This is used for alpha macroblock. |
| vop_structure == "frame" && frame_pred_frame_dct == 0 && macroblock_pattern == 0 && macroblock_intra == 0 | 0("frame") - This is used for alpha macroblock. |

#### 6.3.13.8.3 Motion vectors

motion_vector_count is derived from field_motion_type or frame_motion_type as indicated in the Table 6-102 and Table 6-103.

mv_format is derived from field_motion_type or frame_motion_type as indicated in the Table 6-102 and Table 6-103. mv_format indicates if the motion vector is a field-motion vector or a frame-motion vector. mv_format is used in the syntax of the motion vectors and in the process of motion vector prediction.

**motion_vertical_field_select[r][s]**: This flag indicates which reference field shall be used to form the prediction. If motion_vertical_field_select[r][s] is zero then the top reference field shall be used, if it is one then the bottom reference field shall be used. (See 7.16.5.4.1 for the meaning of the indices; r and s.)

#### 6.3.13.8.4 Motion vector

**motion_code[r][s][t]**: This is a variable length code, as defined in 7.16.5.4.1, which is used in motion vector decoding as described in 7.16.5.4.1. (See Table 7-38 for the meaning of the indices; r, s and t.)

**motion_residual[r][s][t]** : This is an integer which is used in motion vector decoding as described in 7.16.5.4.1. (See Table 7-38 for the meaning of the indices; r, s and t.). The number of bits in the bitstream for motion_residual[r][s][t], r_size, is derived from vop_fcode[s][t] as follows;

$$r\_size = vop\_fcode[s][t] - 1$$

NOTE - The number of bits for both motion_residual[0][s][t] and motion_residual[1][s][t] is denoted by vop_fcode[s][t].

#### 6.3.13.8.5 Coded block pattern

**coded_block_pattern_420**: A variable length code that is used to derive the variable cbp according to Table B.82.

**coded_block_pattern_1** --

**coded_block_pattern_2**: For 4:2:2 and 4:4:4 data the coded block pattern is extended by the addition of either a two bit or six bit fixed length code, coded_block_pattern_1 or coded_block_pattern_2. Then the pattern_code[i] is derived using the following:


© ISO/IEC 2004 – All rights reserved

```
for (i=0; i<12; i++) {
    if (macroblock_intra)
        pattern_code[i] = 1;
    else
        pattern_code[i] = 0;
}
if (macroblock_pattern) {
    for (i=0; i<6; i++)
        if ( cbp & (1<<(5-i)) ) pattern_code[i] = 1;
    if (chroma_format == "4:2:2")
        for (i=6; i<8; i++)
            if ( coded_block_pattern_1 & (1<<(7-i)) ) pattern_code[i] = 1;
    if (chroma_format == "4:4:4")
        for (i=6; i<12; i++)
            if ( coded_block_pattern_2 & (1<<(11-i)) ) pattern_code[i] = 1;
}
```

If pattern_code[i] is equal to 1, i=0 to (block_count-1), then the block number i defined in Figure 6-8, Figure 6-9 and Figure 6-10 is contained in this macroblock.

The number "block_count" which determines the number of blocks in the macroblock is derived from the chrominance format as shown in Table 6-105.

**Table 6-105 — block_count as a function of chroma_format**

| chroma_format | block_count |
|---|---|
| 4:2:0 | 6 |
| 4:2:2 | 8 |
| 4:4:4 | 12 |

If the block number i is transparent, transparent_block(i)=1, the velue of pattern_code[i] does not affect the decoding process.

If the block number i is transparent, transparent_block(i)=1, the velue of pattern_code[i] does not affect the decoding process.

#### 6.3.13.9 Studio Block

**dct_dc_size_luminance**: This is a variable length code as defined in Table B.44 that is used to derive the value of the differential DC coefficients of luminance values in blocks in intra macroblocks. This value categorizes the coefficients according to their size.

© ISO/IEC 2004 – All rights reserved

**dct_dc_differential**: This is a variable length code as defined in Table B.46 that is used to derive the value of the differential DC coefficients in blocks in intra macroblocks. After identifying the category of the DC coefficient in size from dct_dc_size_luminance or dct_dc_size_chrominance, this value denotes which actual difference in that category occurred.

**dct_dc_size_chrominance**: This is a variable length code as defined in Table B.45 that is used to derive the value of the differential DC coefficients of chrominance values in blocks in intra macroblocks. This value categorizes the coefficients according to their size.

#### 6.3.13.9.1 Studio Alpha Block

alpha_pattern_code[i]: The value of this internal flag is set to 1 if the alpha block with the index value i indicates one or more DCT coefficients that are decoded in the same way as the luminance component. Otherwise the value of this flag is set to 0.

**dct_dc_size_alpha**: This is a variable length code as defined in Table B.44 that is used to derive the value of the differential DC coefficients of alpha values in blocks in intra macroblocks. This value categorizes the coefficients according to their size.

The other semantics of StudioAlphaBlock() are described in clause 7.

### 6.3.13.10 Studio DPCM Block

**block_mean**: This is an unsigned integer that indicates the average value of pixels within a DPCM coded block. This mean value is also used for efficient prediction of the sign of residuals in DPCM coding. The value 0 is forbidden.

**rice_parameter**: This is an unsigned integer that indicates the length of the **rice_suffix_code** field. The value 0 is forbidden. The value 15 shall be interpreted as 0. The values 12 - 14 are also forbidden.

**rice_prefix_code**: This is a variable length code that represents the most significant bits of a DPCM residual. This code may also represent an escape sequence in which case a DPCM residual is coded by **dpcm_residual** instead of a combination of **rice_prefix_code** and **rice_suffix_code**.

**dpcm_residual**: This is an unsigned integer that indicates the value of a DPCM residual.

**rice_suffix_code**: This is an unsigned integer that represents the least significant bits of a DPCM residual.

**Table 6-106 — Variable length codes for rice_prefix_code**

| Variable length code | rice_prefix_code |
|---|---|
| 1 | 0 |
| 01 | 1 |
| 001 | 2 |
| 0001 | 3 |
| 0000 1 | 4 |
| 0000 01 | 5 |
| 0000 001 | 6 |
| 0000 0001 | 7 |
| 0000 0000 1 | 8 |
| 0000 0000 01 | 9 |
| 0000 0000 001 | 10 |
| 0000 0000 0001 | escape |


© ISO/IEC 2004 – All rights reserved

### 6.3.14 FGS Video Object

#### 6.3.14.1 FGS Video Object Plane

**fgs_vop_start_code**: This is the bit string '000001B9' in hexadecimal. It marks the start of an fgs vop.

**fgs_vop_coding_type**: The fgs_vop_coding_type identifies whether an fgs vop is an FGS coding type (I), predictive-coded FGS coding type (P), or bidirectionally predictive-coded FGS coding type (B). The meaning of fgs_vop_coding_type is defined in Table 6-107.

**Table 6-107 — Meaning of fgs_vop_coding_type**

| fgs_vop_coding_type | coding method |
|---|---|
| 00 | FGS (I) |
| 01 | predictive-coded FGS (P) |
| 10 | bidirectionally-predictive-coded FGS (B) |
| 11 | reserved |

**fgs_vop_max_level_y**: This is an unsigned integer which specifies the number of vop-bps in which the Y component has at least one non-zero block-bp in the first one of them.

**fgs_vop_max_level_u**: This is an unsigned integer which specifies the number of vop-bps in which the U component has at least one non-zero block-bp in the first one of them.

**fgs_vop_max_level_v**: This is an unsigned integer which specifies the number of vop-bps in which the V component has at least one non-zero block-bp in the first one of them.

**fgs_vop_number_of_vop_bp_coded**: This is an unsigned integer which specifies the number of vop-bps coded into the bitstream.

**fgs_vop_mc_bit_plane_used**: This parameter is reserved for future use. The value of this parameter shall be 0.

**fgs_vop_selective_enhancement_enable**: This flag shall be 1 when selective enhancement is enabled and it shall be 0 otherwise.

**fgs_ref_select_code**: This is a 2-bit unsigned integer which specifies prediction reference choices for "P" and "B" fgs vops with respect to decoded reference layer identified by ref_layer_id. The meaning of allowed values is specified in Table 6-108 and Table 6-109.

**Table 6-108 — Prediction reference choices for "P" fgs vops**

| fgs_ref_select_code | forward prediction reference |
|---|---|
| 00 | Reserved |
| 01 | Most recently VOP in display order belonging to the reference layer. |
| 10 | Next VOP in display order belonging to the reference layer. |
| 11 | Reserved |

© ISO/IEC 2004 – All rights reserved

**Table 6-109 — Prediction reference choices for "B" fgs vops**

| fgs_ref_select_code | forward temporal reference | backward temporal reference |
|---|---|---|
| 00 | Reserved | |
| 01 | Reserved | |
| 10 | Reserved | |
| 11 | Most recently VOP in display order belonging to the reference layer. | Next VOP in display order belonging to the reference layer. |

**fgs_bp_start_code**: This is a string of 32 bits. The first 27 bits are '0000 0000 0000 0000 0000 0001 010' in binary and the last 5 bits represent one of the values in the range of '00000' to '11111' in binary. The fgs_bp_start_code marks a new vop-bp.

**fgs_vop_bp_id**: This is given by the last 5 bits of the fgs_bp_start_code. The fgs_vop_bp_id uniquely identifies a vop-bp. The value of fgs_vop_bp_id is "0" for the most significant vop-bp and increments by 1 for each successively lower vop-bp.

**fgs_resync_marker**: This is a binary string of 22 zeros followed by a one '000 0000 0000 0000 0000 0001'. It is only present when fgs_resync_marker_disable flag is set to '0'.

**fgs_macroblock_number**: This is a variable length code with length between 1 and 14 bits. It identifies the fgs macroblock number of the following fgs macroblock. The actual length of the code depends on the total number of fgs macroblocks in a vop-bp, calculated according to Table 6-110. The code itself is a binary representation of the fgs macroblock number.

**Table 6-110 — Length of fgs_macroblock_number code**

| length of fgs_macroblock_number code | ((video_object_layer_width+15)/16) * ((video_object_layer_height+15)/16) |
|---|---|
| 1 | 1-2 |
| 2 | 3-4 |
| 3 | 5-8 |
| 4 | 9-16 |
| 5 | 17-32 |
| 6 | 33-64 |
| 7 | 65-128 |
| 8 | 129-256 |
| 9 | 257-512 |
| 10 | 513-1024 |
| 11 | 1025-2048 |
| 12 | 2049-4096 |
| 13 | 4097-8192 |
| 14 | 8193-16384 |


© ISO/IEC 2004 – All rights reserved

#### 6.3.14.2 FGS Motion Macroblock

**fgs_not_coded**: This is a 1-bit flag that signals if an FGS motion macroblock is coded or not in a "P" fgs vop. When set to'1' it indicates that an FGS motion macroblock is not coded and no further data is included in the bitstream for this FGS motion macroblock; decoder shall treat this FGS motion macroblock as motion vector equal to zero. When set to '0' it indicates that the FGS motion macroblock is coded and its data is included in the bitstream.

**fgs_p_mb_type**: This is a 1-bit flag that signals the type of motion vector coding in "P" fgs vop. When set to "0", it indicates that the one-motion-vector mode is used in this FGS motion macroblock. When set to "1", it indicates that the four-motion-vector mode is used in this FGS motion macroblock.

**fgs_modb**: This is a 1-bit flag that signals if an FGS motion macroblock is coded or not in a "B" fgs vop. When set to'1' it indicates that an FGS motion macroblock is not coded and no further data is included in the bitstream for this FGS motion macroblock; decoder shall treat this FGS motion macroblock as motion vector equal to zero. When set to '0' it indicates that the FGS motion macroblock is coded and its data is included in the bitstream.

**fgs_b_mb_type**: This variable length code is present only in coded FGS motion macroblocks of "B" fgs vops. The codes for fgs_b_mb_type are shown in Table 6-111.

**Table 6-111 — VLC table for fgs_b_mb_type**

| Code | fgs_b_mb_type |
|---|---|
| 1 | direct |
| 01 | interpolate |
| 001 | backward |
| 0001 | forward |

#### 6.3.14.3 FGS Motion Interlaced Information

**fgs_field_prediction**: This is a 1-bit flag indicating whether the FGS motion macroblock is field predicted or frame predicted. This flag is set to '1' when the FGS motion macroblock is predicted using field motion vectors. If it is set to '0' then frame prediction (16x16 or 8x8) will be used. This flag is only present in the bitstream if the interlaced flag is set to "1", and either the fgs_p_mb_type is "0" in a "P" fgs vop or the FGS motion macroblock is in a "B" fgs vop.

**fgs_forward_top_field_reference**: This is a 1-bit flag which indicates the reference field for the forward motion compensation of the top field. When this flag is set to '0', the top field is used as the reference field. If it is set to '1' then the bottom field will be used as the reference field. This flag is only present in the bitstream if the fgs_field_prediction flag is set to "1" and the FGS motion macroblock is not backward predicted.

**fgs_forward_bottom_field_reference**: This is a 1-bit flag which indicates the reference field for the forward motion compensation of the bottom field. When this flag is set to '0', the top field is used as the reference field. If it is set to '1' then the bottom field will be used as the reference field. This flag is only present in the bitstream if the fgs_field_prediction flag is set to "1" and the FGS motion macroblock is not backward predicted.

**fgs_backward_top_field_reference**: This is a 1-bit flag which indicates the reference field for the backward motion compensation of the top field. When this flag is set to '0', the top field is used as the reference field. If it is set to '1' then the bottom field will be used as the reference field. This flag is only present in the bitstream if the fgs_field_prediction flag is set to "1" and the FGS motion macroblock is not forward predicted.

**fgs_backward_bottom_field_reference**: This is a 1-bit flag which indicates the reference field for the backward motion compensation of the bottom field. When this flag is set to '0', the top field is used as the reference field. If it is set to '1' then the bottom field will be used as the reference field. This flag is only present in the bitstream if the fgs_field_prediction flag is set to "1" and the FGS motion macroblock is not forward predicted.

© ISO/IEC 2004 – All rights reserved

#### 6.3.14.4 FGS Motion Vector

Identical to subclause 6.3.6.2.

#### 6.3.14.5 FGS Macroblock

**fgs_cbp**: This is a variable length code with length between 1 and 9 bits. It specifies the coded bit pattern of fgs_msb_not_reached in the fgs macroblock in the first two vop-bps.

For the first vop-bp with Y, U, and V components, Table 6-112 summarizes the code for fgs_cbp: (note: x denotes either 0 or 1)

**Table 6-112 — Coding table for fgs_cbp in the first vop-bp with Y, U, and V components**

| Code | Pattern of fgs_msb_not_reached yyyy,uv | Meaning |
|---|---|---|
| 1 | 1111,11 | 6 all-zero block-bps |
| 01xxxx | xxxx,11 | U and V block-bps are all-zero block-bps, but at least one of the 4 Y block-bps is not an all-zero block-bp |
| 00xxxx01 | xxxx,00 | both U and V block-bps are not all-zero block-bps |
| 00xxxx10 | xxxx,01 | U block-bp is not an all-zero block-bp and V block-bp is an all-zero block-bp |
| 00xxxx11 | xxxx,10 | U block-bp is an all-zero block-bp and V block-bp is not an all-zero block-bp |

Within a vop-bp, one of the three color components may be absent. If one of the U and V components is absent, the above coding scheme is followed except that the absent color component is not coded. Table 6-113 specifies this case:

**Table 6-113 — Coding table for fgs_cbp in the first vop-bp when one of the U and V components is absent**

| Code | Pattern of fgs_msb_not_reached yyyy,u/v | Meaning |
|---|---|---|
| 1 | 1111,1 | 5 all-zero block-bps |
| 0xxxxx (000001 –011111) | xxxx,x (0000,0 – 1111,0) | not all 5 blocks are all-zero block-bps |

To avoid start code emulation, the 5-bit pattern of fgs_msb_not_reached forms a 5-bit integer and the integer plus 1 in binary is 'xxxxx' in the code.

If both U and V components are absent in vop-bp, Table 6-114 specifies the code:


© ISO/IEC 2004 – All rights reserved

**Table 6-114 — Coding table for fgs_cbp in the first vop-bp when both U and V components are absent**

| Code | Pattern of fgs_msb_not_reached yyyy | Meaning |
|---|---|---|
| 1 | 1111 | 4 all-zero block-bps |
| 0100<br>0101<br>0110<br>0111 | 0111<br>1011<br>1101<br>1110 | 3 all-zero block-bps |
| 001000<br>001001<br>001010<br>001011<br>001100<br>001101 | 0011<br>0101<br>0110<br>1001<br>1010<br>1100 | 2 all-zero block-bps |
| 000100<br>000101<br>000110<br>000111 | 1000<br>0100<br>0010<br>0001 | 1 all-zero block-bp |
| 00001 | 0000 | no all-zero block-bp |

In the following unlikely cases:

- Y component is absent,
- the Y and U components are absent,
- the Y and V components are absent,

the pattern of fgs_msb_not_reached is put into the bitstream without VLC coding.

For the second vop-bp, the pattern of fgs_msb_not_reached is put into the bitstream without VLC coding except for the fgs macroblocks that satisfy the following conditions:

- Y, U, and V components of the second vop-bp are all present;
- The pattern of fgs_msb_not_reached for the fgs macroblock in the first vop-bp with the same fgs macroblock number is all 1's.

The coding method for this case is specified in Table 6-115:

**Table 6-115 — Coding table for fgs_cbp in the second vop-bp**

| Code | Pattern of fgs_msb_not_reached yyyy,uv | Meaning |
|---|---|---|
| 011 | 1111,11 | 6 all-zero block-bps |
| 010 | 0000,11 | 4 Y block-bps are not all-zero block-bps and both U and V block-bps are all-zero block-bps |
| 1xxxx | xxxx,11 | at least one Y block-bp is all-zero block-bp but not all 4 Y block-bps are all-zero block-bps, and both U and V block-bps are all-zero block-bps |

© ISO/IEC 2004 – All rights reserved

| | | |
|---|---|---|
| 10000 | 0000,00 | none of the 6 block-bps are all-zero block-bps |
| 11111 | 1111,00 | 4 Y block-bps are all-zero block-bps and both U and V block-bps are not all-zero block-bps |
| 00100 | 0000,01 | 4 Y block-bps and the U block-bp are not all-zero block-bps, but the V block-bp is an all-zero block-bp |
| 00101 | 0000,10 | 4 Y block-bps and the V block-bp are not all-zero block-bps, but the U block-bp is an all-zero block-bp |
| 00110 | 1111,01 | 4 Y block-bps and the V block-bp are all-zero block-bps, but the U block-bp is not an all-zero block-bp |
| 00111 | 1111,10 | 4 Y block-bps and the U block-bp are all-zero block-bps, but the V block-bp is not an all-zero block-bp |
| 000xxxxxx | xxxx,xx | All other cases |

**fgs_shifted_bits**: This is a variable length code with length between 1 and 5 bits as shown in Table 6-116. When selective enhancement is used, all DCT coefficients in an fgs macroblock are left-shifted by fgs_shifted_bits.

**Table 6-116 — VLC table of fgs_shifted_bits**

| Number of bits | VLC code |
|---|---|
| 0 | 1 |
| 1 | 01 |
| 2 | 001 |
| 3 | 0001 |
| 4 | 00001 |

**fgs_dct_type**: This is a 1-bit flag indicating whether this fgs macroblock is frame DCT coded or field DCT coded. If this flag is set to "1", the fgs macroblock is field DCT coded; otherwise, the fgs macroblock is frame DCT coded. This flag is only present in the bitstream if the interlaced flag is set to "1" in the fgs vop header. This flag is only present once in the first non-zero fgs macroblock with the same fgs macroblock number.

#### 6.3.14.6 FGS Block

**fgs_msb_not_reached**: This flag is 1 if the most significant block-bp in an 8x8 DCT block is not reached yet. In other words, this indicates an all-zero block-bp before the most significant block-bp. This flag is 0 otherwise.

**fgs_run_eop_code**: The VLC code for a (RUN,EOP) symbol as described in Clause 7.

**fgs_sign_bit**: This bit indicates the sign of the DCT coefficient with value 0 indicating positive and value 1 indicating negative.

## 7 The visual decoding process

This clause specifies the decoding process that the decoder shall perform to recover visual data from the coded bitstream. As shown in Figure 7-1, the visual decoding process includes several decoding processes such as shape-motion-texture decoding, still texture decoding, mesh decoding, and face decoding processes. After decoding the coded bitstream, it is then sent to the compositor to integrate various visual objects.


© ISO/IEC 2004 – All rights reserved

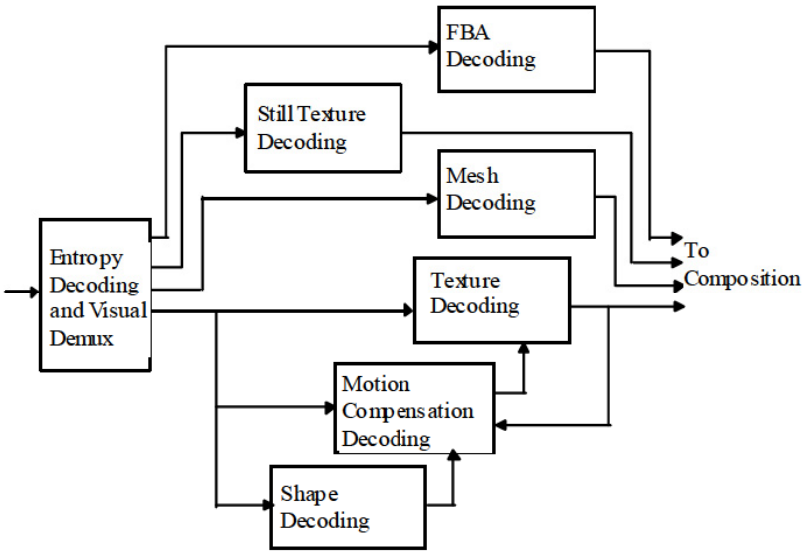



**Figure 7-1 — A high level view of basic visual decoding; specialized decoding such as scalable, sprite and error resilient decoding are not shown**

In subclauses 7.1 through 7.9 the VOP decoding process is specified in which shape, motion, texture decoding processes are the major contents. The still texture object decoding is described in subclauses 7.10. Subclause 7.11 includes the mesh decoding process, and subclause 7.12 features the face object decoding process. The output of the decoding process is explained in subclause 7.13. The video object decoding for the studio profile is specified in subclause 7.16.

### 7.1 Video decoding process

This subclause specifies the decoding process that a decoder shall perform to recover VOP data from the coded video bitstream.

With the exception of the Inverse Discrete Cosine Transform (IDCT) the decoding process is defined such that all decoders shall produce numerically identical results. Any decoding process that produces identical results to the process described here, by definition, complies with this part of ISO/IEC 14496.

The IDCT is defined statistically such that different implementations for this function are allowed. The IDCT specification is given in Annex A.

Figure 7-2 is a diagram of the Video Decoding Process without any scalability feature. The diagram is simplified for clarity. The same decoding scheme is applied when decoding all the VOPs of a given session

NOTE Throughout this part of ISO/IEC 14496 two dimensional arrays are represented as *name*[q][p] where 'q' is the index in the vertical dimension and 'p' the index in the horizontal dimension.

© ISO/IEC 2004 – All rights reserved

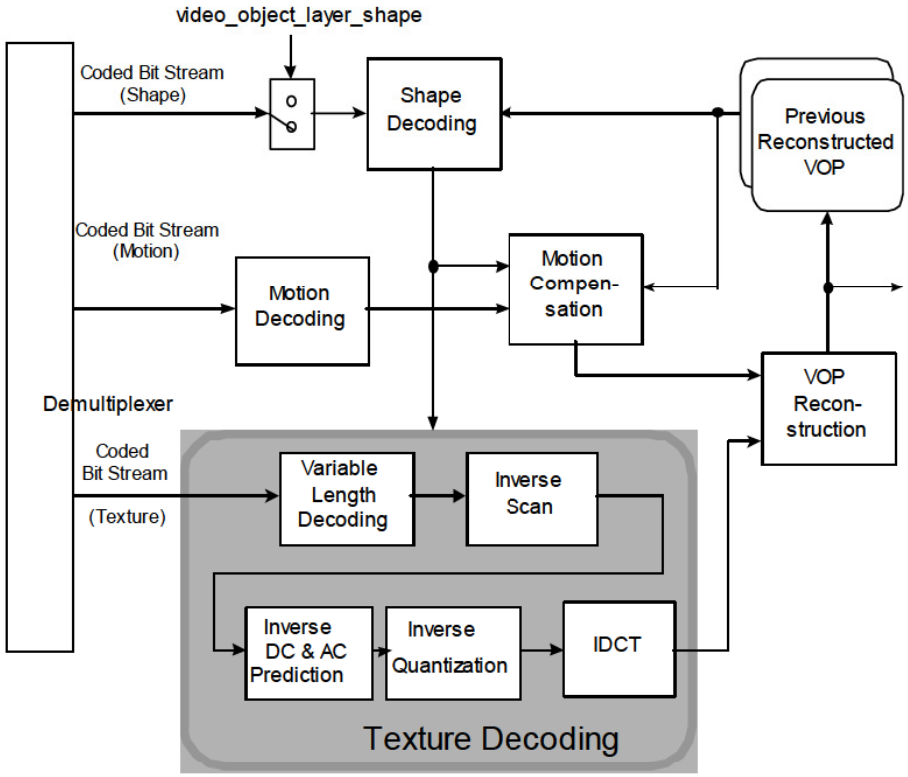



**Figure 7-2 — Simplified Video Decoding Process**

The decoder is mainly composed of three parts: shape decoder, motion decoder and texture decoder. The reconstructed VOP is obtained by combining the decoded shape, texture and motion information.

### 7.2 Higher syntactic structures

The various parameters and flags in the bitstream for VideoObjectLayer(), Group_of_VideoObjectPlane(), VideoObjectPlane(), video_plane_with_short_header(), macroblock() and block(), as well as other syntactic structures related to them shall be interpreted as specified earlier. Many of these parameters and flags affect the decoding process. Once all the macroblocks in a given VOP have been processed, the entire VOP will have been reconstructed. In case the bitstream being decoded contains B-VOPs, reordering of VOPs may be needed as specified in subclause 6.1.3.7.

### 7.3 VOP reconstruction

The luminance and chrominance values of a VOP from the decoded texture and motion information are reconstructed as follows:

1. In case of intra macroblocks, the luminance and chrominance values f[y][x] from the decoded texture data form the luminance and chrominance values of the VOP: d[y][x] = f[y][x].

2. In case of inter macroblocks, first the prediction values p[y][x] are calculated using the decoded motion vector information and the texture information of the respective reference VOPs. Then, the decoded texture data f[y][x] is added to the prediction values, resulting in the final luminance and chrominance values of the VOP: d[y][x] = p[y][x] + f[y][x]


© ISO/IEC 2004 – All rights reserved

3. Finally, the calculated luminance and chrominance values of the reconstructed VOP are saturated so that

$$d[y][x] = \begin{cases} 2^{bits\_per\_pixel} - 1; & d[y][x] > 2^{bits\_per\_pixel} - 1 \\ d[y][x]; & 0 \le d[y][x] \le 2^{bits\_per\_pixel} - 1 \\ 0; & d[y][x] < 0 \end{cases}$$

### 7.4 Texture decoding

This subclause describes the process used to decode the texture information of a VOP. The process of video texture decoding is given in Figure 7-3.

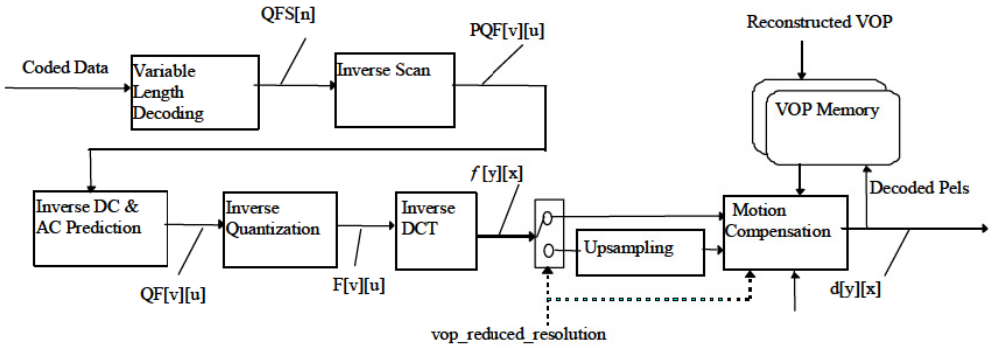



**Figure 7-3 — Video Texture Decoding Process**

NOTE If flag 'sadct_disable' in the header of VideoObjectLayer() is set to '0', some standard texture decoding techniques from version 1 are modified. In 8x8-blocks with at least one transparent and opaque pel, inverse DCT (subclause 7.4.5) is replaced by inverse shape adaptive DCT (SA-DCT). The number of SA-DCT coefficients in such a block is equal to the number of opaque pels. Only after BAB decoding, it is known to which 8x8-blocks inverse SA-DCT is applied. The usage of inverse SA-DCT in such blocks implies modifications in the processes "Inverse Scan" (subclause 7.3.2), "Inverse AC Prediction" (subclause 7.3.3.3) and "Inverse DCT" (subclause 7.3.5) whereas the processing in the remaining blocks of Figure 7-3 keeps unmodified. The processing of inverse SA-DCT additionally depends on 'mb_type'; i.e.: different types of SA-DCT processing are used in intra- and inter-coded blocks (see subclause 7.3.5).

#### 7.4.1 Variable length decoding

This subclause explains the decoding process. Subclause 7.4.1.1 specifies the process used for the DC coefficients (n=0) in an intra coded block. (n is the index of the coefficient in the appropriate zigzag scan order). Subclause 7.4.1.2 specifies the decoding process for all other coefficients; AC coefficients ( $n \neq 0$ ) and DC coefficients in non-intra coded blocks.

##### 7.4.1.1 DC coefficients decoding in intra blocks

Differential DC coefficients in blocks in intra macroblocks are encoded as variable length code denoting dct_dc_size as defined in Table B.13 and Table B.14 in Annex B, and a fixed length code dct_dc_differential (Table B.15). The dct_dc_size categorizes the DC coefficients according to their "size". For each category additional bits are appended to the dct_dc_size code to uniquely identify which difference in that category actually occurred (Table B.15). This is done by appending a fixed length code, dct_dc_differential, of dct_dc_size bits. The final value of the decoded DC coefficient is the sum of this differential DC value and the predicted value.

When short_video_header is 1, the DC coefficient of an intra block is not coded differentially. It is instead transmitted as a fixed length unsigned integer code of size 8 bits, unless this integer has the value 255. The values 0 and 128 shall not be used – they are reserved. If the integer value is 255, this is interpreted as a signaled value of 128.

© ISO/IEC 2004 – All rights reserved

#### 7.4.1.2 Other coefficients

The AC coefficients are obtained by decoding the variable length codes to produce EVENTs. An EVENT is a combination of a last non-zero coefficient indication (LAST; "0": there are more nonzero coefficients in this block, "1": this is the last nonzero coefficient in this block), the number of successive zeros preceding the coded coefficient (RUN), and the non-zero value of the coded coefficient (LEVEL).

When short_video_header is 1, the most commonly occurring EVENTS are coded with the variable length codes given in Table B.17 (for all coefficients other than intra DC whether in intra or inter blocks). The last bit "s" denotes the sign of level, "0" for positive and "1" for negative.

When short_video_header is 0, the variable length code table is different for intra blocks and inter blocks. The most commonly occurring EVENTs for the luminance and chrominance components of intra blocks in this case are decoded by referring to the intra columns of Table B.23 when reversible_vlc is set to '1' in I-, P-, or S(GMC)-VOPs, and by referring to Table B.16, otherwise. The most commonly occurring EVENTs for the luminance and chrominance components of inter blocks in this case are decoded by referring to the inter columns of Table B.23 when reversible_vlc is set to '1' in I-, P-, or S(GMC)-VOPs, and by referring to Table B-17, otherwise. The last bit "s" denotes the sign of level, "0" for positive and "1" for negative. The combinations of (LAST, RUN, LEVEL) not represented in these tables are decoded as described in subclause 7.4.1.3.

#### 7.4.1.3 Escape code

The EVENTS that have no variable length code to represent them are coded using an Escape Coding method. The escape codes of DCT coefficients are encoded in five modes. The first three of these modes are used when short_video_header is 0 and in the case that the reversible VLC tables are not used, and the fourth is used when short_video_header is 1. In the case that the reversible VLC tables are used, the fifth escape coding method as in Table B.23 is used. Use of escape sequence of the reversible VLC (Table B.24 and Table B.25) for encoding the combinations listed in Table B.23 is prohibited. Their decoding process is specified below.

Type 1: ESC is followed by "0", and the code following ESC + "0" is decoded as a variable length code using the Tcoef VLC codes given in Table B.16 and Table B.17, but the values of LEVEL are modified following decoding to give the restored value $LEVEL^S$, as follows:

$$LEVEL^S = sign(LEVEL^+) \times [\, abs(\, LEVEL^+) + LMAX \,]$$

where $LEVEL^+$ is the value after variable length decoding and LMAX is obtained from Table B.19 and Table B.20 as a function of the decoded values of RUN and LAST.

Type 2: ESC is followed by "10", and the code following ESC + "10" is decoded as a variable length code using the Tcoef VLC codes given in Table B.16 and Table B.17, but the values of RUN are modified following decoding to give the restored value $RUN^S$, as follows:

$$RUN^S = RUN^+ + (RMAX + 1)$$

where $RUN^+$ is the value after variable length decoding. RMAX is obtained from Table B.21 and Table B.22 as a function of the decoded values of LEVEL and LAST.

Type 3: ESC is followed by "11", and the code following ESC + "11" is decoded as fixed length codes. This type of escape codes are represented by 1-bit LAST, 6-bit RUN and 12-bit LEVEL. A marker bit is inserted before and after the 12-bit-LEVEL in order to avoid the resync_marker emulation. Use of this escape sequence for encoding the combinations listed in Table B.16 and Table B.17 is prohibited. The codes for RUN and LEVEL are given in Table B.18 a and b.

Type 4: The fourth type of escape code is used if and only if short_video_header is 1. In this case, the 15 bits following ESC are decoded as fixed length codes represented by 1-bit LAST, 6-bit RUN and 8-bit LEVEL. The values 0000 0000 and 1000 000 for LEVEL are not used (they are reserved). The codes for RUN and LEVEL are given in Table B.18 a and c.


© ISO/IEC 2004 – All rights reserved

#### 7.4.1.4 Intra DC coefficient decoding for the case of switched vlc encoding

At the VOP layer, using quantiser scale value as the threshold, a 3 bit code (intra_dc_vlc_thr) allows switching between 2 VLCs (DC Intra VLC and AC Intra VLC) when decoding DC coefficients of Intra macroblocks, see Table 6.25.

NOTE When the intra AC VLC is turned on, Intra DC coefficients are not handled separately any more, but treated the same as all other coefficients. That means that a zero Intra DC coefficient will not be coded but will only increase the run for the following AC coefficients. The definitions of mcbpc and cbpy in subclause 6.3.6 are changed accordingly.

### 7.4.2 Inverse scan

This subclause specifies the way in which the one dimensional data, QFS[n] is converted into a two-dimensional array of coefficients denoted by PQF[v][u] where u and v both lie in the range of 0 to 7. Let the data at the output of the variable length decoder be denoted by QFS[n] where n is in the range of 0 to 63. Three scan patterns are defined as shown in Figure 7-4. The scan that shall be used is determined by the following method. For intra blocks, if acpred_flag=0, zigzag scan is selected for all blocks in a macroblock. Otherwise, DC prediction direction is used to select a scan on block basis. For instance, if the DC prediction refers to the horizontally adjacent block, alternate-vertical scan is selected for the current block. Otherwise (for DC prediction referring to vertically adjacent block), alternate-horizontal scan is used for the current block. For all other blocks, the 8x8 blocks of transform coefficients are scanned in the "zigzag" scanning direction.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 10 | 11 | 12 | 13 |
| 4 | 5 | 8 | 9 | 17 | 16 | 15 | 14 |
| 6 | 7 | 19 | 18 | 26 | 27 | 28 | 29 |
| 20 | 21 | 24 | 25 | 30 | 31 | 32 | 33 |
| 22 | 23 | 34 | 35 | 42 | 43 | 44 | 45 |
| 36 | 37 | 40 | 41 | 46 | 47 | 48 | 49 |
| 38 | 39 | 50 | 51 | 56 | 57 | 58 | 59 |
| 52 | 53 | 54 | 55 | 60 | 61 | 62 | 63 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 4 | 6 | 20 | 22 | 36 | 38 | 52 |
| 1 | 5 | 7 | 21 | 23 | 37 | 39 | 53 |
| 2 | 8 | 19 | 24 | 34 | 40 | 50 | 54 |
| 3 | 9 | 18 | 25 | 35 | 41 | 51 | 55 |
| 10 | 17 | 26 | 30 | 42 | 46 | 56 | 60 |
| 11 | 16 | 27 | 31 | 43 | 47 | 57 | 61 |
| 12 | 15 | 28 | 32 | 44 | 48 | 58 | 62 |
| 13 | 14 | 29 | 33 | 45 | 49 | 59 | 63 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 5 | 6 | 14 | 15 | 27 | 28 |
| 2 | 4 | 7 | 13 | 16 | 26 | 29 | 42 |
| 3 | 8 | 12 | 17 | 25 | 30 | 41 | 43 |
| 9 | 11 | 18 | 24 | 31 | 40 | 44 | 53 |
| 10 | 19 | 23 | 32 | 39 | 45 | 52 | 54 |
| 20 | 22 | 33 | 38 | 46 | 51 | 55 | 60 |
| 21 | 34 | 37 | 47 | 50 | 56 | 59 | 61 |
| 35 | 36 | 48 | 49 | 57 | 58 | 62 | 63 |

**Figure 7-4 — (a) Alternate-Horizontal scan (b) Alternate-Vertical scan (c) Zigzag scan**

The inverse scan pattern from Figure 7-4 represent a conversion from the one-dimensional array 'QFS[n]' to two-dimensional array 'PQF[v][u]. This conversion can be expressed as follows where the arrays 'inv_scan_u[scan_type][n]' and 'inv_scan_v[scan_type][n]' denote the u- and v-components of the selected inverse scan path from Figure 7-4:

```
for (n=0; n<64; n++)
   PQF[inv_scan_v[scan_type][n]][inv_scan_u[scan_type][n]] = QFS[n];
```

If 'sadct_disable' is set to '0', this conversion is modified for those 8x8-blocks in which the number of opaque pels is less than 64:

```
if ((video_object_layer!="rectangular") && (sadct_disable==0) &&
  (opaque_pels<64)) {
   coeff_count=0;
   for (n=0; n<64; n++){
      PQF[inv_scan_v[scan_type][n]][inv_scan_u[scan_type][n]]=0;
      if (coeff_width[inv_scan_v[scan_type][n]] > inv_scan_u[scan_type][n]) {
         PQF[inv_scan_v[scan_type][n]][inv_scan_u[scan_type][n]]=
           QFS[coeff_count];
         coeff_count++;
      }
   }
}
```

© ISO/IEC 2004 – All rights reserved

NOTES -

1 After finalization of modified inverse scan the maximal number of non-zero SA-DCT coefficients (see parameter 'coeff_count') equals to 'opaque_pels'. It is absolutely necessary to keep this constraint by setting PQF[v][u] to zero at positions where SA-DCT coefficients are not defined. A non-zero value at these undefined positions of PQF[v][u] would confuse subsequent AC prediction.

2 The permitted values of elements in arrays 'inv_scan_u[scan_type][n]', 'inv_scan_v [scan_type][n]' and 'coeff_width[v]' lie in a range from 0 to 7.

3 Apart from the usage of inverse SA-DCT in subclause 7.3.5 and slight modifications in subclause 7.3.3.3 (inverse AC predicion) the further processing of 'PQF[v][u]' is independent of the setting of flag 'sadct_disable' and keeps unmodified for 'sadct_disable=0'.

The above definition of the modified inverse scan requires an auxiliary array 'coeff_ width[v]' which has to be derived from the decoded BAB. It indicates the number of SA-DCT coefficients which are available in each particular row of PQF[v][u]. See Annex A (subclause A.3.1) for details.

### 7.4.3 DC and AC prediction for intra macroblocks

This subclause specifies the prediction process for decoding of coefficients. This prediction process is only carried out for intra-macroblocks (I-MBs) and when short_video_header is "0". When short_video_header is "1" or the macroblock is not an I-MB, this prediction process is not performed.

#### 7.4.3.1 DC and AC Prediction Direction

This adaptive selection of the DC and AC prediction direction is based on comparison of the horizontal and vertical DC differences around the block to be decoded. Figure 7-5 shows the three blocks surrounding the block to be decoded. Block 'X', 'A', 'B' and 'C' respectively refer to the current block, the left block, the above-left block, and the block immediately above, as shown.

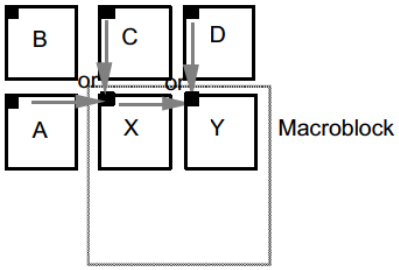



**Figure 7-5 — Previous neighboring blocks used in DC prediction**

The inverse quantised DC values of the previously decoded blocks, F[0][0], are used to determine the direction of the DC and AC prediction as follows.

if ( $|F_A[0][0] - F_D[0][0]| < |F_D[0][0] - F_C[0][0]|$)

predict from block C

else

predict from block A

If any of the blocks A, B or C are outside of the VOP boundary, or the video packet boundary, or they do not belong to an intra coded macroblock, their F[0][0] values are assumed to take a value of $2^{(bits\_per\_pixel+2)}$ and are used to compute the prediction values.


© ISO/IEC 2004 – All rights reserved

#### 7.4.3.2 Adaptive DC Coefficient Prediction

The adaptive DC prediction method involves selection of either the F[0][0] value of immediately previous block or that of the block immediately above it (in the previous row of blocks) depending on the prediction direction determined above.

if (predict from block C)

$QF_X[0][0] = PQF_X[0][0] + F_C[0][0]$ // dc_scaler

else

$QF_X[0][0] = PQF_X[0][0] + F_A[0][0]$ // dc_scaler

dc_scalar is defined in Table 7-1. This process is independently repeated for every block of a macroblock using the appropriate immediately horizontally adjacent block 'A' and immediately vertically adjacent block 'C'.

DC predictions are performed similarly for the luminance and each of the two chrominance components.

#### 7.4.3.3 Adaptive AC coefficient prediction

This process is used when ac_pred_flag = '1', which indicates that AC prediction is performed when decoding the coefficients.

Either coefficients from the first row or the first column of a previous coded block are used to predict the co-sited coefficients of the current block. On a block basis, the direction (from among horizontal and vertical directions) for DC coefficient prediction is also used to select the direction for AC coefficients prediction. The AC coefficients prediction is illustrated in Figure 7-6.

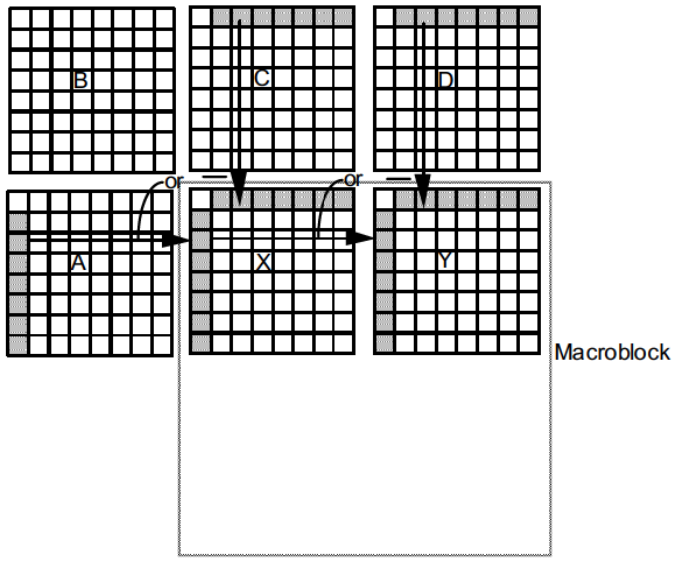



**Figure 7-6 — Previous neighboring blocks and coefficients used in AC prediction**

To compensate for differences in the quantisation of previous horizontally adjacent or vertically adjacent blocks used in AC prediction of the current block, scaling of prediction coefficients becomes necessary. Thus the prediction is modified so that the predictor is scaled by the ratio of the current quantiser scale and the quantiser scale of the predictor block. The definition is given in the equations below.

© ISO/IEC 2004 – All rights reserved

If block 'A' was selected as the predictor for the block for which coefficient prediction is to be performed, calculate the first column of the quantised AC coefficients as follows.

$$QF_X[v][0] = PQF_X[v][0] + (QF_A[v][0] * Qp_A) \,//\, Qp_X \qquad v = 1 \text{ to } 7$$

If block 'C' was selected as the predictor for the block for which coefficient prediction is to be performed, calculate the first row of the quantised AC coefficients as follows.

$$QF_X[0][u] = PQF_X[0][u] + (QF_C[0][u] * Qp_C) \,//\, Qp_X \qquad u = 1 \text{ to } 7$$

If the prediction block (block 'A' or block 'C') is outside of the boundary of the VOP or video packet, then all the prediction coefficients of that block are assumed to be zero. $Qp_A$, $Qp_C$ and $Qp_X$ are the quantiser scale values of macroblocks A, C and X respectively.

When AC coefficient prediction is used in conjunction with 'sadct_disable=0', the results of AC prediction has to be post-processed by shape parameters. This shape-adaptive post-processing takes care that those elements of 'QF[v][u]', which are not defined in SA-DCT and which should be zero therefore, are again set to zero after AC prediction. An auxiliary array 'coeff_ width[v]' which has to be derived from the decoded BAB (see subclause A.3.1 in Annex A for details) is used for this purpose: in column 'u=0' ('A'-predictor) or in row 'v=0' ('C'-predictor) all elements of 'QF[v][u]' with 'coeff_width[v] = 0' or 'u >= coeff_width[0]' are set to zero after AC prediction

#### 7.4.3.4 Saturation of QF[v][u]

The quantised coefficients resulting from the DC and AC Prediction are saturated to lie in the range [–2048, 2047]. Thus:

$$QF[v][u] = \begin{cases} 2047 & QF[v][u] > 2047 \\ QF[v][u] & -2048 \le QF[v][u] \le 2047 \\ 2048 & QF[v][u] < 2048 \end{cases}$$

### 7.4.4 Inverse quantisation

The two-dimensional array of coefficients, *QF*[*v*][*u*], is inverse quantised to produce the reconstructed DCT coefficients. This process is a multiplication by the quantisation step size. The quantisation step size is modified by two mechanisms; a quantisation matrix is used to modify the step size within a block and a quantiser scale is used in order that the step size can be modified at the cost of only a few bits (as compared to encoding an entire new quantisation matrix).

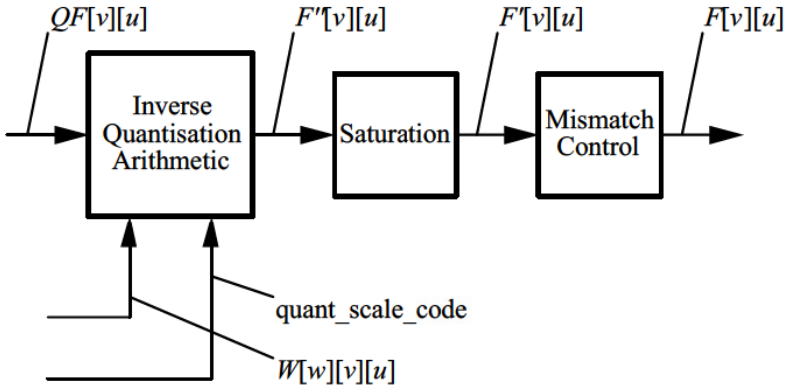



**Figure 7-7 — Inverse quantisation process**

Figure 7-7 illustrates the overall inverse quantisation process. After the appropriate inverse quantisation arithmetic the resulting coefficients, *F''*[*v*][*u*], are saturated to yield *F'*[*v*][*u*] and then a mismatch control operation is performed to give the final reconstructed DCT coefficients, *F*[*v*][*u*].


© ISO/IEC 2004 – All rights reserved

NOTE Attention is drawn to the fact that the method of achieving mismatch control in this part of ISO/IEC 14496 is identical to that employed by ISO/IEC 13818-2:2000.

#### 7.4.4.1 First inverse quantisation method

This subclause specifies the first of the two inverse quantisation methods. The method described here is used when quant_type equals 1.

##### 7.4.4.1.1 Intra DC coefficient

The DC coefficients of intra coded blocks shall be inverse quantised in a different manner to all other coefficients.

In intra blocks *F'*[0][0] shall be obtained by multiplying *QF*[0][0] by a constant multiplier.

The reconstructed DC values are computed as follows.

*F'*[0][0] = *dc_scaler* QF*[0][0]

When short_video_header is 1, dc_scaler is 8, otherwise dc_scaler is defined in Table 7-1.

##### 7.4.4.1.2 Other coefficients

All coefficients other than the DC coefficient of an intra block shall be inverse quantised as specified in this subclause. Two quantisation matrices are used. One shall be used for intra macroblocks and the other for non-intra macroblocks. Each matrix has a default set of values which may be overwritten by down-loading a user defined matrix.

Let the quantisation matrices be denoted by *W*[*w*][*v*][*u*] where *w* takes the values 0 to 1 indicating which of the matrices is being used. *W*[*0*][*v*][*u*] is for intra macroblocks, and *W*[*1*][*v*][*u*] is for non-intra macroblocks. The value of *quantiser_scale* is determined by vop_quant, dquant, dbquant, and quant_scale for luminance and chrominance, and additionally by vop_quant_alpha for grayscale alpha. For example, the value of *quantiser scale* for luminance and chrominance shall be an integer from 1 to 31 when not_8_bit == '0'. The following equation specifies the arithmetic to reconstruct *F''*[*v*][*u*] from *QF*[*v*][*u*] (for all coefficients except intra DC coefficients).

$$F''[v][u] = \begin{cases} 0, & \text{if } QF[v][u] = 0 \\ ((2 \times QF[v][u] + k) \times W[w][v][u] \times quantiser\_scale) \,/\, 16, & \text{if } QF[v][u] \neq 0 \end{cases}$$

where :

$$k = \begin{cases} 0 & \text{intra blocks} \\ Sign(QF[v][u]) & \text{non - intra blocks} \end{cases}$$

NOTE The above equation uses the "/" operator as defined in subclause 4.1.

#### 7.4.4.2 Second inverse quantisation method

This subclause specifies the second of the two inverse quantisation methods. The method described here is used for all the coefficients other than the DC coefficient of an intra block when quant_type==0. In the second inverse quantisation method, the DC coefficient of an intra block is quantised using the same method as in the first inverse quantisation method (see subclause 7.4.4.1.1). The quantiser scale *quantiser_scale* may take integer values from 1 to $2^{quant\_precision}$-1.

##### 7.4.4.2.1 Inverse quantisation

$$|F''[v][u]| = \begin{cases} 0, & \text{if } QF[v][u] = 0, \\ (2 \times |QF[v][u]| + 1) \times quantiser\_scale, & \text{if } QF[v][u] \neq 0,\ quantiser\_scale \text{ is odd}, \\ (2 \times |QF[v][u]| + 1) \times quantiser\_scale - 1, & \text{if } QF[v][u] \neq 0,\ quantiser\_scale \text{ is even}. \end{cases}$$

The sign of *QF*[*v*][*u*] is then incorporated to obtain *F''*[*v*][*u*]: *F''*[*v*][*u*]= Sign(*QF*[*v*][*u*])×|*F''*[*v*][*u*]|.

© ISO/IEC 2004 – All rights reserved

#### 7.4.4.3 Nonlinear inverse DC quantisation

NOTE This subclause is valid for both quantisation methods.

Within an Intra macroblock for which short_video_header is 0, luminance blocks are called type 1 blocks, chroma blocks are classified as type 2. When short_video_header is 1, the inverse quantisation of DC intra coefficients is equivalent to using a fixed value of dc_scaler = 8, as described above in subclause 7.4.1.1.

- DC coefficients of Type 1 blocks are quantised by Nonlinear Scaler for Type 1
- DC coefficients of Type 2 blocks are quantised by Nonlinear Scaler for Type 2

Table 7-1 specifies the nonlinear dc_scaler expressed in terms of piece-wise linear characteristics.

**Table 7-1 — Non linear scaler for DC coefficients of DCT blocks, expressed in terms of relation with quantiser_scale**

| Component:Type | dc_scaler for quantiser_scale range | | | |
|---|---|---|---|---|
| | 1 through 4 | 5 through 8 | 9 through 24 | >= 25 |
| Luminance: Type1 | 8 | 2x quantiser_scale | quantiser_scale +8 | 2 x quantiser_scale -16 |
| Chrominance: Type2 | 8 | (quantiser_scale +13)/2 | | quantiser_scale -6 |

#### 7.4.4.4 Saturation

The coefficients resulting from the inverse quantisation arithmetic are saturated to lie in the range $[-2^{bits\_per\_pixel + 3}, 2^{bits\_per\_pixel + 3} - 1]$. Thus:

$$F'[v][u] = \begin{cases} 2^{bits\_per\_pixel+3} - 1 & F''[v][u] > 2^{bits\_per\_pixel+3} - 1 \\ F''[v][u] & -2^{bits\_per\_pixel+3} \le F''[v][u] \le 2^{bits\_per\_pixel+3} - 1 \\ -2^{bits\_per\_pixel+3} & F''[v][u] < -2^{bits\_per\_pixel+3} \end{cases}$$

#### 7.4.4.5 Mismatch control

This mismatch control is only applicable to the first inverse quantisation method. Mismatch control shall be performed by any process equivalent to the following. Firstly all of the reconstructed, saturated coefficients, *F'[v][u]* in the block shall be summed. This value is then tested to determine whether it is odd or even. If the sum is even then a correction shall be made to just one coefficient; *F*[7][7]. Thus:

$$sum = \sum_{v=0}^{v<8} \sum_{u=0}^{u<8} F'[v][u]$$

$$F[v][u] = F'[v][u] \text{ for all } u, v \text{ except } u = v = 7$$

$$F[7][7] = \begin{cases} F'[7][7] & \text{if } sum \text{ is odd} \\ \left\{ \begin{matrix} F'[7][7] - 1 & \text{if } F'[7][7] \text{ is odd} \\ F'[7][7] + 1 & \text{if } F'[7][7] \text{ is even} \end{matrix} \right\} & \text{if } sum \text{ is even} \end{cases}$$

NOTE 1 It may be useful to note that the above correction for F[7][7] may be implemented by toggling the least significant bit of the twos complement representation of the coefficient. Also since only the "oddness" or "evenness" of the sum is of interest an exclusive OR (of just the least significant bit) may be used to calculate "sum".

NOTE 2 Warning. Small non-zero inputs to the IDCT may result in zero output for compliant IDCTs. If this occurs in an encoder, mismatch may occur in some pictures in a decoder that uses a different compliant IDCT. An encoder should avoid this problem and may do so by checking the output of its own IDCT. It should ensure that it never inserts any non-zero coefficients into the bitstream when the block in question reconstructs to zero through its own IDCT function. If this action is not taken by the encoder, situations can arise where large and very visible mismatches between the state of the encoder and decoder occur.


© ISO/IEC 2004 – All rights reserved

#### 7.4.4.6 Summary of quantiser process for method 1

In summary, the method 1 inverse quantisation process is any process numerically equivalent to:

```
for (v=0; v<8;v++) {
    for (u=0; u<8;u++) {
        if (QF[v][u] == 0)
            F''[v][u] = 0;
        else if ( (u==0) && (v==0) && (macroblock_intra) ) {
            F'[v][u] = dc_scaler * QF[v][u];
        } else {
            if ( macroblock_intra ) {
                F'[v][u] = ( QF[v][u] * W[0][v][u] * quantiser_scale * 2 ) / 16;
            } else {
                F'[v][u] = ( ( ( QF[v][u] *  2 ) + Sign(QF[v][u]) ) * W[1][v][u]
                                                        * quantiser_scale ) / 16;
            }
        }
    }
}

sum = 0;
for (v=0; v<8;v++) {
    for (u=0; u<8;u++) {
        if ( F''[v][u] > 2^(bits_per_pixel + 3) – 1 ) {
            F[v][u] = 2^(bits_per_pixel + 3) – 1;
        } else {
            if ( F''[v][u] < -2^(bits_per_pixel + 3) ) {
                F[v][u] = -2^(bits_per_pixel + 3);
            } else {
                F[v][u] = F'[v][u];
            }
        }
    sum = sum + F[v][u];
    F[v][u] = F[v][u];
    }
}

if ((sum & 1) == 0) {
    if ((F[7][7] & 1) != 0) {
        F[7][7] = F[7][7] - 1;
    } else {
        F[7][7] = F[7][7] + 1;
    }
}
```

### 7.4.5 Inverse DCT

Once the DCT coefficients, F[u][v] are reconstructed, the inverse DCT transform defined in Annex A shall be applied to obtain the inverse transformed values, $f[y][x]$. These values shall be saturated so that: $-2^{bits\_per\_pixel} \leq f[y][x] \leq 2^{bits\_per\_pixel} - 1$, for all $x$, $y$.

© ISO/IEC 2004 – All rights reserved

If flag 'sadct_disable' in the header of VideoObjectLayer() is set to '0', inverse 8x8-DCT – as specified in version 1 - is only applied to 8x8-blocks with 64 opaque pels. In 8x8-blocks with a least one transparent pel (i.e. number of opaque pels is less than 64), the inverse DCT is replaced by inverse shape adaptive DCT (SA-DCT). Similar to 8x8-DCT, SA-DCT reconverts F[v][u] to f[x][y]. The internal processing of inverse SA-DCT is controlled by special shape parameters which have to be computed from decoded BAB. An extended version of SA-DCT, called ΔDC-SA-DCT, is used in intra-coded 8x8-blocks with 'opaque_pels<64'. The decision on the transform to be used in the current 8x8-block depends on the settings of 'video_object_layer_shape', 'sadct_disable', 'vop_coding_type' and 'derived_mb_type' as well as parameter 'opaque_pels'. The decision rule is specified in Table 7-2. Details on internal SA-DCT and ΔDC-SA-DCT processing are given in subclauses 10.3 and 10.4 of Annex A.

**Table 7-2 — Decision rule for the using 8x8-DCT, SA-DCT and ΔDC-SA-DCT**

| decision rules | DCT tool |
|---|---|
| (video_object_layer_shape=="rectangular") \|\| (sadct_disable==1) \|\| (opaque_pels== 64) | 8x8-DCT |
| (video_object_layer_shape !="rectangular") && (sadct_disable==0) && (opaque_pels <64) && (vop_coding_type !="B") && ((derived_mb_type==3) \|\| (derived_mb_type==4)) | ΔDC-SA-DCT |
| (video_object_layer_shape !="rectangular") && (sadct_disable==0) && (opaque_pels < 64) && ( ( (vop_coding_type ="P") && (derived_mb_type!=3) && (derived_mb_type != 4)) \|\| (vop_coding_type ="B")) | SA-DCT |

### 7.4.6 Upsampling of the Inverse DCT output for Reduced Resolution VOP

For the VOP with vop_reduced_resolution flag equal to "1", the upsampling process described in this subclause shall be applied to the Inverse DCT output $f[y][x]$. When the flag is set to "0" or the flag is not present, this upsampling process shall be skipped.

The 16×16 reconstructed prediction error block is obtained by upsampling the 8×8 reconstructed prediction error block $f[y][x]$. In order to realize a simple implementation, filtering is closed within a block which enables to perform an individual upsampling on block basis. Figure 7-8 shows the positioning of samples. The upsampling procedure for the luminance and chrominance pixels which are inside the 16×16 reconstructed prediction error blocks is defined in subclause 7.4.6.1. For the creation of the luminance and chrominance pixels which are at the boundary of 16×16 reconstructed prediction error block, the procedure is defined in subclause 7.4.6.2. Chrominance blocks as well as luminance blocks are up-sampled. The upsampled 16 x 16 prediction error blocks are used for the motion compensation decoding process for Reduced Resolution VOP described in the subclause 7.6.10.


© ISO/IEC 2004 – All rights reserved

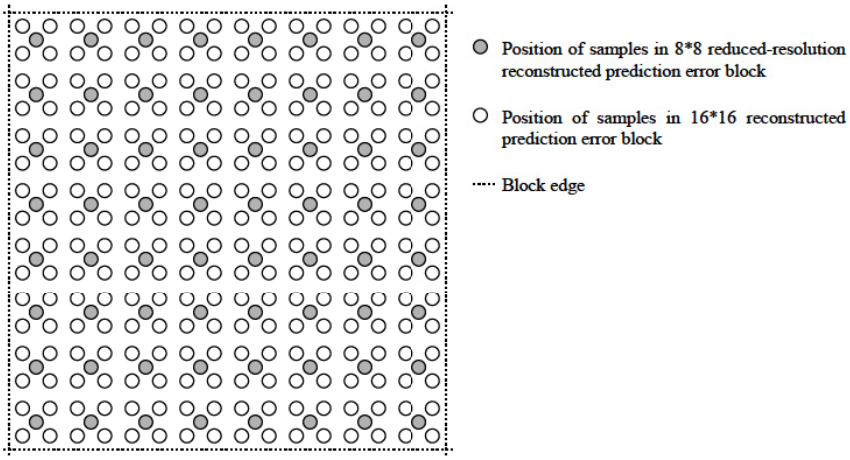



**Figure 7-8 — Positioning of samples in 8×8 reduced-resolution reconstructed prediction error block and 16×16 reconstructed prediction error block**

#### 7.4.6.1 Upsampling procedure for the pixels inside a 16x16 reconstructed prediction error block

The creation of reconstructed prediction error for pixels inside block is described in Figure 7-9.

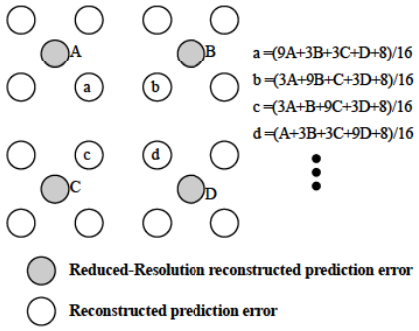



**Figure 7-9 — Creation of reconstructed prediction error for pixels inside block**

#### 7.4.6.2 Upsampling procedure for the pixels at the boundary of 16x16 reconstructed prediction error block

The creation of reconstructed prediction error for pixels of a 16x16 block is shown in Figure 7-10.

© ISO/IEC 2004 – All rights reserved

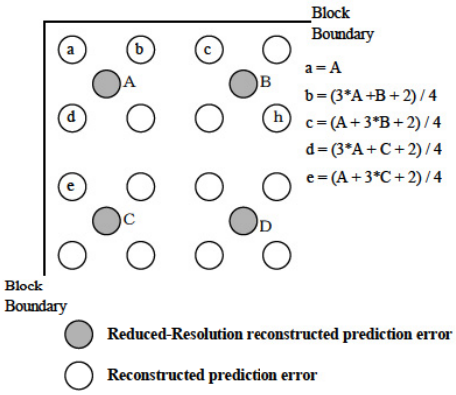



**Figure 7-10 — Creation of reconstructed prediction error for pixels at the block boundary**

## 7.5 Shape decoding

Binary shape decoding is based on a block-based representation. The primary coding methods are block-based context-based binary arithmetic decoding and block-based motion compensation. The primary data structure used is denoted as the binary alpha block (bab). The bab is a square block of binary valued pixels representing the opacity/transparency for the pixels in a specified block-shaped spatial region of size 16x16 pels. In fact, each bab is co-located with each texture macroblock.

### 7.5.1 Higher syntactic structures

#### 7.5.1.1 VOL decoding

If video_object_layer_shape is equal to '00' then no binary shape decoding is required. Otherwise, binary shape decoding is carried out.

#### 7.5.1.2 VOP decoding

If video_object_layer_shape is not equal to '00' then, for each subsequent VOP, the dimensions of the bounding rectangle of the reconstructed VOP are obtained from:

- vop_width
- vop_height

If these decoded dimensions are not multiples of 16, then the values of vop_width and vop_height are rounded up to the nearest integer, which is a multiple of 16.

Additionally, in order to facilitate motion compensation, the horizontal and spatial position of the VOP are obtained from:

- vop_horizontal_mc_spatial_ref
- vop_vertical_mc_spatial_ref

These spatial references may be different for each VOP but the same coordinate system must be used for all VOPs within a vol. Additionally, the decoded spatial references must have an even value.

- vop_shape_coding_type


© ISO/IEC 2004 – All rights reserved

This flag enables the use of intra shape codes in P, B, or S(GMC)-VOPs. Finally, in the VOP class, it is necessary to decode

- change_conv_ratio_disable

This specifies whether conv_ratio is encoded at the macroblock layer.

Once the above elements have been decoded, the binary shape decoder may be applied to decode the shape of each macroblock within the bounding rectangle.

### 7.5.2 Macroblock decoding

The shape information for each macroblock residing within the bounding rectangle of the VOP is decoded into the form of a 16x16 bab.

#### 7.5.2.1 Mode decoding

Each bab belongs to one of seven types listed in Table 7-3. The type information is given by the bab_type field which influences decoding of further shape information. For I-VOPs only three out of the seven modes are allowed as shown in Table 7-3.

**Table 7-3 — List of bab types**

| bab_type | Semantic | Used in |
|---|---|---|
| 0 | MVDs==0 && No Update | P- ,B-, and S(GMC)-VOPs |
| 1 | MVDs!=0 && No Update | P- , B-, and S(GMC)-VOPs |
| 2 | Transparent | All VOP types |
| 3 | Opaque | All VOP types |
| 4 | IntraCAE | All VOP types |
| 5 | MVDs==0 && interCAE | P- , B-, and S(GMC)-VOPs |
| 6 | MVDs!=0 && interCAE | P- , B-, and S(GMC)-VOPs |

##### 7.5.2.1.1 I-VOPs

Suppose that `f(x,y)` is the bab_type of the bab located at `(x,y)`, where x is the BAB column number and y is the BAB row number. The code word for the bab_type at the position `(i,j)` is determined as follows. A context C is computed from previously decoded bab_type's.

C = 27*(f(i-1,j-1)-2) + 9*(f(i,j-1)-2) + 3*(f(i+1,j-1)-2) + (f(i-1,j)-2)

If `f(x,y)` references a bab outside the current VOP, bab_type is assumed to be transparent for that bab (i.e. `f(x,y)=2`). The bab_type of babs outside the current video packet is also assumed to be transparent. The VLC used to decode bab_type for the current bab is switched according to the value of the context C. This context-switched VLC table is given in Table B-27.

##### 7.5.2.1.2 P- and B-, and S(GMC)-VOPs

The decoding of the current bab_type is dependent on the bab_type of the co-located bab in the reference VOP. The reference VOP is either a forward reference VOP or a backward reference VOP. The forward reference VOP is defined as the most recent non-empty (i.e. vop_coded != 0 ) I- or P-, or S(GMC)-VOP in the past, while the backward VOP is defined as the most recently decoded I- or P-, or S(GMC)-VOP in the future. If the current VOP is a P-, or S(GMC)-VOP, the forward reference VOP is selected as the reference VOP. If the current VOP is a B-VOP the following decision rules are applied:

© ISO/IEC 2004 – All rights reserved

1. If the backwards reference VOP is empty, the non-empty one (forward) is selected as the reference VOP for the current B-VOP.

2. If both reference VOPs are non-empty, the forward reference VOP is selected if its temporal distance to the current B-VOP is not larger than that of the backward reference VOP, otherwise, the backward one is chosen.

In the special cases when closed_gov == 1 and the forward reference VOP belongs to the previous GOV, the current B-VOP takes the backward VOP as reference.

If the sizes of the current and reference VOPs are different, some babs in the current VOP may not have a co-located equivalent in the reference VOP. Therefore the bab_type matrix of the reference VOP is manipulated to match the size of the current VOP. Two rules are defined for that purpose, namely a cut rule and a copy rule:

- ***cut rule***. If the number of lines (respectively columns) is smaller in the current VOP than in the reference VOP, the bottom lines (respectively rightmost columns) are eliminated from the reference VOP such that both VOP sizes match.
- ***copy rule***. If the number of lines (respectively columns) is larger in the current VOP than in the reference VOP, the bottom line (respectively rightmost column) is replicated as many times as needed in the reference VOP such that both VOP sizes match.

An example is shown in Figure 7-11 where both rules are applied.

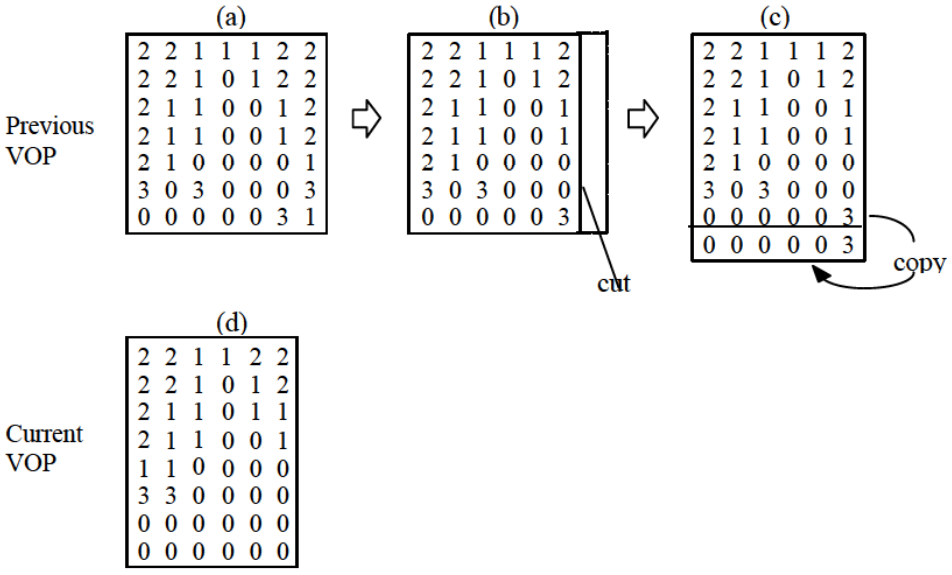



**Figure 7-11 — Example of size fitting between current VOP and reference VOP. The numbers represent the type of each bab**

The VLC to decode the current bab_type is switched according to the value of bab_type of the co-located bab in the reference VOP. This context-switched VLC tables for P, B, and S(GMC)-VOPs are given in Table B.28. If the type of the bab is transparent, then the current bab is filled with zero (transparent) values. A similar procedure is carried out if the type is opaque, where the reconstructed bab is filled with values of 255 (opaque). For both transparent and opaque types, no further decoding of shape-related data is required for the current bab. Otherwise further decoding steps are necessary, as listed in Table 7-4. Decoding for motion compensation is described in subclause 7.5.2.2, and cae decoding in subclause 7.5.2.5.


© ISO/IEC 2004 – All rights reserved

**Table 7-4 — Decoder components applied for each type of bab**

| bab_type | Motion compensation | CAE decoding |
|---|---|---|
| 0 | yes | no |
| 1 | yes | no |
| 2 | no | no |
| 3 | no | no |
| 4 | no | yes |
| 5 | yes | yes |
| 6 | yes | yes |

#### 7.5.2.2 Binary alpha block motion compensation

Motion Vector of shape (MVs) is used for motion compensation (MC) of shape. The value of MVs is reconstructed as described in subclause 7.5.2.3. Unrestricted motion compensation is applied, however, Overlapped MC, half sample MC and 8x8 MC are not carried out. Integer pixel unrestricted motion compensation is carried out on a 16x16 block basis according to subclause 7.5.2.4.

If bab_type is MVDs==0 && No Update or MVDs!=0 && No Update then the motion compensated bab is taken to be the decoded bab, and no further decoding of the bab is necessary. Otherwise, cae decoding is required.

#### 7.5.2.3 Motion vector decoding

If bab_type indicates that MVDs!=0, then mvds_x and mvds_y are VLC decoded. For decoding mvds_x, the VLC given in Table B.29 is used. The same table is used for decoding mvds_y, unless the decoded value of mvds_x is zero. If mvds_x == 0, the VLC given in Table B.30 is used for decoding mvds_y. If bab_type indicates that MVDs==0, then both mvds_x and mvds_y are set to zero.

The integer valued shape motion vector MVs=(mvs_x,mvs_y) is determined as the sum of a predicted motion vector MVPs and MVDs = (mvds_x,mvds_y), where MVPs is determined as follows.

MVPs is determined by analysing certain candidate motion vectors of shape (MVs) and motion vectors of selected texture blocks (MV) around the MB corresponding to the current bab. They are located and denoted as shown in Figure 7-12 where MV1, MV2 and MV3 are rounded up to integer values towards 0. If the selected texture block is a field predicted macroblock, then MV1, MV2 or MV3 are generated by averaging the two field motion vectors and rounding toward zero. Regarding the texture MV's, the convention is that a MB possessing only 1 MV is considered the same as a MB possessing 4 MV's, where the 4 MV's are equal. By traversing MVs1, MVs2, MVs3, MV1, MV2 and MV3 in this order, MVPs is determined by taking the first encountered MV that is defined. If no candidate motion vectors is defined, MVPs = (0,0).

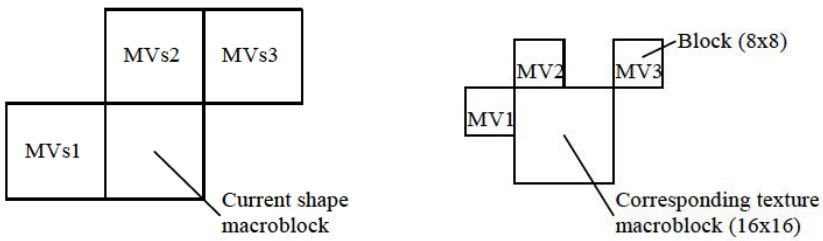



**Figure 7-12 — Candidates for MVPs**

© ISO/IEC 2004 – All rights reserved

In the case that video_object_layer_shape is "binary_only" or vop_coding_type indicates B-VOP, MVPs is determined by considering the motion vectors of shape (MVs1, MVs2 and MVs3) only. The following subclauses explain the definition of MVs1, MVs2, MVs3, MV1, MV2 and MV3 in more detail.

***Defining candidate predictors from texture motion vectors:***

One shape motion vector predictor $MV_i$ ( $i$ =1,2,3 ) is defined for each block located around the current bab according to Figure 7-12 (2). The definition only depends on the transparency of the reference MB. MVi is set to the corresponding block vector as long as it is in a non-transparent reference MB, otherwise, it is not defined. Note that if a reference MB is outside the current VOP or video packet, it is treated as a transparent MB.

***Defining candidate predictors from shape motion vectors:***

The candidate motion vector predictors $MVs_i$ are defined by the shape motion vectors of neighbouring bab located according to Figure 7-12 (1). The $MVs_i$ are defined according to Table 7-5.

**Table 7-5 — Definition of candidate shape motion vector predictors MVs1, MVs2, and MVs3 from shape motion vectors for P, B, and S(GMC)-VOPs. Note that interlaced modes are not included**

| Shape mode of reference MB | $MVs_i$ for each reference shape block-i (a shape block is 16x16) |
|---|---|
| MVDs == 0 or MVDs !=0<br>bab_type 0, 1, 5,6 | The retrieved shape motion vector of the said reference MB is defined as $MVs_i$. Note that $MVs_i$ is defined, and hence valid, even if the reconstructed shape block is transparent. |
| all_0, bab_type 2 | $MVs_i$ is undefined |
| all=255, bab_type 3 | $MVs_i$ is undefined |
| Intra, bab_type 4 | $MVs_i$ is undefined |

If the reference MB is outside of the current video packet, $MV_i$ and $MVs_i$ are undefined.

#### 7.5.2.4 Motion compensation

For inter mode babs (bab_type = 0,1,5 or 6), motion compensation is carried out by simple MV displacement according to the MVs.

Specifically, when bab_type is equal to 0 or 1 i.e. for the no-update modes, a displaced block of 16x16 pixels is copied from the binary alpha map of the previously decoded I or P-, or S(GMC)- VOP for which vop_coded is not equal to '0'. When the bab_type is equal to 5 or 6 i.e. when interCAE decoding is required, then the pixels immediately bordering the displaced block (to the left, right, top and bottom) are also copied from the most recent valid reference VOP's (as defined in subclause 6.3.5) binary alpha map into a temporary shape block of 18x18 pixels size (see Figure 7-15). If the displaced position is outside the bounding rectangle, then these pixels are assumed to be "transparent".

If the current VOP is a B-VOP the following decision rules are applied:

- If the backwards reference VOP is empty, the non-empty one (forward) is selected as the reference VOP for the current B-VOP.
- If both reference VOPs are non-empty, the forward reference VOP is selected if its temporal distance to the current B-VOP is not larger than that of the backward reference VOP, otherwise, the backward one is chosen.

In the special cases when closed_gov == 1 and the forward reference VOP belongs to the previous GOV, the current B-VOP takes the backward VOP as reference.


© ISO/IEC 2004 – All rights reserved

#### 7.5.2.5 Context based arithmetic decoding

Before decoding the binary_arithmetic_code field, border formation (see subclause 7.5.2.5.2) needs to be carried out. Then, if the scan_type field is equal to 0, the bordered to-be decoded bab and the eventual bordered motion compensated bab need to be transposed (as for matrix transposition). If change_conv_rate_disable is equal to 0, then conv_ratio is decoded to determine the size of the sub-sampled BAB, which is 16/conv_ratio by 16/conv_ratio pixels large. If change_conv_rate_disable is equal to 1, then the decoder assumes that the bab is not subsampled and thus the size is 16x16 pixels. Binary_arithmetic_code is then decoded by a context-based arithmetic decoder as follows. The arithmetic decoder is firstly initialised (see subclause 7.5.3.3). The pixels of the sub-sampled bab are decoded in raster order. At each pixel,

1. A context number is computed based on a template, as described in subclause 7.5.2.5.1.
2. The context number is used to access the probability table (Table B.32).
3. Using the accessed probability value, the next bits of binary_arithmetic_code are decoded by the arithmetic decoder to give the decoded pixel value.

When all pixels in sub-sampled BAB have been decoded, the arithmetic decoder is terminated (see subclause 7.5.3.6).

If the scan_type field is equal to 0, the decoded bab is transposed. Then up-sampling is carried out if conv_ratio is different from 1, as described in subclause 7.5.2.5.3. Then the decoded bab is copied into the decoded shape map.

##### 7.5.2.5.1 Context computation

For INTRA coded BABs, a 10 bit context $C = \sum_k c_k \cdot 2^k$ is built for each pixel as illustrated in Figure 7-13 (a), where $c_k$=0 for transparent pixels and $c_k$=1 for opaque pixels.

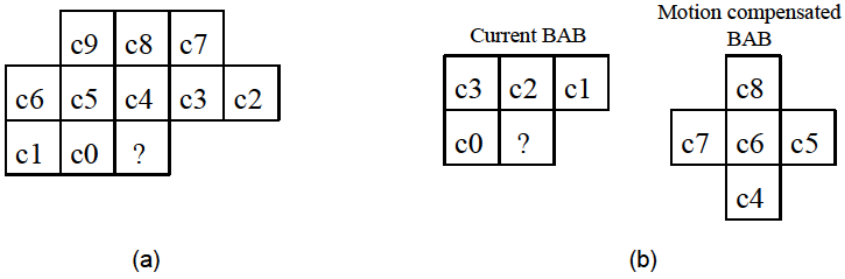



**Figure 7-13 — (a) The INTRA template (b) The INTER template where c6 is aligned with the pixel to be decoded. The pixel to be decoded is marked with '?'**

For INTER coded BABs, temporal redundancy is exploited by using pixels from the bordered motion compensated BAB (depicted in Figure 7-15) to make up part of the context. Specifically, a 9 bit context $C = \sum_k c_k \cdot 2^k$ is built as illustrated in Figure 7-13 (b).

There are some special cases to note.

- When building contexts, any pixels outside the bounding rectangle of the current VOP to the left and above are assumed to be zero (transparent).
- When building contexts, any pixels outside the space of the current video packet to the left and above are assumed to be zero (transparent).
- The template may cover pixels from BABs which are unknown at decoding time. Unknown pixels are defined as area U in Figure 7-14.

© ISO/IEC 2004 – All rights reserved

- The values of these unknown pixels are defined by the following procedure:
    - When constructing the INTRA context, the following steps are taken in the sequence

        1. if (c7 is unknown) c7=c8,

        2. if (c3 is unknown) c3=c4,

        3. if (c2 is unknown) c2=c3.

    - When constructing the INTER context, the following conditional assignment is performed.

    if (c1 is unknown) c1=c2

#### 7.5.2.5.2 Border formation

When decoding a BAB, pixels from neighbouring BABs shall be used to make up the context. For both the INTRA and INTER cases, a 2 pixel wide border about the current BAB is used where pixels values are known, as depicted in Figure 7-14.

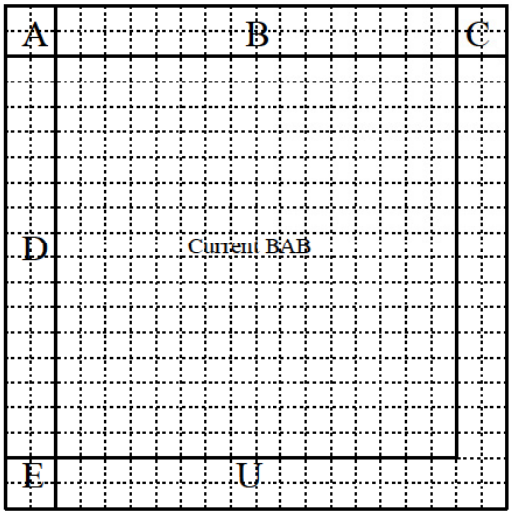



**Figure 7-14 — Bordered BAB. A: TOP_LEFT_BORDER. B: TOP_BORDER. C: TOP_RIGHT_BORDER. D: LEFT_BORDER. E: BOTTOM_LEFT_BORDER. U: pixels which are unknown when decoding the current BAB**

If the value of conv_ratio is not equal to 1, a sub-sampling procedure is further applied to the BAB borders for both the current BAB and the motion compensated BAB.

The border of the current BAB is partitioned into 5 regions:

- TOP_LEFT_BORDER, which contains pixels from the BAB located to the upper-left of the current BAB and which consists of 2 lines of 2 pixels
- TOP_BORDER, which contains pixels from the BAB located above the current BAB and which consists of 2 lines of 16 pixels
- TOP_RIGHT_BORDER, which contains pixels from the BAB located to the upper-right of the current BAB and which consists of 2 lines of 2 pixels
- LEFT_BORDER, which contains pixels from the BAB located to the left of the current BAB and which consists of 2 columns of 16 pixels


© ISO/IEC 2004 – All rights reserved

- BOTTOM_LEFT_BORDER, which contains pixels from the BAB located to the bottom-left of the current BAB and which consists of 2 lines of 2 pixels

The TOP_LEFT_BORDER and TOP_RIGHT_BORDER are not sub-sampled, and kept as they are. The TOP_BORDER and LEFT_BORDER are sub-sampled such as to obtain 2 lines of 16/conv_ratio pixels and 2 columns of 16/conv_ratio pixels, respectively.

The sub-sampling procedure is performed on a line-basis for TOP_BORDER, and a column-basis for LEFT_BORDER. For each line (respectively column), the following algorithm is applied: the line (respectively column) is split into groups of conv_ratio pixels. For each group of pixels, one pixel is associated in the sub-sampled border. The value of the pixel in the sub-sampled border is OPAQUE if half or more pixels are OPAQUE in the corresponding group. Otherwise the pixel is TRANSPARENT.

The 2x2 BOTTOM_LEFT_BORDER is filled by replicating downwards the 2 bottom border samples of the LEFT_BORDER after the down-sampling (if any).

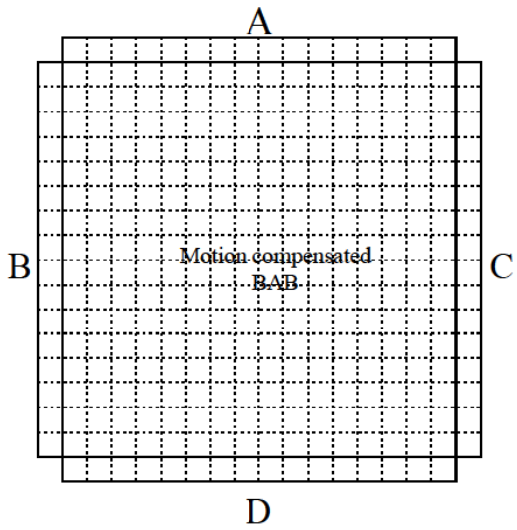



**Figure 7-15 — Bordered motion compensated BAB. A: TOP_BORDER. B: LEFT_BORDER. C: RIGHT_BORDER. D: BOTTOM_BORDER**

In the case of a motion compensated BAB, the border is also partitioned into 4, as shown Figure 7-15:

- TOP_BORDER, which consists of a line of 16 pixels
- LEFT_BORDER, which consists of a column of 16 pixels
- RIGHT_BORDER, which consists of a column of 16 pixels
- BOTTOM_BORDER, which consists of a line of 16 pixels

The very same sub-sampling process as described above is applied to each of these borders.

#### 7.5.2.5.3 Upsampling

When conv_ratio is different from 1, up-sampling is carried out for the BAB. This is illustrated in Figure 7-16 where "O" in this figure is the coded pixel and "X" is the interpolated pixel. To compute the value of the interpolated pixel, a filter context from the neighboring pixels is first calculated. For the pixel value calculation, the value of "0" is used for a transparent pixel, and "1" for an opaque pixel. The values of the interpolated pixels (Pi, i=1,2,3,4, as shown in Figure 7-17) can then be determined by the following equation:

© ISO/IEC 2004 – All rights reserved

P1 : if( 4*A + 2*(B+C+D) + (E+F+G+H+I+J+K+L) > Th[Cf]) then "1" else "0"

P2 : if( 4*B + 2*(A+C+D) + (E+F+G+H+I+J+K+L) > Th[Cf]) then "1" else "0"

P3 : if( 4*C + 2*(B+A+D) + (E+F+G+H+I+J+K+L) > Th[Cf]) then "1" else "0"

P4 : if( 4*D + 2*(B+C+A) + (E+F+G+H+I+J+K+L) > Th[Cf]) then "1" else "0"

The 8-bit filter context, Cf, is calculated as follows:

$$C_f = \sum_k c_k \cdot 2^k$$

Based on the calculated Cf, the threshold value (Th[Cf]) can be obtained from the look-up table as follows:

Th[256] = {

3, 6, 6, 7, 4, 7, 7, 8, 6, 7, 5, 8, 7, 8, 8, 9,

6, 5, 5, 8, 5, 6, 8, 9, 7, 6, 8, 9, 8, 7, 9, 10,

6, 7, 7, 8, 7, 8, 8, 9, 7, 10, 8, 9, 8, 9, 9, 10,

7, 8, 6, 9, 6, 9, 9, 10, 8, 9, 9, 10, 11, 10, 10, 11,

6, 9, 5, 8, 5, 6, 8, 9, 7, 10, 10, 9, 8, 7, 9, 10,

7, 6, 8, 9, 8, 7, 7, 10, 8, 9, 9, 10, 9, 8, 10, 9,

7, 8, 8, 9, 6, 9, 9, 10, 8, 9, 9, 10, 9, 10, 10, 9,

8, 9, 11, 10, 7, 10, 10, 11, 9, 12, 10, 11, 10, 11, 11, 12,

6, 7, 5, 8, 5, 6, 8, 9, 5, 6, 6, 9, 8, 9, 9, 10,

5, 8, 8, 9, 6, 7, 9, 10, 6, 7, 9, 10, 9, 10, 10, 11,

7, 8, 6, 9, 8, 9, 9, 10, 8, 7, 9, 10, 9, 10, 10, 11,

8, 9, 7, 10, 9, 10, 8, 11, 9, 10, 10, 11, 10, 11, 9, 12,

7, 8, 6, 9, 8, 9, 9, 10, 10, 9, 7, 10, 9, 10, 10, 11,

8, 7, 7, 10, 7, 8, 8, 9, 9, 10, 10, 11, 10, 11, 11, 12,

8, 9, 9, 10, 9, 10, 10, 9, 9, 10, 10, 11, 10, 11, 11, 12,

9, 10, 10, 11, 10, 11, 11, 12, 10, 11, 11, 12, 11, 12, 12, 13 };

TOP_LEFT_BORDER, TOP_RIGHT_BORDER, sub-sampled TOP_BORDER and sub-sampled LEFT_BORDER described in the previous subclause are used. The other pixels outside the BAB are extended from the outermost pixels inside the BAB as shown in Figure 7-16.

In the case that conv_ratio is 4, the interpolation is processed twice. The above mentioned borders of 4x4 BAB are used for the interpolation from 4x4 to 8x8, and top-border (respectively left-border) for the interpolation from 8x8 to 16x16 are up-sampled from the 4x4 BAB top-border (respectively left-border) by simple repetition.

When the BAB is on the left (and/or top) border of VOP, the borders outside VOP are set to zero value. The upsampling filter shall not use pixel values outside of the current video packet.


© ISO/IEC 2004 – All rights reserved

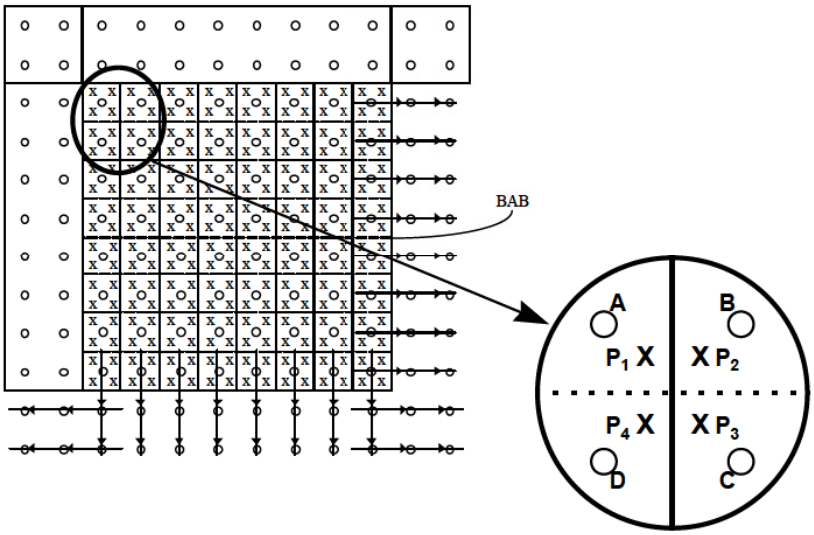



Figure 7-16 — Upsampling

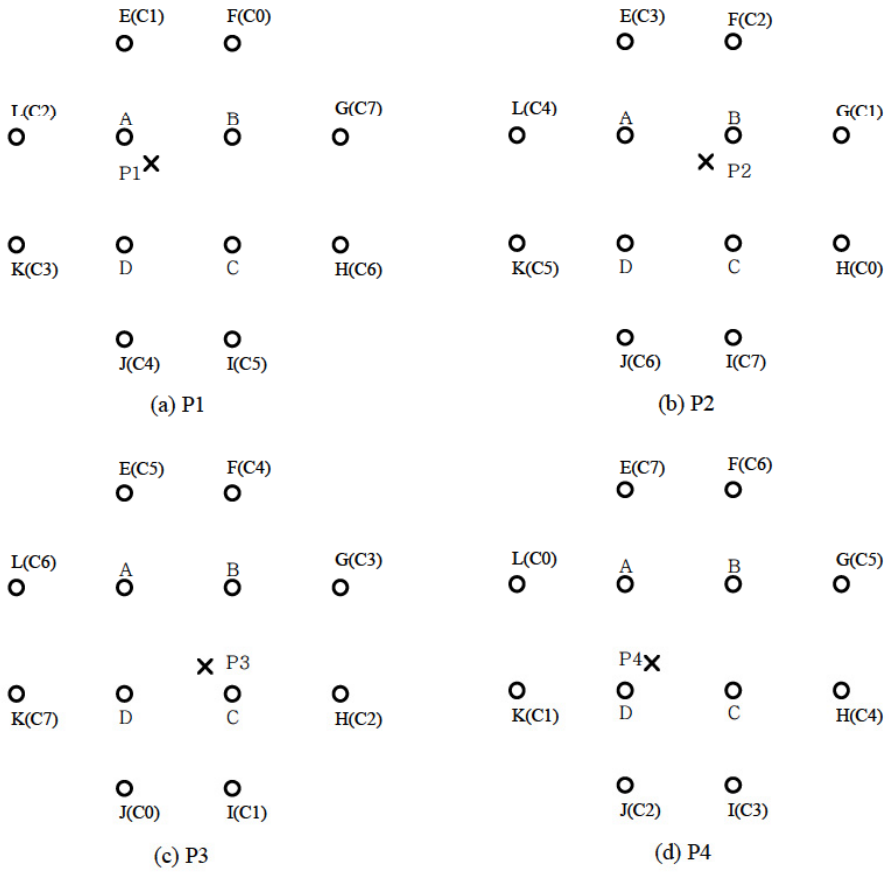



Figure 7-17 — Interpolation filter and interpolation construction

© ISO/IEC 2004 – All rights reserved

#### 7.5.2.5.4 Down-sampling process in inter case

If bab_type is '5' or '6' (see Table 7-4), downsampling of the motion compensated bab is needed for calculating the 9 bit context in the case that conv_ratio is not 1. The motion compensated bab of size 16x16 pixels is down sampled to bab of size 16/conv_ratio by 16/conv_ratio pixels by the following rules:

- conv_ratio==2

If the average of pixel values in 2 by 2 pixel block is equal to or greater than 127.5 the pixel value of the downsampled bab is set to 255 otherwise it is set to 0.

- conv_ratio==4

If the average of pixel values in 4 by 4 pixel block is equal to or greater than 127.5 the pixel value of the downsampled bab is set to 255 otherwise it is set to 0.

### 7.5.3 Arithmetic decoding

Arithmetic decoding consists of four main steps:

- Removal of stuffed bits
- Initialization which is performed prior to the decoding of the first symbol
- Decoding of the symbol themselves. The decoding of each symbol may be followed by a re-normalization step.
- Termination which is performed after the decoding of the last symbol

#### 7.5.3.1 Registers, symbols and constants

Several registers, symbols and constants are defined to describe the arithmetic decoder.

- HALF: 32-bit fixed point constant equal to ½ (0x80000000)
- QUARTER: 32-bit fixed point constant equal to ¼ (0x40000000)
- L: 32-bit fixed point register. Contains the lower bound of the interval
- R: 32-bit fixed point register. Contains the range of the interval.
- V: 32-bit fixed point register. Contains the value of the arithmetic code. V is always larger than or equal to L and smaller than L+R.
- p0: 16-bit fixed point register. Probability of the '0' symbol.
- p1: 16-bit fixed point register. Probability of the '1' symbol.
- LPS: boolean. Value of the least probable symbol ('0' or '1').
- bit: boolean. Value of the decoded symbol.
- pLPS: 16-bit fixed point register. Probability of the LPS.
- rLPS: 32-bit fixed point register. Range corresponding to the LPS.


© ISO/IEC 2004 – All rights reserved

#### 7.5.3.2 Bit stuffing

In order to avoid start code emulation, 1's are stuffed into the bitstream whenever there are too many successive 0's. If the first MAX_HEADING bits are 0's, then a 1 is transmitted after the MAX_HEADING-th 0. If MAX_MIDDLE or more 0's are sent successively a 1 is inserted after the MAX_MIDDLE-th 0. If the number of trailing 0's is larger than MAX_TRAILING, then a 1 is appended to the stream. The decoder shall properly skip these inserted 1's when reading data into the V register (see subclauses 7.5.3.3 and 7.5.3.5).

MAX_HEADING equals 3, MAX_MIDDLE equals 10, and MAX_TRAILIING equals 2.

#### 7.5.3.3 Initialization

The lower bound L is set to 0, the range R to HALF-0x1 (0x7fffffff) and the first 31 bits are read in register V.

#### 7.5.3.4 Decoding a symbol

When decoding a symbol, the probability p0 of the '0' symbol is provided according to the context computed in subclause 7.5.2.5.1 and using Table B.32. p0 uses a 16-bit fixed-point number representation. Since the decoder is binary, the probability of the '1' symbol is defined to be 1 minus the probability of the '0' symbol, i.e. p1 = 1-p0.

The least probable symbol LPS is defined as the symbol with the lowest probability. If both probabilities are equal to ½ (0x8000), the '0' symbol is considered to be the least probable.

The range rLPS associated with the LPS may be computed as R*pLPS: The 16 most significant bits of register R are multiplied by the 16 bits of pLPS to obtain the 32 bit rLPS number.

The interval [L,L+R) is split into two intervals [L,L+R-rLPS) and [L+R-rLPS,L+R). If V is in the latter interval then the decoded symbol is equal to LPS. Otherwise the decoded symbol is the opposite of LPS. The interval [L,L+R) is then reduced to the sub-interval in which V lies.

After the new interval has been computed, the new range R might be smaller than QUARTER. If so, re-normalization is carried out, as described below.

#### 7.5.3.5 Re-normalization

As long as R is smaller than QUARTER, re-normalization is performed.

- f the interval [L,L+R) is within [0,HALF), the interval is scaled to [2L,2L+2R). V is scaled to 2V.
- If the interval [L,L+R) is within [HALF,1) the interval is scaled to [2(L-HALF),2(L-HALF)+2R). V is scaled to 2(V-HALF).
- Otherwise the interval is scaled to [2(L-QUARTER),2(L-QUARTER)+2R). V is scaled to 2(V-QUARTER).

After each scaling, a bit is read and copied into the least significant bit of register V.

#### 7.5.3.6 Termination

After the last symbol has been decoded, additional bits need to be "consumed". They were introduced by the encoder to guarantee decodability.

In general 3 further bits need to be read. However, in some cases, only two bits need to be read. These cases are defined by:

- if the current interval covers entirely [QUARTER-0x1,HALF)
- if the current interval covers entirely [HALF-0x1, 3QUARTER)

© ISO/IEC 2004 – All rights reserved

After these additional bits have been read, 32 bits shall be "unread", i.e. put the content of register V back into the bit buffer.

#### 7.5.3.7 Software

The example software for arithmetic decoding for binary shape decoding is included in Annex B.

### 7.5.4 Spatial scalable binary shape decoding

#### 7.5.4.1 Decoding of base layer

The decoding process of the base layer is the same as non-scalable binary shape decoding process.

#### 7.5.4.2 Decoding of enhancement layer

When spatial scalability is enabled (scalability is set to 1 and hierarchy_type is set to 0) with enhancement_type == 0 or when spatial scalability is enabled with enhancement_type == 1 and use_ref_shape == 0, scalable shape coding process is used for decoding of binary shape.

If spatial scalability is enabled, use_ref_shape is set to 1 and enhancement_type is set to 1, the same non-scalable decoding process is applied for binary shape of enhancement layer. In this case, the following rules are applied for enhancement layer.

1. In PVOP, Inter shape coding should be done as bab_type of co-located MB in the reference VOP (lower layer) is "Opaque".

2. In BVOP, forward reference VOP, most recently decoded non-empty VOP in the same layer, is always selected as reference VOP of shape coding.

If spatial scalability is enabled, use_ref_shape is set to 1 and enhancement_type is set to 0, then the up-sampled binary shape from reference layer is used for the binary shape of enhancement layer. The up sampling and down sampling process of this purpose also follows up-down sampling method described in the subclause 7.5.4.4.

When spatial scalability is enabled and enhancement_type is set to 0, in the enhancement layer, the forward prediction in P-VOP and the backward prediction in B-VOP are used as the spatial prediction. In that case the shape information is coded by scan interleaving (SI) based method. For the forward prediction in B-VOP a motion compensated temporal prediction is made from the reference VOP in the enhancement layer. In that case the shape information is coded by the CAE method as like in base layer except that the shape motion vectors are obtained from those of the collocated bab in the lower layer. Motion vector and shape coding mode(bab_type) of collocated bab in the lower layer are used for decoding the enhancement layer bab.

The location of collocated bab in the lower layer can be found by following method.

```
collocated_MBX
    = min ( max ( 0, current_MBX*shape_hor_sampling_factor_m/shape_hor_sampling_factor_n ),
            NumMBXLower-1 );

collocated_MBY
    = min ( max( 0, current_MBY*shape_ver_sampling_factor_m/shap_ever_sampling_factor_n ),
            NumMBYLower-1);
```

For the current MB location [current_MBX, current_MBY], the location of collocated bab in the reference layer is denoted as [collocated_MBX, collocated_MBY]. current_MBX, current_MBY, collocated_MBX and collocated_MBY are the MB-unit coordinations. NumMBXLower and NumMBYLower denote the number of microblocks in the lower layer VOP on horizontal and vertical directions, respectively.


© ISO/IEC 2004 – All rights reserved

#### 7.5.4.3 Prediction in the enhancement layer

A motion compensated temporal prediction is made from the reference VOP in the enhancement layer. In addition, a spatial prediction is formed from the lower reference layer decoded VOP. These predictions are selected individually or combined to form the actual prediction.

vop_shape_coding_type is not used in the Spatial Scalable Binary Shape Coding process.

#### 7.5.4.4 Spatial prediction

The spatial prediction is made by resampling the lower reference layer reconstructed VOP to the same sampling grid as the enhancement layer. For the resampling, repetition is used on the lower layer.

For enhancement layer encoding/decoding, the reference layer VOP should be up-sampled as the sampling ratio, which is included in the VOL syntax. In VOL syntax for enhancement layer, there are four fields for the up-sampling ratio, i.e., shape_hor_sampling_factor_n, shape_hor_sampling_factor_m, shape_vert_sampling_factor_n and shape_vert_sampling_factor_n.

**Down sampling**
For the sampling rate, below sampling process is adopted for both horizontal and vertical directions.

For the down-sampling rate m/n,

$m/n = [1/2^k][(m2^k)/n]$ , where k is the largest integer that is equal or smaller than $\log_2(n/m)$.

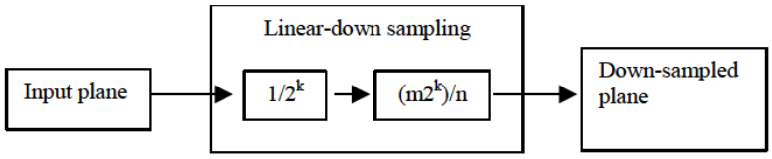



**Figure 7-18 — Down sampling process**

1. Find $k = \text{floor}( \log_2(n/m) )$.

2. Sampling a pixel per $2^k$ pixels:

   if( $x\%2^k == (2^k - 1)$ ) Sampling F(x) pixel;

3. On the result of 2, sampling a pixel per $n/(m2^k)$ pixels.

**Up sampling**
For the sampling rate, below sampling process is adopted for both horizontal and vertical directions.

For the up-sampling rate n/m,

$n/m = [n/(m2^k)]\,[2^k]$, where k is the largest integer that is equal or smaller than $\log_2(n/m)$.

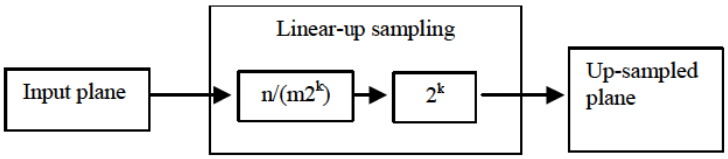



**Figure 7-19 — Up sampling process**

© ISO/IEC 2004 – All rights reserved

1. Find k = floor( $\log_2(n/m)$ ).

2. Repeat one pixel per $(m2^k)/(n-m2^k)$ pixels.

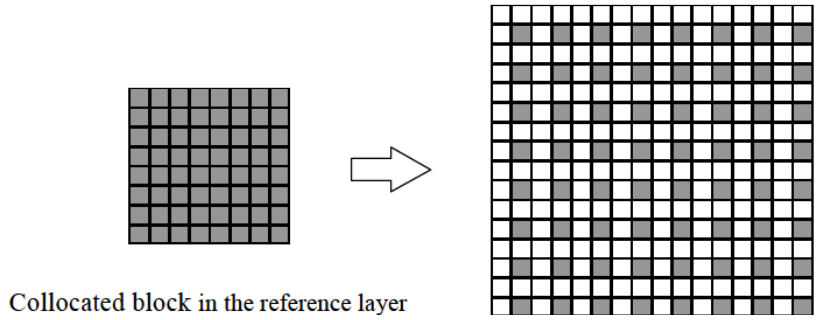
3. On the result of 2, Repeat $2^k - 1$ pixels per pixel.

For a case that the sampling ratio is 1/2 for both horizontal and vertical direction, one pixel value in the lower layer is upsampled to 2x2 pixels in the current layer by the following methods:

The binary alpha image $f_{current}[y_H][x_H]$ in the enhancement layer is upsampled by repetition of the pixels of the image $f_{lower}[y_L][x_L]$ in the lower layer according to the following formula:

$$f_{current}[2\,y_L+i][2\,x_L+j] = f_{lower}[y_L][x_L], \qquad (i, j=0, 1)$$

In Figure 7-20, the spatial prediction of one BAB is shown for example. In the enhancement layer the pixel value $f_{current}[2y_L+1][2x_L+1$ is always same with the corresponding pixel value $f_{lower}[y_L][x_L]$ in the lower layer.



Collocated block in the reference layer

Upsampled BAB for spatial prediction

**Figure 7-20 — Associated pixel locations and spatial prediction of current BAB by repetition**

In the case of arbitrary shape VOPs, its size and locations change from time to time. In order to ensure the formation of the spatial prediction, it is necessary to identify the location of the resampled VOPs in the absolute coordinate system. vop_horizontal_mc_spatial_ref and vop_vertical_mc_spatial_ref in the resampled reference layer should be coincident with those in the enhancment layer. Hence after the position of the spatial reference of the reference layer is scaled by *_sampling_factor_n/*_sampling_factor_m, the resampled reference VOP should be relocated by referencing the offset values that are the differences between the positions of the spatial references of two VOPs in the horizontal and the vertical direction.

Since the shape motion vectors and the shape modes in the reference layer are also needed for the actual shape coding in the enhancement layer bab, the bab type matrix and the motion vector matrix are resampled by repetition. In case of the motion vector matrix each components are scaled by the sampling ratio on both directions.

#### 7.5.4.5 Temporal prediction

A motion compensated temporal prediction is made from the reference VOP in the enhancement layer. For predicting a bab, the motion vectors of the collocated bab in the lower layer is used. Enhancement layer shape motion vector [mvX, mvY] can be used by scaling the lower layer shape motion vector [mvXLower, mvYLower]. The scaling method is as follows:

mvX = (mvXLower * shape_hor_sampling_factor_n) // shape_hor_sampling_factor_m

mvY = (mvYLower * shape_ver_sampling_factor_n) // shape_ver_sampling_factor_m


© ISO/IEC 2004 – All rights reserved

#### 7.5.4.6 Types for enhancement layer bab

Each bab belongs to one of four types given in Table 7-6. This enh_bab_type field influences decoding of further shape information of enhancement layers. For P-VOP coding only 0 and 1 of the enh_bab_type are allowed. For B-VOP coding scheme, all of the four modes are allowed.

**Table 7-6 — List of enh_bab_types and usage**

| enh_bab_type | Semantic | Used in |
|---|---|---|
| 0 | intra NOT_CODED | P-/B- VOPs |
| 1 | intra CODED | P-/B- VOPs |
| 2 | inter NOT_CODED | B- VOPs |
| 3 | inter CODED | B- VOPs |

When the enh_bab_type == 1, the bab is predicted by upsampling from the collocated block in the lower reference layer and is decoded by intra-mode CAE based on Scan Interleaving(SI).

The enh_bab_type == 0 means that the error of the predicted bab from the lower reference layer is less than the threshold. The predicted bab is given by upsampling the collocated block in the lower reference layer. The spatial prediction method in 7.5.4.4 or linear repetition method can be used for the upsampling. The predicted bab is used as the current bab.

For enh_bab_type==2 and enh_bab_type==3, no motion vector is decoded for the temporal prediction of enhancement layer bab. The motion vector of the collocated block in the lower reference layer can be utilized. Each component of the vector are scaled by two for referencing. If the collocated bab in the lower reference layer is intra coding mode, the value of the motion vector is set to zero. If enh_bab_type is decode as "inter NOT_CODED", then the error of the predicted bab from the previous VOP is less than the threshold. And the predicted bab is used as the current bab. If enh_bab_type is decode as "inter CODED", the current bab is decoded by inter-CAE which is used in non-scalable shape decoding.

#### 7.5.4.7 Decoding bab types for enhancement layer

In order to decode the bab types, the collocated bab of the lower layer is also examined. Based on the lower layer bab types, the following VLC table is used for decoding the enh_bab_type field. For B-VOP decoding in enhancement layer enh_bab_type is decoded using in Table 7-8. For P-VOP decoding in enhancement layer, there are only two enh_bab_type fields, i.e., 0 and 1, and they are decoded by 1~2 bit described in Table 7-7.

**Table 7-7 — Decoding of enh_bab_type for P-VOP in enhancement layer**

| enh_bab_type | Code |
|---|---|
| 0 | 1 |
| 1 | 01 |

**Table 7-8 — Decoding of enh_bab_type for B-VOP in enhancement layer**

| | | enh_bab_type in the enhancement layer | | | |
|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 |
| enh_bab_type in the lower reference layer or (bab_type in the base layer) | 0(2,3) | 1 | 01 | 001 | 000 |
| | 1(4) | 110 | 0 | 10 | 111 |
| | 2(0,1) | 001 | 01 | 1 | 000 |
| | 3(5,6) | 110 | 0 | 111 | 10 |

© ISO/IEC 2004 – All rights reserved

Note that the numbers 0-3 in these tables correspond to the enh_bab_types given in 7.8.2.1.6. When the lower layer is the base layer, {"MVDs==0 && No Update", "MVDs!=0 && No Update"}, {"all_0", "all_255"}, {"intraCAE"} and {"MVDs==0 && interCAE", "MVDs!=0 && interCAE"} modes are converted to the enh_bab_type 2, 0, 1 and 3, respectively.

#### 7.5.4.8 Intra coded enhancement layer decoding

Intra coded enhancement layer decoding uses scan interleaving algorithm before performing intra-mode CAE. The decoding order with SI scanning is as follows:

1. Copy B from reference layer

2. Decoding order with Vertical scanning : Vr --> Vp1 --> Vp2 --> ... --> Vpk

3. Decoding order with Horizontal scanning : Hr --> Hp1 --> Hp2 --> ... --> Hpl

where,

B: Pixel that can be copied from collocated pixel in the reference layer.$V_r$: Vertical scanning Pixel of Residual term of vertical linear-up sampling

$V_{pk}$: Vertical scanning Pixel of $2^k$ term of vertical linear-up sampling

$H_r$: Horizontal scanning Pixel of Residual term of Horizontal linear-up sampling

$H_{pl}$: Horizontal scanning Pixel of $2^l$ term of Horizontal linear-up sampling

k: The largest integer that is equal or smaller than $\log_2$(shape_hor_sampling_factor_n / shape_hor_sampling_factor_m)

l: The largest integer that is equal or smaller than $\log_2$(shape_ver_sampling_factor_n / shape_ver_sampling_factor_m)

For an example, when shape_hor_sampling_factor_n = 8, shape_hor_sampling_factor_m = 3, shape_ver_sampling_factor_n = 5, shape_hor_sampling_factor_m =1, the scan order of pixels for decoding is shown in Figure 7-21.


© ISO/IEC 2004 – All rights reserved

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 |
| Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 |
| Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 |
| Hr | Hr | Hr | Hr | Hr | Hr | Hr | Hr | Hr | Hr | Hr | Hr | Hr | Hr | Hr | Hr |
| Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 |
| Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 |
| Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 |
| Vp1 | Vr | Vp1 | B | Vp1 | B | Vp1 | B | Vp1 | Vr | Vp1 | B | Vp1 | B | Vp1 | B |
| Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 |
| Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 |
| Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 |
| Vp1 | Vr | Vp1 | B | Vp1 | B | Vp1 | B | Vp1 | Vr | Vp1 | B | Vp1 | B | Vp1 | B |
| Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 |
| Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 | Hp1 |
| Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 | Hp2 |
| Vp1 | Vr | Vp1 | B | Vp1 | B | Vp1 | B | Vp1 | Vr | Vp1 | B | Vp1 | B | Vp1 | B |

**Figure 7-21 — An example of decoding order when enh_bab_type==1 (Decoding order: B → $V_r$ → $V_{p1}$ → $H_r$ → $H_{p1}$ → $H_{p2}$)**

### 7.5.4.9 Border formation for intra-mode CAE for enhancement layer decoding

The border formation for decoding of intra CODED bab (enh_bab_type==1) is the same as that of the non-scalable shape coding (see subclause 7.5.2.5.2) except E and U regions in Figure 7-14. The border of E and U region of the current bordered bab contains the pixels from the collocated regions in the up-sampled lower layer plane and which consists of 2 lines in bottom and right of the current bab.

### 7.5.4.10 Scan Interleaving Algorithm

The spatial (or resolution) scalability can be easily sought by incorporating the so-called Scan Interleaving (SI) method. In SI, coded-scan-lines are decoded by using the prediction from the closest upper and lower neighboring two lines (reference-scan-lines) as shown in Figure 7-22. In the figure, horizontal direction scanning is described. The pixels on coded-scan-lines are the pixels to be decoded and the pixels on reference-scan-lines are the pixels that are already reconstructed The closest upper and lower neighboring pixel values in reference-scan-lines are used for reconstructing the current pixel. (The closest upper and lower neighboring pixels mean the closest left and right neighboring pixels in vertical direction scanning case.) By using those two reference pixels, the current decoded pixel has three types as follows:

Transitional sample data (TSD): The closest upper and lower neighboring pixels that are already reconstructed have different colors each other.

Exceptional sample data (ESD): The closest upper and lower neighboring pixels that are already reconstructed are same and the current pixel is different with the neighboring pixels.

© ISO/IEC 2004 – All rights reserved

Predictable sample data (PSD): The closest upper and lower neighboring pixels that are already reconstructed are the same as the current pixel.

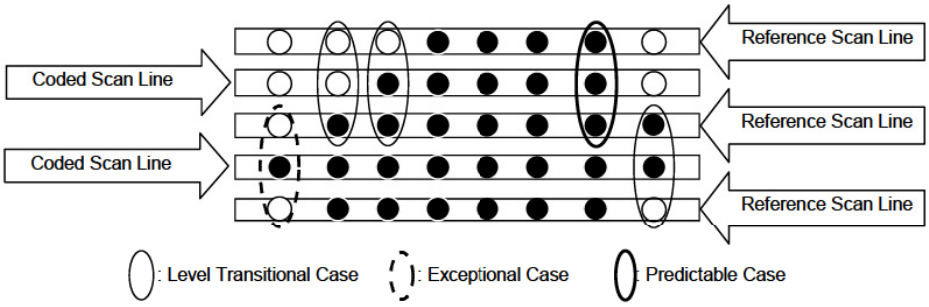



**Figure 7-22 — Scan Interleaving for Scalable Coding: an example of a times by 2 case**

To encoding/decoding shape mask plane, SI performs the scanning method in vertical and horizontal directions, successively. SI with vertical scanning (V-SI) and SI with horizontal scanning (H-SI) are separately applied to the input image.

For decoding a bab in the current layer, the spatially predicted bab from collocated block in the lower layer is enhanced by V-SI. And then H-SI is successively performed on the enhanced plane of the V-SI. The reconstructing order is presented in subclause 7.5.4.8.

In the first step of decoding a bab, a binary flag, which denotes the current bab type of SI, is decoded by BAC. There are two bab types of SI : transitional bab (TSD-B) and exceptional bab (ESD-B). If a bab has no ESD, the bab is set to TSD-B. Otherwise the bab is set to ESD-B. When the flag is '0', the current bab is TSD-B. Otherwise, the current bab is ESD-B.

In the encoder, for encoding TSD-B, only TSD pixel are encoded by CAE with the context shown in Figure 7-24. And for encoding ESD-B, all the pixels in the coded-scan-lines are encoded by CAE with the context shown in Figure 7-24. The encoding order is the same as shown in subclause 7.5.4.8.

If a bab is TSD-B, only TSD are decoded in the order of V-SI and H-SI. The specific decoding order is presented in 7.5.4.8. The decoding process of TSD-B is as follows.

Following the scanning order, if the current pixel is TSD, then the pixel is decoded by BAC with the context in Figure 7-24. And if the two neighbor pixels (the closest upper(left) and bottom(right)) are the same value, then the current pixel has the same value as its neighbor pixels.

If a bab is ESD-B, all the pixels in coded scan line are decoded by BAC with the context in Figure 7-24.

#### 7.5.4.11 Determination of the scanning type in enhancement layer decoding

When enh_bab_type is 1 (intra CODED) and the scale ratio between the layers is square (i.e 200x300 <--> 400x600), before performing BAC, the scanning type should be determined. For the determination, the top and left borders of the collocated block of the lower layer are extended with its neighboring pixels like Figure 7-23. If an extended border is outside VOP, the pixel values of the extended border are set to zero. Two counts, $N_L$ and $N_U$ are calculated ($N_L$: the number of pixels which have different values from the left neighboring pixel value; $N_U$: the number of pixels which have different values from the upper neighboring value). If $N_U$ is greater than $N_L$, the spatially predicted bab for decoding the enhancement layer bab has to be transposed and then the intra-mode CAE for enhancement layer is performed, Otherwise, the intra-mode CAE for enhancement layer is performed on the spatially predicted bab without transposing, that is default condition.


© ISO/IEC 2004 – All rights reserved

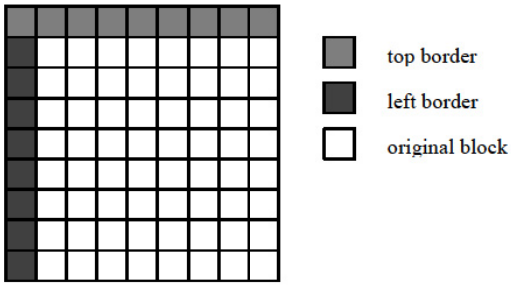



**Figure 7-23 — Extension of border pixels**

#### 7.5.4.12 Decoding of bab type of SI

The first decoded symbol of enh_binary_arithmetic_code(), which performs BAC, denotes the current bab type of SI. The probability of SI_bab_type_prob[] given in Annex B is used for decoding bab type of SI. When this flag is '0', the current bab is TSD-B. Otherwise, the current bab is ESD-B.

#### 7.5.4.13 Decoding of intra-mode CAE for enhancement layer

For decoding intra mode CAE for enhancement layer, BAC using the V-SI and H-SI decoding method is performed. By using neighboring 7 bits context, a pixel is arithmetic decoded as in CAE, such that:

Compute a context using the templates shown in Figure 7-24 ((a): for V-SI, (b): for H-SI).

Using the context, find probability from the probability table, i.e. enh_intra_v_prob[] for V-SI and enh_intra_h_prob[] for H-SI. These tables are given in Annex B.

Decoding the pixel value using the probability.

The following subclause describes the computation of the contexts for intra mode CAE.

#### 7.5.4.14 Contexts for intra-mode CAE for enhancement layer

Depending on the scanning direction, a 7 bit context $C = \sum c_k \cdot 2^k$ is built for each coded pixel as shown in Figure 7-24, where $c_k$=0 for transparent pixels and $c_k$=1 for opaque pixels. (a) of Figure 7-24 is for the intra-mode CAE with V-SI (SI by the vertical scanning), and (b) of Figure 7-24 is for the intra-mode CAE with H-SI (SI by the horizontal scanning). In (a) of the figure, C6, C5 and C4 are the closest decoded pixels in top-left, top, and top-right location, respectively. C3, and C2 are the closest decoded pixels in left and right location, respectively. And C1 and C0 are the closest decoded pixels in bottom-left and bottom-right location, respectively.

In (b) of the figure, C6, C5 and C4 are the closest decoded pixels in top-left, top, and top-right location, respectively. C3 is the closest decoded pixel in left location. And C2, C1 and C0 are the closest decoded pixels in bottom-left, bottom, and bottom-right location, respectively.

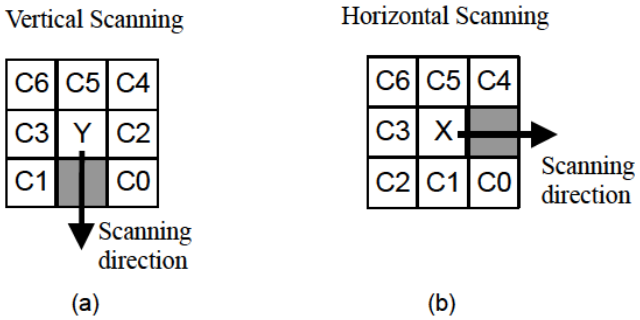



**Figure 7-24 — The context information at different scanning of SI**

© ISO/IEC 2004 – All rights reserved

There are some special cases to consider:

When building contexts, any pixels outside the bounding box of the current VOP to the left and above are assumed to be zero (transparent).

When building contexts and resync_marker_disabled==0, any pixels outside the space of the current video packet to the left and above are assumed to be zero (transparent).

When constructing the context, any pixel outside the bordered BAB is assumed to be the same pixel value of the closest pixel in the border of the current BAB from the context location.

#### 7.5.4.15 Decoding of Inter-mode CAE for enhancement layer

For decoding of enhancement layer with B-VOP coding, the inter-mode CAE is performed in the case of the "inter coded" mode (enh_bab_type==3). The context information in Figure 7-25 is utilized for this decoding. The decoding process of a pixel is done in the same way as non-scalable binary shape coding.

When constructing the Inter-mode CAE context the following conditional assignment is performed.

if(c1 is unknown) c1=c2

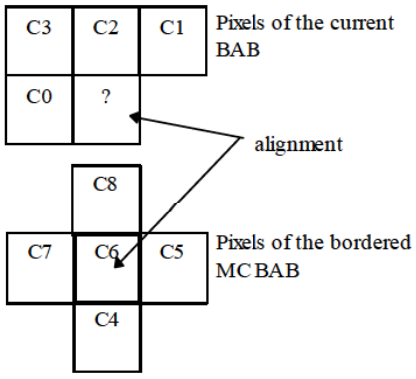



**Figure 7-25 — The templates of Inter-mode CAE for enhancement layer. C6 is aligned with the pixel to be decoded. The pixel to be decoded is marked with '?'**

#### 7.5.4.16 Quality (SNR) Scalable Coding

Quality Scalability can be easily sought by spatial scalability. Using the spatial scalability and spatial prediction, binary shape can achieve the quality (SNR) scalability. In the decoding process, each spatial scalable layer can be spatially predicted to the full resolution. For the binary shape, this scalable structure is equivalent to quality scalable structure.

### 7.5.5 Grayscale Shape Decoding

Grayscale alpha plane decoding is achieved by the separate decoding of a support region and the values of the alpha channel. The support region is transmitted by using the binary shape as described above. The alpha values are transmitted as texture data with arbitrary shape, using almost the same coding method as is used for the luminance texture channel.


© ISO/IEC 2004 – All rights reserved

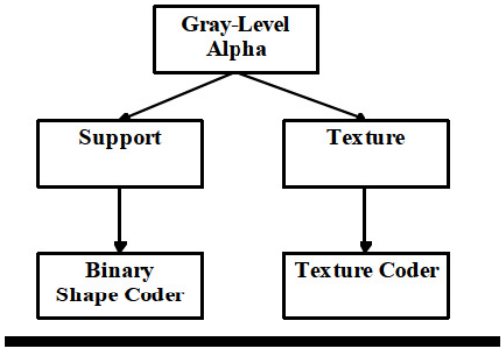



**Figure 7-26 — Grayscale shape coding**

All samples which are indicated to be transparent by the binary shape data, must be set to zero in the decoded grayscale alpha plane. Within the VOP, alpha samples have the values produced by the grayscale alpha decoding process. Decoding of binary shape information is not dependent on the decoding of grayscale alpha. The alpha values are decoded into 16x16 macroblocks in the same way as the luminance channel (see subclauses 7.4 and 7.6, and 7.8.7). The 16x16 blocks of alpha values are referred to as alpha macroblocks hereafter. The data for each alpha macroblock is present in the bitstream immediately following the texture data for the corresponding texture macroblock. Any aspect of alpha decoding that is not covered in this document should be assumed to be the same as for the decoding of luminance.

#### 7.5.5.1 Grayscale Alpha COD Modes

When decoding grayscale alpha macroblocks, CODA is first encountered and indicates the coding status for alpha. It is important to understand that the macroblock syntax elements for alpha are still present in the bitstream for P, B, or S(GMC) macroblocks even if the texture syntax elements indicate "not-coded" (not_coded='1'). In this respect, the decoding of the alpha and texture data are independent. The only exception is for B-VOPs when the colocated P-VOP texture macroblock is skipped. In this case, no syntax is transmitted for texture or grayscale alpha, as both types of macroblock are skipped.

For macroblocks which are completely transparent (indicated by the binary shape coding), no alpha syntax elements are present and the grayscale alpha samples must all be set to zero (transparent). If CODA="all opaque" (I, P, B, or S(GMC) macroblocks) or CODA="not coded" (P, B, or S(GMC) macroblocks) then no more alpha data is present. Otherwise, other alpha syntax elements follow, including the coded block pattern (cbpa), followed by alpha texture data for those 8x8 blocks which are coded and non-transparent, as is the case for regular luminance macroblock texture data.

When CODA="all opaque", the corresponding decoded alpha macroblock is filled with a constant value of 255. This value will be called AlphaOpaqueValue.

#### 7.5.5.2 Alpha Plane Scale Factor

For both binary and grayscale shape, the VOP header syntax element "vop_constant_alpha" can be used to scale the alpha plane. If this bit is equal to '1', then each pixel in the decoded VOP is scaled before output, using vop_constant_alpha_value. The scaling formula is:

scaled_pixel = (original_pixel * (vop_constant_alpha_value + 1) ) / 256

Scaling is applied at the output of the decoder, such that the decoded original values, not the scaled values are used as the source for motion compensation.

© ISO/IEC 2004 – All rights reserved

#### 7.5.5.3 Gray Scale Quantiser Scale

When no_gray_quant_update is equal to "1", the grayscale alpha quantiser scale is fixed for all macroblocks to the value indicated by vop_alpha_quant. Otherwise, the grayscale quantiser scale is reset at each new macroblock to a value that depends on the current texture quantiser scale (after any update by dquant). The relation is:

current_alpha_quant = (current_texture_quant * vop_alpha_quant) / vop_quant

The resulting value of current_alpha_quant must then be clipped so that it never becomes less than 1.

#### 7.5.5.4 Intra Macroblocks

When the texture mb_type indicates an intra macroblock in an I-, P-, or S(GMC)-VOP , the grayscale alpha data is also decoded using intra mode.

The intra DC value is decoded in the same way as for luminance, using the same non-linear transform to convert from alpha_quant to DCScalarA. However, intra_dc_vlc_thr is not used for alpha, and therefore AC coeffiecient VLCs are never used to code the differential intra DC coefficient.

DC prediction is used in the same way as for luminance. However, when coda_i indicates that a macroblock is all opaque, a synthetic intra DC value is created for each block in the current macroblock so that adjacent macroblocks can correctly obtain intra DC prediction values. The synthetic intra DC value is given as:

BlockIntraDC = (((AlphaOpaqueValue * 8) + (DcScalerA>>1)) / DcScalerA) * DcScalerA

AlphaOpaqueValue is described in subclause 7.5.4.1.

The intra cbpa VLC makes use of the *inter* cbpy VLC table, but the intra alpha block DCT coefficients are decoded in the same manner as with luminance intra macroblocks.

When interlaced is equal to "1", alternate scan should not be applied to coding of gray-level alpha.

When both interlaced is equal to "1" and video_object_layer_shape is equal to "11" (grayscale), only the frame DCT should be applied to coding of gray-level alpha.

#### 7.5.5.5 Inter Macroblocks and Motion Compensation

Motion compensation is carried out for P-, B-, and S(GMC)-VOPs, using the 8x8 or 16x16 luminance motion vectors, or the warping of the previous decoded VOP, in the same way as for luminance data, except that regular motion compensation is used instead of OBMC. Forward, backward, bidirectional and direct mode motion compensation are used for B-VOPs. Where the luminance motion vectors are not present because the texture macroblock is skipped, the exact same style of non-coded motion compensation used for luminance is applied to the alpha data (but without OBMC). Note that this does not imply that the alpha macroblock is skipped, because an error signal to update the resulting motion compensated alpha macroblock may still be present if indicated by coda_pb. When the colocated P-VOP texture macroblock is skipped for B-VOPs, then the alpha macroblock is assumed to be skipped with no syntax transmitted.

cbpa and the alpha inter DCT coefficients are decoded in the same way as with luminance cbpy and inter DCT cofficients.

When both interlaced is equal to "1" and video_object_layer_shape is equal to "11" (grayscale), the field padding should be applied to coding of gray-level alpha.

When both interlaced is equal to "1" and video_object_layer_shape is equal to "11" (grayscale), both frame and field motion compensation are applicable in coding of texture and gray-level alpha, but only the frame DCT should be applied to coding of gray-level alpha.


© ISO/IEC 2004 – All rights reserved

#### 7.5.5.6 Method to be used when blending with greyscale alpha signal

The following explains the blending method to be applied to the video object in the compositor, which is controlled by the composition_method flag and the linear_composition flag. The linear_composition flag is informative only, and the decoder may ignore it and proceed as if it had the value 0. However, it is normative that the composition_method flag be acted upon.

The descriptions below show the processing taking place in YUV space; note that the processing can of course be implemented in RGB space to obtain equivalent results.

**composition_method=0 (cross-fading)**

If layer N, with an n-bit alpha signal, is overlaid over layer M to generate a new layer P, the composited Y, U, V and alpha values are:

$$Pyuv = ( (2^n\text{-}1 - Nalpha) * Myuv + (Nalpha * Nyuv ) ) / (2^n\text{-}1)$$

$$Palpha = (2^n\text{-}1)$$

**composition_method=1 (Additive mixing)**

If layer N, with an n-bit alpha signal, is overlaid over layer M to generate a new layer P, the composited Y, U, V and alpha values are:

$$Pyuv = \begin{cases} Myuv & \text{..... } Nalpha = 0 \\ (Myuv - BLACK) - ( (Myuv - BLACK) * Nalpha ) / (2^n\text{-}1) + Nyuv & \text{..... } Nalpha > 0 \end{cases}$$

(this is equivalent to Pyuv = Myuv*(1-alpha) + Nyuv, taking account of black level and the fact that the video decoder does not produce an output in areas where alpha=0)

$$Palpha = Nalpha + Malpha - (Nalpha*Malpha) / (2^n\text{-}1)$$

where

BLACK is the common black value of foreground and background objects.

NOTE The compositor must convert foreground and background objects to the same black value and signal range before composition. The black level of each video object is specified by the video_range bit in the video_signal_type field, or by the default value if the field is not present. (The RGB values of synthetic objects are specified in a range from 0 to 1, as described in ISO/IEC 14496-1:2001).

- linear_composition = 0: The compositing process is carried out using the video signal in the format from which it is produced by the video decoder, that is, without converting to linear signals. Note that because video signals are usually non-linear ("gamma-corrected"), the composition will be approximate.
- linear_composition = 1: The compositing process is carried out using linear signals, so the output of the video decoder is converted to linear if it was originally in a non-linear form, as specified by the video_signal_type field. Note that the alpha signal is always linear, and therefore requires no conversion.

### 7.5.6 Multiple Auxiliary Component Decoding

Auxiliary components are defined for the VOP on a pixel-by-pixel basis, and contain data related to the video object, like disparity, depth, additional texture. Up to 3 auxiliary components (including the grayscale shape) are possible. The number and type of these components is indicated by the video_object_layer_shape_extension given in Table 6-17. For example, a value '0000' indicates the grayscale (alpha) shape. The same support region as described in 7.5.4 is used for all auxiliary components, and the decoding procedure is the same as described in subclauses 7.5.4.1 - 7.5.4.6.

© ISO/IEC 2004 – All rights reserved

### 7.6 Motion compensation decoding

In order to perform motion compensated prediction on a per VOP basis, a special padding technique, i.e. the macroblock-based repetitive padding, is applied for the reference VOP. The details of these techniques are described in the following subclauses.

Since a VOP may have arbitrary shape, and this shape can change from one instance to another, conventions are necessary to ensure the consistency of the motion compensation process.

The absolute (frame) coordinate system is used for referencing every VOP. At every given instance, a bounding rectangle that includes the shape of that VOP, as described in subclause 7.5, is defined. The left and top corner, in the absolute coordinates, of the bounding rectangle is decoded from VOP spatial reference. Thus, the motion vector for a particular feature inside a VOP, e.g. a macroblock, refers to the displacement of the feature in absolute coordinates. No alignment of VOP bounding rectangles at different time instances is performed.

For motion compensation, the amplitude resolution of the motion vectors is signaled by the quarter_sample flag in the VOL header, see subclause 6.2.3. Either ½ pel resolution ("half sample mode", quarter_sample==0) or ¼ pel ("quarter sample mode", quarter_sample==1) is used. For ½ pel resolution, the necessary ½ pel samples are calculated by means of half sample interpolation as described in subclause 7.6.2.1. For ¼ pel resolution, the necessary ½ and ¼ pel samples are calculated by means of half and quarter sample interpolation as described in subclause 7.6.2.2.

In addition to the above motion compensation processing, three additional processes are supported, namely, unrestricted motion compensation, four MV motion compensation,and overlapped motion compensation. Note that in all three modes, macroblock-based padding of the arbitrarily shaped reference VOP is performed for motion compensation.

#### 7.6.1 Padding process

The padding process defines the values of luminance and chrominance samples outside the VOP for prediction of arbitrarily shaped objects. Figure 7-27 shows a simplified diagram of this process.

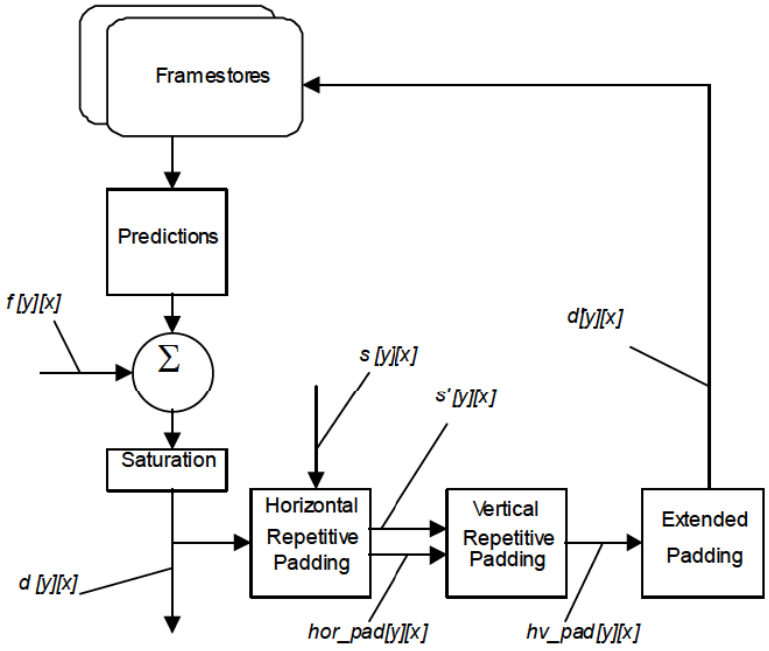



**Figure 7-27 — Simplified padding process**


© ISO/IEC 2004 – All rights reserved

A decoded macroblock *d[y][x]* is padded by referring to the corresponding decoded shape block *s[y][x]*. The luminance component is padded per 16 x 16 samples, while the chrominance components are padded per 8 x 8 samples. A macroblock that lies on the VOP boundary (hereafter referred to as a boundary macroblock) is padded by replicating the boundary samples of the VOP towards the exterior. This process is divided into horizontal repetitive padding and vertical repetitive padding. The remaining macroblocks that are completely outside the VOP (hereafter referred to as exterior macroblocks) are filled by extended padding.

NOTE The padding process is applied to all macroblocks inside the bounding rectangle of a VOP. The bounding rectangle of the luminance component is defined by vop_width and vop_height extended to multiple of 16, while that of the chrominance components is defined by (vop_width>>1) and (vop_height>>1) extended to multiple of 8.

#### 7.6.1.1 Horizontal repetitive padding

Each sample at the boundary of a VOP is replicated horizontally to the left and/or right direction in order to fill the transparent region outside the VOP of a boundary macroblock. If there are two boundary sample values for filling a sample outside of a VOP, the two boundary samples are averaged (//2).

*hor_pad[y][x]* is generated by any process equivalent to the following example. For every line with at least one shape sample *s[y][x]* == 1(inside the VOP):

```
for (x=0; x<N; x++) {
    if (s[y][x] == 1) { hor_pad[y][x] = d[y][x]; s'[y][x] = 1; }
    else {
        if ( s[y][x'] == 1 && s[y][x"] == 1)  {
            hor_pad[y][x] = (d[y][x']+ d[y][x"])//2;
            s'[y][x] = 1;
        } else if ( s[y][x'] == 1 ) {
            hor_pad[y][x] = d[y][x']; s'[y][x] = 1;
        } else if ( s[y][x"] == 1 ) {
            hor_pad[y][x] = d[y][x"]; s'[y][x] = 1;
        }
    }
}
```

where *x'* is the location of the nearest valid sample (*s[y][x']* == 1) at the VOP boundary to the left of the current location *x*, *x"* is the location of the nearest boundary sample to the right, and *N* is the number of samples of a line in a macroblock. *s'[y][x]* is initialized to 0.

#### 7.6.1.2 Vertical repetitive padding

The remaining unfilled transparent horizontal samples (where *s'[y][x]* == 0) from subclause 7.6.1.1 are padded by a similar process as the horizontal repetitive padding but in the *vertical* direction. The samples already filled in subclause 7.6.1.1 are treated as if they were inside the VOP for the purpose of this vertical pass.

*hv_pad[y][x]* is generated by any process equivalent to the following example. For every column of *hor_pad[y][x]* :

```
for (y=0; y<M; y++) {
    if (s'[y][x] == 1)
        hv_pad[y][x] =hor_pad[y][x];
    else {
        if ( s'[y'][x] == 1 && s'[y"][x] == 1 )
            hv_pad[y][x] = (hor_pad[y'][x] + hor_pad[y"][x])//2;
        else if ( s'[y'][x] == 1 )
            hv_pad[y][x] = hor_pad[y'][x];
        else if (s'[y"][x] == 1 )
            hv_pad[y][x] = hor_pad[y"][x];
    }
}
```

where *y'* is the location of the nearest valid sample (*s'[y'][x]* == 1) above the current location *y* at the boundary of *hv_pad*, *y"* is the location of the nearest boundary sample below *y*, and *M* is the number of samples of a column in a macroblock.

© ISO/IEC 2004 – All rights reserved

#### 7.6.1.3 Extended padding

Exterior macroblocks immediately next to boundary macroblocks are filled by replicating the samples at the border of the boundary macroblocks. Note that the boundary macroblocks have been completely padded in subclause 7.6.1.1 and subclause 7.6.1.2. If an exterior macroblock is next to more than one boundary macroblocks, one of the macroblocks is chosen, according to the following convention, for reference.

The boundary macroblocks surrounding an exterior macroblock are numbered in priority according to Figure 7-28. The exterior macroblock is then padded by replicating upwards, downwards, leftwards, or rightwards the row of samples from the horizontal or vertical border of the boundary macroblock having the largest priority number.

The remaining exterior macroblocks (not located next to any boundary macroblocks) are filled with $2^{bits\_per\_pixel-1}$. For 8-bit luminance component and associated chrominance this implies filling with 128.

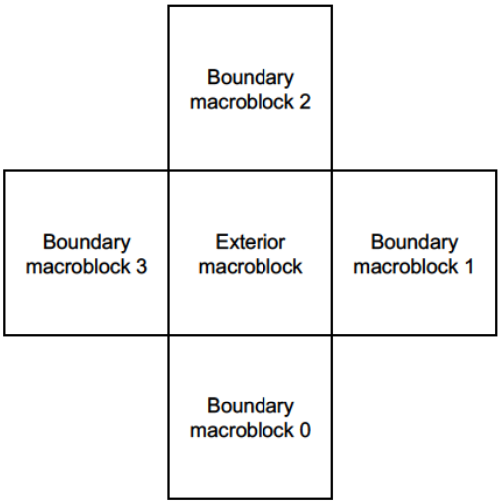



**Figure 7-28 — Priority of boundary macroblocks surrounding an exterior macroblock**

#### 7.6.1.4 Padding for chrominance components

Chrominance components are padded according to subclauses 7.6.1.1 through 7.6.1.3 for each 8 x 8 block. The padding is performed by referring to a shape block generated by decimating the shape block of the corresponding luminance component. This decimating of the shape block is performed by the subsampling process described in subclause 6.1.3.6.

#### 7.6.1.5 Padding of interlaced macroblocks

Macroblocks of interlaced VOP (interlaced = 1) are padded according to subclauses 7.6.1.1 through 7.6.1.3. The vertical padding of the luminance component, however, is performed for each field independently. A sample outside of a VOP is therefore filled with the value of the nearest boundary sample of the same field. Completely transparent blocks are padded with $2^{bits\_per\_pixel-1}$. Chrominance components of interlaced VOP are padded according to subclause 7.6.1.4, however, based on fields to enhance subjective quality of display in 4:2:0 format. The padding method described in this subclause is not used outside the bounding rectangle of the VOP.

#### 7.6.1.6 Vector padding technique

The vector padding technique is applied to generate the vectors for the transparent blocks within a non-transparent macroblock, for an INTRA-coded macroblock and for a skipped macroblock included in a P-VOP. It works in a similar way as the horizontal followed by the vertical repetitive padding, and can be regarded as the repetitive padding performed on a 2x2 block except that the padded values are two dimensional vectors. A macroblock has


© ISO/IEC 2004 – All rights reserved

four 8x8 luminance blocks, let {MVx[i], MVy[i], i=0,1,2,3} and {Transp[i], i=0,1,2,3} be the vectors and the transparencies of the four 8x8 blocks, respectively, the vector padding is any process numerically equivalent to:

```
if (the macroblock is INTRA-coded, or skipped and included in a P-VOP) {
    MVx[0] = MVx[1] = MVx[2] = MVx[3] = 0
    MVy[0] = MVy[1] = MVy[2] = MVy[3] = 0
} else {
    if(Transp[0] == TRANSPARENT) {
        MVx[0]=(Transp[1] != TRANSPARENT) ? MVx[1] :((Transp[2]!=TRANSPARENT) ?
        MVx[2]:MVx[3]));
        MVy[0]=(Transp[1] != TRANSPARENT) ? MVy[1]:((Transp[2]!=TRANSPARENT) ?
        MVy[2]:MVy[3]));
    }
if(Transp[1] == TRANSPARENT) {
    MVx[1]=(Transp[0] != TRANSPARENT) ? MVx[0] :((Transp[3]!=TRANSPARENT) ?     MVx[3]:MVx[2]));
    MVy[1]=(Transp[0] != TRANSPARENT) ? MVy[0]:((Transp[3]!=TRANSPARENT) ?     MVy[3]:MVy[2]));
}
if(Transp[2] == TRANSPARENT) {
    MVx[2]=(Transp[3] != TRANSPARENT) ? MVx[3] :((Transp[0]!=TRANSPARENT) ?     MVx[0]:MVx[1]));
    MVy[2]=(Transp[3] != TRANSPARENT) ? MVy[3]:((Transp[0]!=TRANSPARENT) ?     MVy[0]:MVy[1]));
}
if(Transp[3] == TRANSPARENT) {
    MVx[3]=(Transp[2] != TRANSPARENT) ? MVx[2] :((Transp[1]!=TRANSPARENT) ?     MVx[1]:MVx[0]));
    MVy[3]=(Transp[2] !=TRANSPARENT) ? MVy[2]:((Transp[1]!=TRANSPARENT) ?  MVy[1]:MVy[0]));
    }
}
```

Vector padding is only used in I-, P-, and S(GMC)-VOPs, it is applied on a macroblock directly after it is decoded. The block vectors after padding are used in the P-, and S(GMC)-VOP vector decoding and binary shape decoding, and in the B-VOP direct mode decoding. Note that the averaged motion vector described in 7.8.7.3 is used as the motion vectors of a GMC macroblock (i.e. a macroblock included in an S (GMC)-VOP and mcsel == '1') for this padding process.

### 7.6.2 Sample interpolation for non-integer motion vectors

The sample interpolation in case of non-integer motion vectors is performed for 16x16 or 8x8 frame blocks in case of progressive and for 16x8 field blocks in case of interlaced macroblocks. It is done using the luminance and chrominance components of the respectively reconstructed and padded reference VOP.

The value for the quarter_sample flag is defined in the VOL header (see subclause 6.2.4), the value of rounding_control is defined using the vop_rounding_type bit in the vop header (see subclause 6.3.5). Note that the samples outside the padded region cannot be used for interpolation. For interlaced video, the half and quarter sample values are vertically interpolated between two successive lines of the same field. The field motion vectors are given in frame coordinates; that is the vertical coordinates of the integer samples differ by 2.

#### 7.6.2.1 Half sample mode interpolation

The process for interpolation of half sample values defined in this subclause is carried out only in half sample mode, i.e. when quarter_sample==0. Here, the half sample values are calculated by bilinear interpolation as depicted in Figure 7-29. The value of rounding_control is defined using the vop_rounding_type bit in the VOP header (see subclause 6.3.5). Note that the samples outside the padded region cannot be used for interpolation.

© ISO/IEC 2004 – All rights reserved

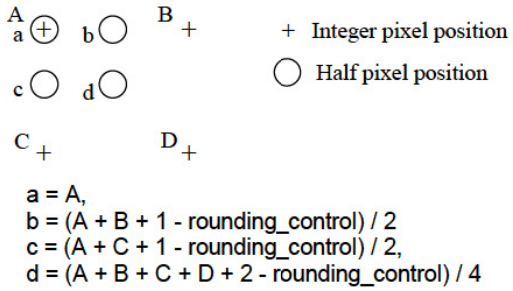



**Figure 7-29 — Interpolation scheme for half sample search**

#### 7.6.2.2 Quarter sample mode interpolation

The process for interpolation of half and quarter sample values defined in this subclause is carried out only in quarter sample mode, i.e. when quarter_sample==1. Further, the process is only carried out for the luminance component of a VOP; for motion compensation of the chrominance components, the respective motion vectors are rounded to the next half pel position (see Table 7-13) and then processed as described in half pel sample interpolation (subclause 7.6.2.1).

In quarter sample mode interpolation, for each block of size MxN in the reference VOP which position is defined by the decoded motion vector for the block to be predicted, a reference block of size (M+1)x(N+1) biased in the direction of the half or quarter sample position is read from the reconstructed and padded reference VOP. Then, this reference block is symmetrically extended at the block boundaries by three samples using block boundary mirroring according to Figure 7-30.

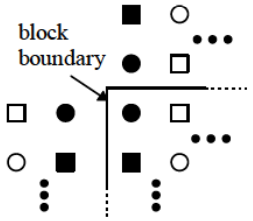



**Figure 7-30 — block boundary mirroring**

The half sample values and (if applicable) the quarter sample values are then calculated from this extended reference block in two steps described in the following.

##### 7.6.2.2.1 Calculation of the half sample values

Figure 7-31 shows integer and half sample positions. The half sample values are calculated by horizontal filtering to obtain values at positions $b_i$ followed by vertical filtering to obtain the values at positions c and d. For the filtering in both directions an 8-tap FIR interpolation filter described by

(C[4], C[3], C[2], C[1], C[1], C[2], C[3], C[4])/256

where C[1..4]= [160, -48, 24, -8] is used. The equations for interpolation of half sample positions are

$$a = A_{-1,-1},$$

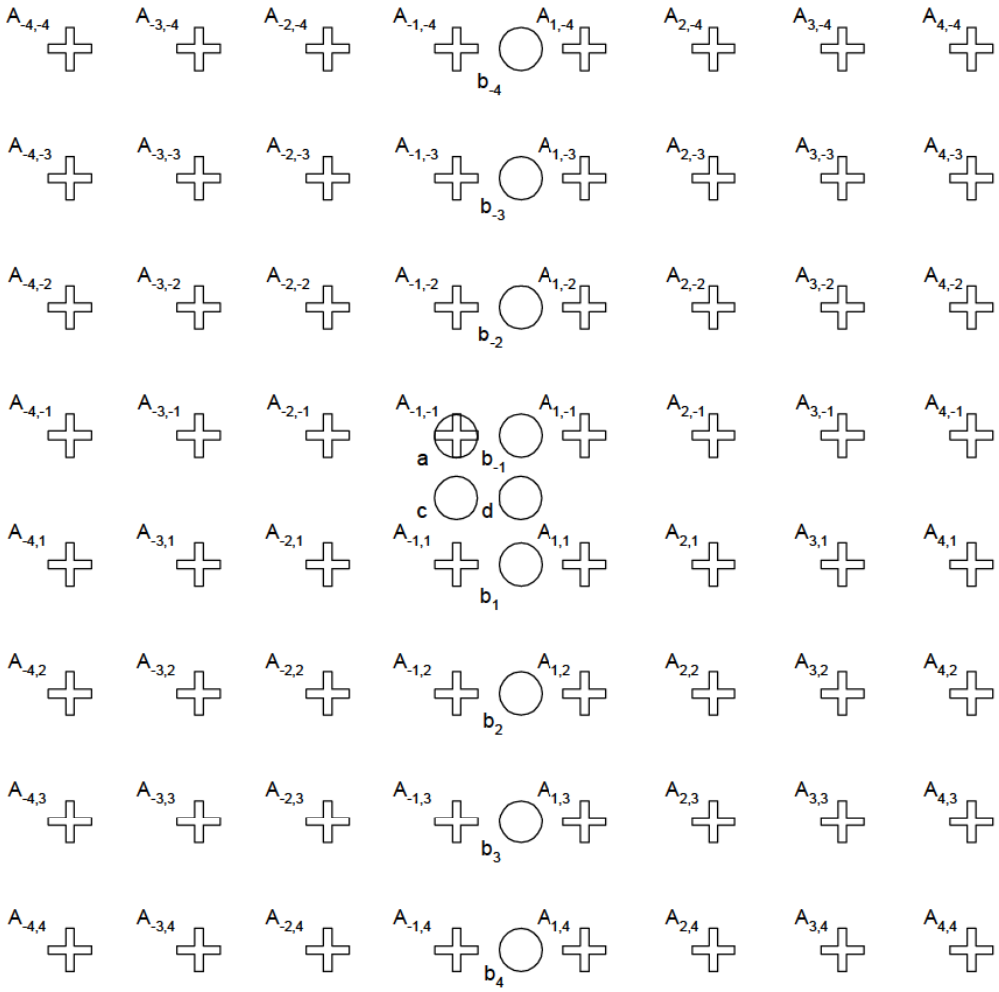

© ISO/IEC 2004 – All rights reserved

$$b_i = \left( \left( \sum_{j=1}^{4} C[j] \cdot (A_{-j,i} + A_{+j,i}) \right) + 128 - rounding\_control \right) / 256,$$

$$c = \left( \left( \sum_{j=1}^{4} C[j] \cdot (A_{-1,-j} + A_{-1,+j}) \right) + 128 - rounding\_control \right) / 256,$$

and

$$d = \left( \left( \sum_{j=1}^{4} C[j] \cdot (b_{-j} + b_{+j}) \right) + 128 - rounding\_control \right) / 256.$$




+: Integer sample position; o: Half sample position

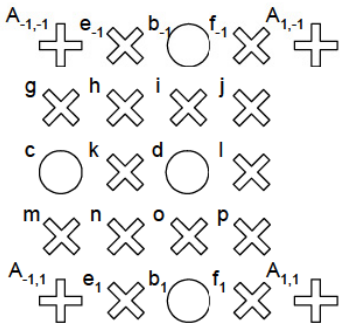
**Figure 7-31 — Half sample interpolation in quarter sample mode (FIR filter)**

© ISO/IEC 2004 – All rights reserved

The values generated by both horizontal and vertical filtering pass are clipped to the range of [0, $2^{bits\_per\_pixel}$-1]. The value of rounding_control is defined using the vop_rounding_type bit in the vop header (see subclause 6.3.5).

#### 7.6.2.2.2 Calculation of the quarter sample values

If applicable, the quarter sample values are calculated in a second step following the half sample interpolation described above. Figure 7-32 shows integer, half, and quarter sample positions.




+: Integer sample position; o: Half sample position; x: Quarter sample position

**Figure 7-32 — Quarter sample interpolation in quarter sample mode (FIR and bilinear filter)**

The quarter sample values are approximately bilinear interpolations of the half sample positions. However, because of the nonlinear operations of rounding and clipping, the interpolation is not perfectly bilinear. The interpolation shall be accomplished by any method equivalent to the following equations:

$$e_i = \left(A_{-1,i} + b_i + 1 - rounding\_control\right)/2$$

$$f_i = \left(b_i + A_{1,i} + 1 - rounding\_control\right)/2$$

$$k = \left(\sum_{j=1}^{4} C[j]\cdot(e_{-j} + e_{+j}) + 128 - rounding\_control\right)/256$$

$$l = \left(\sum_{j=1}^{4} C[j]\cdot(f_{-j} + f_{+j}) + 128 - rounding\_control\right)/256$$

$$g = \left(A_{-1,-1} + c + 1 - rounding\_control\right)/2$$

$$h = \left(e_{-1} + k + 1 - rounding\_control\right)/2$$

$$i = \left(b_{-1} + d + 1 - rounding\_control\right)/2$$

$$j = \left(l + f_{-1} + 1 - rounding\_control\right)/2$$

$$m = \left(c + A_{-1,1} + 1 - rounding\_control\right)/2$$

$$n = \left(k + e_1 + 1 - rounding\_control\right)/2$$


© ISO/IEC 2004 – All rights reserved

$$o = (d + b_1 + 1 - rounding\_control)/2$$

$$p = (l + f_1 + 1 - rounding\_control)/2$$

The values computed are clipped to the range of [0, $2^{bits\_per\_pixel}$-1]. The value of rounding_control affects the direction of rounding.

### 7.6.3 General motion vector decoding process

To decode a motion vector (MVx, MVy), the differential motion vector (MVDx, MVDy) shall be extracted from the bitstream by using the variable length decoding as described in subclause 6.3.6.2. Then it shall be added component-wise to a motion vector predictor (Px, Py) to form the final motion vector. The general motion vector decoding process shall be any process that is equivalent to the following one:

All calculations are carried out as unitless integers. If quarter_sample==0, then the integer values represent halfsample units; if quarter_sample==1, then the integer values represent quartersample units. This process is valid for the motion vector decoding in interlaced/progressive P-, S(GMC)- and B-VOPs except that the generation of the predictor (Px, Py) may be different.

```
r_size = vop_fcode - 1
f = 1 << r_size
high = ( 32 * f ) -  1;
low = ( (-32) * f );
range = ( 64 * f );

if ( (f  == 1) || (horizontal_mv_data == 0) )
    MVDx = horizontal_mv_data;
else {
    MVDx = ( ( Abs(horizontal_mv_data) - 1 ) * f ) + horizontal_mv_residual + 1;
    if (horizontal_mv_data < 0)
        MVDx = -MVDx;
}

if ( (f  == 1) || (vertical_mv_data == 0) )
    MVDy = vertical_mv_data;
else {
    MVDy = ( ( Abs(vertical_mv_data) - 1 ) * f ) + vertical_mv_residual + 1;
    if (vertical_mv_data < 0)
        MVDy = -MVDy;
}

MVx = Px + MVDx;
if ( MVx < low )
    MVx = MVx + range;
if (MVx > high)
    MVx = MVx - range;

MVy = Py + MVDy;
if ( MVy < low )
    MVy = MVy + range;
if (MVy > high)
    MVy = MVy - range;
```

The value of mv_data (i.e., horizontal_mv_data and vertical_mv_data) is equal to two times the value found in the 'vector differences' column of Table B.12 associated with the received codeword.

The parameters in the bitstream shall be such that the components of the reconstructed differential motion vector, MVDx and MVDy, shall lie in the range [low:high], at the time of their use in calculating the values of MVx and MVy

© ISO/IEC 2004 – All rights reserved

(i.e., intermediate values of MVDx and MVDy may occur that are outside the range [low:high]). In addition the components of the reconstructed motion vector, *MVx* and *MVy,* shall also lie in the range [*low* : *high*]. The allowed range [low : high] for the motion vectors depends on the parameter vop_fcode; it is shown in Table 7-9.

The variables *r_size*, *f*, *MVDx, MVDy*, *high*, *low* and *range* are temporary variables that are not used in the remainder of this part of ISO/IEC 14496. The parameters horizontal_mv_data, vertical_mv_data, horizontal_mv_residual and vertical_mv_residual are parameters recovered from the bitstream.

The variable *vop_fcode* refers either to the parameter vop_fcode_forward or to the parameter vop_fcode_backward which have been recovered from the bitstream, depending on the respective prediction mode. In the case of P- or S(GMC)-VOP prediction only forward prediction applies. In the case of B-VOP prediction, forward as well as backward prediction may apply.

**Table 7-9 — Range for motion vectors**

| vop_fcode_forward or vop_fcode_backward | motion vector range in unitless integers [low:high] |
|---|---|
| 1 | [-32,31] |
| 2 | [-64,63] |
| 3 | [-128,127] |
| 4 | [-256,255] |
| 5 | [-512,511] |
| 6 | [-1024,1023] |
| 7 | [-2048,2047] |

If the current macroblock is a field motion compensated macroblock, then the same prediction motion vector (Px, Py) is used for both field motion vectors. Because the vertical component of a field motion vector is an even integer, the vertical differential motion vector encoded in the bitstream is

$$MVy = MVDy_{field} + PY / 2.$$

### 7.6.4 Unrestricted motion compensation

Motion vectors are allowed to point outside the decoded area of a reference VOP when (and only when) the short video header format is not in use (i.e., when short_video_header is 0). For an arbitrary shape VOP, the decoded area refers to the area within the bounding rectangle, padded as described in subclause 7.6.1. A bounding rectangle is defined by vop_width and vop_height extended to multiple of 16. In case of half sample mode, when a sample referenced by a motion vector stays outside the decoded VOP area, an edge sample is used. This edge sample is retrieved by limiting the motion vector to the last full pel position inside the decoded VOP area. Limitation of a motion vector is performed on a sample basis and separately for each component of the motion vector, as depicted in Figure 7-33. In case of quarter sample mode, when a sample needed for interpolation (see subclause 7.6.2.2) stays outside the decoded VOP area, an edge sample is used prior to block boundary mirroring.


© ISO/IEC 2004 – All rights reserved

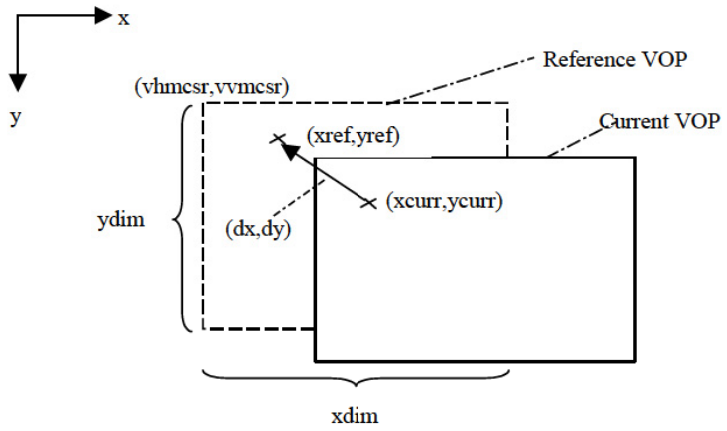



**Figure 7-33 — Unrestricted motion compensation**

The coordinates of a reference sample in the reference VOP, (yref, xref) is determined as follows :

$$xref = MIN ( MAX (xcurr+dx, vhmcsr), xdim+vhmcsr-1 )$$

$$yref = MIN ( MAX (ycurr+dy, vvmcsr), ydim+vvmcsr-1)$$

where vhmcsr = vop_horizontal_mc_spatial_ref, vvmcsr = vop_vertical_mc_spatial_ref, (ycurr, xcurr) are the coordinates of a sample in the current VOP, (yref, xref) are the coordinates of a sample in the reference VOP, (dy, dx) is the motion vector, and (ydim, xdim) are the dimensions of the bounding rectangle of the reference VOP. All coordinates are related to the absolute coordinate system shown in Figure 7-33. Note that for rectangular VOP, a reference VOP is defined by video_object_layer_width and video_object_layer_height, extended to a multiple of 16. For an arbitrary shape VOP, a reference VOP of luminance is defined by vop_width and vop_height extended to a multiple of 16, while that of chrominance is defined by (vop_width>>1) and (vop_height>>1) extended to a multiple of 8.

### 7.6.5 Vector decoding processing and motion-compensation in progressive P- and S(GMC)-VOP

An inter-coded macroblock comprises either one motion vector for the complete macroblock or K ( 1< K<=4) motion vectors, one for each non-transparent 8x8 pel blocks forming the 16x16 pel macroblock, as is indicated by the mcbpc code.

For decoding a motion vector, the horizontal and vertical motion vector components are decoded differentially by using a prediction, which is formed by a median filtering of three vector candidate predictors (MV1, MV2, MV3) from the spatial neighbourhood macroblocks or blocks already decoded. Note that if any spatial neighbourhood macroblock is a GMC macroblock (i.e. included in an S (GMC)-VOP and mcsel == '1') and it is in the current VOP and video packet, the averaged motion vector described in subclause 7.8.7.3 is used as candidate motion vectors. The spatial position of candidate predictors for each block vector is depicted in Figure 7-34. In the case of only one motion vector present for the complete macroblock, the top-left case in Figure 7-34 is applied. When the short video header format is in use (i.e., when short_video_header is "1"), only one motion vector shall be present for a macroblock.

© ISO/IEC 2004 – All rights reserved

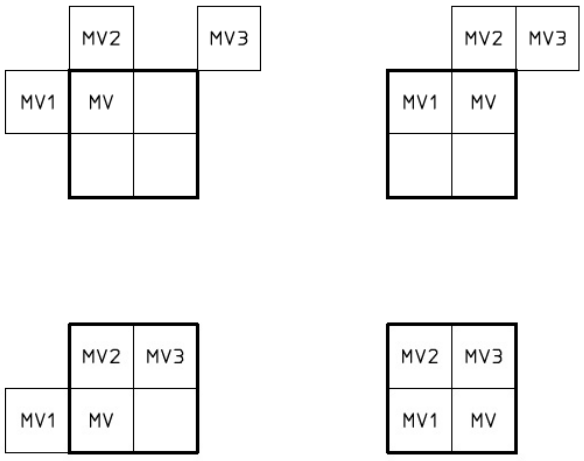



**Figure 7-34 — Definition of the candidate predictors MV1, MV2 and MV3 for each of the luminance blocks in a macroblock**

The following four decision rules are applied to obtain the value of the three candidate predictors:

- If a candidate predictor MVi is in a transparent spatial neighbourhood macroblock or in a transparent block of the current macroblock it is not valid, otherwise, it is set to the corresponding block vector.
- If one and only one candidate predictor is not valid, it is set to zero.
- If two and only two candidate predictors are not valid, they are set to the third candidate predictor.
- If all three candidate predictors are not valid, they are set to zero.

Note that any neighbourhood macroblock outside the current VOP or video packet or outside the current GOB (when short_video_header is "1") for which gob_header_empty is "0" is treated as transparent in the above sense. The median value of the three candidates for the same component is computed as predictor, denoted by Px and Py:

$$Px = Median(MV1x, MV2x, MV3x)$$
$$Py = Median(MV1y, MV2y, MV3y)$$

For instance, if MV1=(-2,3), MV2=(1,5) and MV3=(-1,7), then Px = -1 and Py = 5. The final motion vector is then obtained by using the general decoding process defined in the subclause 7.6.3.

If four vectors are used, each of the motion vectors is used for all pixels in one of the four luminance blocks in the macroblock. The numbering of the motion vectors is equivalent to the numbering of the four luminance blocks as given in Figure 6-8. Motion vector $MVD_{CHR}$ for both chrominance blocks is derived by calculating the sum of the *K* luminance vectors, that corresponds to *K* 8x8 blocks that do not lie outside the VOP shape and dividing this sum by 2**K;* in quarter sample mode the vectors are divided by 2 before summation. The component values of the resulting sixteenth/twelfth/eighth/fourth sample resolution vectors are modified towards the nearest half sample position as indicated below.


© ISO/IEC 2004 – All rights reserved

**Table 7-10 — Modification of sixteenth sample resolution chrominance vector components**

| sixteenth pixel position | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | //16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| resulting position | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | //2 |

**Table 7-11 — Modification of twelfth sample resolution chrominance vector components**

| twelfth pixel position | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | //12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| resulting position | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | //2 |

**Table 7-12 — Modification of eighth sample resolution chrominance vector components**

| eighth pixel position | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | //8 |
|---|---|---|---|---|---|---|---|---|---|
| resulting position | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 2 | //2 |

**Table 7-13 — Modification of fourth sample resolution chrominance vector components**

| fourth pixel position | 0 | 1 | 2 | 3 | //4 |
|---|---|---|---|---|---|
| resulting position | 0 | 1 | 1 | 1 | //2 |

Half sample values are found using bilinear interpolation as described in subclause 7.6.2. The prediction for luminance is obtained by overlapped motion compensation as described in subclause 7.6.6 if indicated by obmc_disable==0. The prediction for chrominance is obtained by applying the motion vector $MVD_{CHR}$ to all pixels in the two chrominance blocks.

### 7.6.6 Overlapped motion compensation

This subclause specifies the overlapped motion compensation process. This process is performed when the flag obmc_disable=0. Note that overlapped motion compensation is disabled over the boundary between macroblocks with different values for mcsel in an S (GMC)-VOP.

Each pixel in an 8*8 luminance prediction block is a weighted sum of three prediction values, divided by 8 (with rounding). In order to obtain the three prediction values, three motion vectors are used: the motion vector of the current luminance block, and two out of four "remote" vectors:

- the motion vector of the block at the left or right side of the current luminance block;
- the motion vector of the block above or below the current luminance block.

For each pixel, the remote motion vectors of the blocks at the two nearest block borders are used. This means that for the upper half of the block the motion vector corresponding to the block above the current block is used, while for the lower half of the block the motion vector corresponding to the block below the current block is used. Similarly, for the left half of the block the motion vector corresponding to the block at the left side of the current block is used, while for the right half of the block the motion vector corresponding to the block at the right side of the current block is used.

© ISO/IEC 2004 – All rights reserved

The creation of each pixel, $\overline{p}(i,j)$, in an 8*8 luminance prediction block is governed by the following equation:

$$\overline{p}(i,j) = (q(i,j) \times H_0(i,j) + r(i,j) \times H_1(i,j) + s(i,j) \times H_2(i,j) + 4) \,//\, 8,$$

where $q(i,j)$, $r(i,j)$, and $s(i,j)$ are the pixels from the referenced picture as defined by

$$q(i,j) = p(i + MV_x^0, j + MV_y^0),$$
$$r(i,j) = p(i + MV_x^1, j + MV_y^1),$$
$$s(i,j) = p(i + MV_x^2, j + MV_y^2).$$

Here, $(MV_x^0, MV_y^0)$ denotes the motion vector for the current block, $(MV_x^1, MV_y^1)$ denotes the motion vector of the block either above or below, and $(MV_x^2, MV_y^2)$ denotes the motion vector either to the left or right of the current block as defined above.

The matrices $H_0(i,j), H_1(i,j)$ and $H_2(i,j)$ are defined in Figure 7-35, Figure 7-36, and Figure 7-37, where $(i,j)$ denotes the column and row, respectively, of the matrix.

If one of the surrounding blocks was not coded, the corresponding remote motion vector is set to zero. If one of the surrounding blocks was coded in intra mode, the corresponding remote motion vector is replaced by the motion vector for the current block. If the current block is at the border of the VOP and therefore a surrounding block is not present, the corresponding remote motion vector is replaced by the current motion vector. In addition, if the current block is at the bottom of the macroblock, the remote motion vector corresponding with an 8*8 luminance block in the macroblock below the current macroblock is replaced by the motion vector for the current block.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4 | 5 | 5 | 5 | 5 | 5 | 5 | 4 |
| 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 5 | 5 | 6 | 6 | 6 | 6 | 5 | 5 |
| 5 | 5 | 6 | 6 | 6 | 6 | 5 | 5 |
| 5 | 5 | 6 | 6 | 6 | 6 | 5 | 5 |
| 5 | 5 | 6 | 6 | 6 | 6 | 5 | 5 |
| 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 4 | 5 | 5 | 5 | 5 | 5 | 5 | 4 |

**Figure 7-35 — Weighting values, $H_0$, for prediction with motion vector of current luminance block**


© ISO/IEC 2004 – All rights reserved

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 1 | 1 | 2 | 2 | 2 | 2 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 2 | 2 | 2 | 1 | 1 |
| 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |

**Figure 7-36 — Weighting values, H1, for prediction with motion vectors of the luminance blocks on top or bottom of current luminance block**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | 1 | 1 | 1 | 1 | 1 | 1 | 2 |
| 2 | 2 | 1 | 1 | 1 | 1 | 2 | 2 |
| 2 | 2 | 1 | 1 | 1 | 1 | 2 | 2 |
| 2 | 2 | 1 | 1 | 1 | 1 | 2 | 2 |
| 2 | 2 | 1 | 1 | 1 | 1 | 2 | 2 |
| 2 | 2 | 1 | 1 | 1 | 1 | 2 | 2 |
| 2 | 2 | 1 | 1 | 1 | 1 | 2 | 2 |
| 2 | 1 | 1 | 1 | 1 | 1 | 1 | 2 |

**Figure 7-37 — Weighting values, $H_2$, for prediction with motion vectors of the luminance blocks to the left or right of current luminance block**

### 7.6.7 Temporal prediction structure

A forward reference VOP is defined as a most recently decoded I- or P- or S(GMC)-VOP in the past for which "vop_coded==1". A backward reference VOP is defined as the most recently decoded I- or P- or S(GMC)-VOP in the future, regardless of its value for "vop_coded".

A target P- or S(GMC)-VOP shall make reference to the forward reference VOP

A target B-VOP can make reference

- to the forward and/or the backward reference VOP, if for the backward reference VOP "vop_coded==1"
- only to the forward reference VOP, if for the backward reference VOP "vop_coded==0"

Note that for the reference VOP selection of binary shape coding the rules stated in subclause 7.5.2.4 shall be applied

The temporal prediction structure is depicted in Figure 7-38.

© ISO/IEC 2004 – All rights reserved