INTERNATIONAL STANDARD

ISO/IEC 14496-2

Second edition
2001-12-01

AMENDMENT 1
2002-02-01

Information technology — Coding of audio-visual objects —

Part 2: **Visual**

AMENDMENT 1: Studio profile

Technologies de l'information — Codage des objets audiovisuels —

Partie 2: Codage visuel

AMENDEMENT 1: Profil du studio

Reference number
ISO/IEC 14496-2:2001/Amd.1:2002(E)

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this Amendment may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Amendment 1 to International Standard ISO/IEC 14496-2:2001 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

Information technology — Coding of audio-visual objects — Part 2: Visual

AMENDMENT 1: Studio profile

1) Add the following text at the end of 'Overview of the object based non scalable syntax' of 'Introduction':

"

In order to preserve the lossless quality, or to restrict the maximum bit count of block data, the block based DPCM coding can be used for ISO/IEC 14496-2:2001 Amendment 1 (Studio Profile Amendment).

"

2) Replace text in 'Coding of Shapes' of 'Introduction',

"

In natural video scenes, VOPs are generated by segmentation of the scene according to some semantic meaning. For such scenes, the shape information is thus binary (binary shape). Shape information is also referred to as alpha plane. The binary alpha plane is coded on a macroblock basis by a coder which uses the context information, motion compensation and arithmetic coding.

"

with

"

In natural video scenes, VOPs are generated by segmentation of the scene according to some semantic meaning. For such scenes, the shape information is thus binary (binary shape). Shape information is also referred to as alpha plane. The binary alpha plane is coded on a macroblock basis by a coder which uses the context information, motion compensation and arithmetic coding. For high quality applications, the uncompressed binary alpha block coding is used.

"

3) Add the following text in 'Introduction' following 'Coding of Shapes':

"

Coding interlaced video

Each frame of interlaced video consists of two fields which are separated by one field-period. This part of ISO/IEC 14496 allows either the frame to be encoded as a VOP or the two fields to be encoded as two VOPs. Frame encoding or field encoding can be adaptively selected on a frame-by-frame basis. Frame encoding is typically preferred when the video scene contains significant detail with limited motion. Field encoding, in which the second field can be predicted from the first, works better when there is fast movement.

"

4) Replace text in ‘Motion representation - macroblocks' of ‘Introduction’,

"

The choice of 16×16 blocks (referred to as macroblocks) for the motion-compensation unit is a result of the trade-off between the coding gain provided by using motion information and the overhead needed to represent it. Each macroblock can further be subdivided to 8×8 blocks for motion estimation and compensation depending on the overhead that can be afforded. In order to encode the highly active scene with higher vop rate, a Reduced Resolution VOP tool is provided. When this tool is used , the size of the macroblock used for motion compensation decoding is 32 x 32 pixels and the size of block is 16 x 16 pixels.

"

with

"

The choice of 16×16 blocks (referred to as macroblocks) for the motion-compensation unit is a result of the trade-off between the coding gain provided by using motion information and the overhead needed to represent it. Each macroblock can further be subdivided to 8×8 blocks for motion estimation and compensation depending on the overhead that can be afforded. In order to encode the highly active scene with higher vop rate, a Reduced Resolution VOP tool is provided. When this tool is used , the size of the macroblock used for motion compensation decoding is 32 x 32 pixels and the size of block is 16 x 16 pixels.

In frame encoding, the prediction from the previous reference frame can itself be either frame-based or field-based.

"

5) Replace text in ‘Chrominance formats’ of ‘Introduction’,

"

This part of ISO/IEC 14496 currently supports the 4:2:0 chrominance format.

"

with

"

This part of ISO/IEC 14496 currently supports the 4:2:0 chrominance format.

ISO/IEC 14496-2:2001 Amendment 1 also supports the 4:2:2 and 4:4:4 choromminance formats in addition.

"

6) Add the following text in ‘Introduction’ following ‘Chrominance formats’:

"

RGB color components

ISO/IEC 14496-2:2001 Amendment 1 supports coding of RGB color components. The resolution of each component shall be identical when input data is treated as RGB color components.

7) Add the following text at the end of ‘Pixel depth’ of ‘Introduction’:

"

ISO/IEC 14496-2:2001 Amendment 1 supports 8, 10 and 12 bits in luminance and chrominance or RGB planes.

"

8) Replace subclauses 3.38, 3.82, 3.107, and 3.131 with the following:

"

3.38 **component**: A matrix, block or single sample from one of the three matrices (luminance and two chrominance or green, blue and red color primaries) that make up a picture.

3.82 **frame**: A frame contains lines of spatial information of a video signal. For progressive video, these lines contain samples starting from one time instant and continuing through successive lines to the bottom of the frame. For interlaced video a frame consists of two fields, a top field and a bottom field. One of these fields will commence one field period later than the other.

3.107 **macroblock**: The four 8×8 blocks of luminance data and the two (for 4:2:0 chrominance format), four (for 4:2:2 chrominance format) or eight (for 4:4:4 chrominance format) corresponding 8×8 blocks of chrominance data coming from a 16×16 section of the luminance component of the picture. Macroblock is sometimes used to refer to the sample data and sometimes to the coded representation of the sample values and other data elements defined in the macroblock header of the syntax defined in this part of ISO/IEC 14496. The usage is clear from the context.

3.131 **picture**: Source, coded or reconstructed image data. A source or reconstructed picture consists of three rectangular matrices of N-bit numbers representing the luminance and two chrominance signals or rgb colour signals. A “coded VOP” was defined earlier. For progressive video, a picture is identical to a frame, while for interlaced video, a picture can refer to a frame, or the top field or the bottom field of the frame depending on the context.

"

9) Add the following subclauses in clause 3 and renumber the subsequent items.

"

3.6 **B-field VOP**: A field structure B-VOP.

3.7 **B-frame VOP**: A frame structure B-VOP.

3.20 **bottom field**: One of two fields that comprise a frame. Each line of a bottom field is spatially located immediately below the corresponding line of the top field.

3.33 **coded B-frame**: A B-frame VOP or a pair of B-field VOPs that is coded.

3.34 **coded frame**: A coded frame is a coded I-frame, a coded P-frame or a coded B-frame.

3.35 **coded I-frame**: An I-frame VOP or a pair of field VOPs that is coded where the first field VOP is an I-VOP and the second field VOP is an I-VOP or a P-VOP..

3.36 **coded P-frame**: A P-frame VOP or a pair of field VOPs that is coded.

3.42 **coded order**: The order in which the VOPs are transmitted and decoded. This order is not necessarily the same as the display order.

3.64 **display aspect ratio**: The ratio height/width (in spatial measurement units such as centimeters) of the intended display.

3.66 **display process**: The (non-normative) process by which reconstructed frames are displayed.

3.85 **fast forward playback**: The process of displaying a sequence, or parts of a sequence, of VOPs in display-order, faster than real-time.

3.86 **fast reverse playback**: The process of displaying a sequence, or parts of a sequence, of VOPs in the reverse of display order, faster than real-time.

3.88 **field**: For an interlaced video signal, a “field” is the assembly of alternate lines of a frame. Therefore an interlaced frame is composed of two fields, a top field and a bottom field.

3.89 **field-based prediction**: A prediction mode using only one field of the reference frame. The predicted block size is 16x16 luminance samples. Field-based prediction is not used in progressive frames.

3.90 **field period**: The reciprocal of twice the frame rate.

3.91 **field VOP; field structure VOP**: A field structure VOP is a coded VOP with vop_structure is equal to “Top field” or “Bottom field”.

3.99 **frame-based prediction**: A prediction mode using both fields of the reference frame.

3.102 **frame VOP; frame structure VOP**: A frame structure VOP is a coded VOP with vop_structure is equal to “Frame”.

3.103 **future reference frame (field)**: A future reference frame (field) is a reference frame (field) that occurs at a later time than the current VOP in display order.

3.113 **I-field VOP**: A field structure I-VOP.

3.114 **I-frame VOP**: A frame structure I-VOP.

3.147 **RGB component**: A matrix, block or single sample representing one of the three primary colours. The symbols used for the rgb signals are Green, Blue and Red.

3.148 **P-field VOP**: A field structure P-VOP.

3.149 **P-frame VOP**: A frame structure P-VOP.

3.171 **sample aspect ratio**: (abbreviated to **SAR**). This specifies the relative distance between samples. It is defined (for the purposes of this specification) as the vertical displacement of the lines of luminance samples in a frame divided by the horizontal displacement of the luminance samples. Thus its units are (metres per line) ÷ (metres per sample)

3.182 **skipped macroblock**: A macroblock for which no data is encoded.

3.192 **top field**: One of two fields that comprise a frame. Each line of a top field is spatially located immediately above the corresponding line of the bottom field.

"

10) Add the following subclause 5.2.9 after subclause 5.2.8:

"

5.2.9 Definition of next_start_code_studio() function

The next_start_code_studio() function removes any zero bit and zero byte stuffing and locates the next start code.

next_start_code_studio() {	**No. of bits**	**Mnemonic**
while (!bytealigned())		
zero_bit	1	‘0’
while (nextbits() != ‘0000 0000 0000 0000 0000 0001’)		
zero_byte	8	‘0000 0000’
}		

This function checks whether the current position is byte aligned. If it is not, zero stuffing bits are present. After that any number of zero stuffing bytes may be present before the start code. Therefore start codes are always byte aligned and may be preceded by any number of zero stuffing bits.

"

11) Replace subclause 6.1.1 with the following:

"

6.1.1 Visual object sequence

Visual object sequence is the highest syntactic structure of the coded visual bitstream.

A visual object sequence commences with a visual_object_sequence_start_code which is followed by profile_and_level_indication, and one or more visual objects coded concurrently. The visual object sequence is terminated by a visual_object_sequence_end_code.

At various points in the visual object sequence, a repeat visual_object_sequence_start_code can be inserted for coded video data. In that case, the repeat visual_object_sequence_start_code shall follow a particular VOP.

When profile_and_level_indication indicates a Studio Profile, StudioVisualObject() shall follow it.

"

12) Replace subclause 6.1.2 with the following:

"

6.1.2 Visual object

A visual object commences with a visual_object_start_code and a visual object id, which are followed by a video object, a still texture object, a mesh object, or an FBA object.

For Studio Profiles, only video object type is supported.

"

13) Replace subclause 6.1.3 with the following:

"

6.1.3 Video object

A video object commences with a video_object_start_code, and is followed by one or more video object layers.

A video object layer commences with video_object_layer_start_code which may optionally be followed by Group_of_StudioVideoObjectPlane() and then by one or more coded VOPs. The order of the coded frames in the coded bitstream is the order in which the decoder processes them, which is not necessarily the display order.

"

14) Replace subclause 6.1.3.1 with the following:

"

6.1.3.1 Progressive and interlaced sequences

This part of ISO/IEC 14496 deals with coding of both progressive and interlaced sequences.

The sequence, at the output of the decoding process, consists of a series of reconstructed VOPs separated in time and are readied for display via the compositor.

For Studio Profiles paticularly, the output of the decoding process for interlaced sequences consists of a series of reconstructed fields that are separated in time by a field period. The two fields of a frame may be coded separately (field-VOPs). Alternatively the two fields may be coded together as a frame (frame-VOPs). Both frame VOPs and field VOPs may be used in a single video sequence.

In progressive sequences each VOP in the sequence shall be a frame VOP. The sequence, at the output of the decoding process, consists of a series of reconstructed frames that are separated in time by a frame period.

"

15) Replace subclause 6.1.3.2 with the following :

"

6.1.3.2 Frame

A frame consists of three rectangular matrices of integers; a luminance matrix (Y), and two chrominance matrices (Cb and Cr).

The relationship between these Y, Cb and Cr components and the primary (analogue) Red, Green and Blue Signals (E'_R, E'_G and E'_B), the chromaticity of these primaries and the transfer characteristics of the source frame may be specified in the bitstream (or specified by some other means). This information does not affect the decoding process.

For Studio Profiles particularly, the three rectangular matrices can be the primary RGB colour matrices.

"

16) Add the following subclause in subclause 6.1.3 and renumber the subsequent items

"

6.1.3.3 Field

A field consists of every other line of samples in the three rectangular matrices of integers representing a frame.

A frame is the union of a top field and a bottom field. The top field is the field that contains the top-most line of each of the three matrices. The bottom field is the other one.

"

17) Replace subclause 6.1.3.3 with the following:

"

6.1.3.3 VOP

A reconstructed VOP is obtained by decoding a coded VOP. A coded VOP may have been derived from a progressive or interlaced frame or an interlaced field. A reconstructed VOP is either a reconstructed frame (when decoding a frame VOP), or one field of a reconstructed frame (when decoding a field VOP).

An I-frame VOP or a pair of field VOPs, where the first field VOP is an I-picture and the second field VOP is an I-VOP or a P-VOP, is called a coded I-frame.

A P-frame VOP or a pair of P-field VOPs is called a coded P-frame.

A B-frame VOP or a pair of B-field VOPs is called a coded B-frame.

A coded I-frame, a coded P-frame or a coded B-frame is called a coded frame.

6.1.1.4.1 Field VOPs

If field VOPs are used, then they shall occur in pairs (one top field followed by one bottom field, or one bottom field followed by one top field) and together constitute a coded frame. The two field VOPs that comprise a coded frame shall be encoded in the bitstream in the order in which they shall occur at the output of the decoding process.

When the first VOP of the coded frame is a P-field VOP, then the second VOP of the coded frame shall also be a P-field VOP. Similarly when the first VOP of the coded frame is a B-field VOP the second VOP of the coded frame shall also be a B-field VOP.

When the first VOP of the coded frame is a I-field VOP, then the second VOP of the frame shall be either an I-field VOP or a P-field VOP. If the second VOP is a P-field VOP, then certain restrictions apply (see 7.16.7.4.5).

6.1.1.4.2 Frame VOPs

When coding interlaced sequences using frame VOPs, the two fields of the frame shall be interleaved with one another and then the entire frame is coded as a single frame-VOP.

"

18) Replace the following text in subclause 6.1.3.5,

"

1) the modulo part (i.e. the full second units) of the time base for the next VOP after the GOV header in display order

"

with

"

1) the modulo part (i.e. the full second units) of the time base for the next VOP after the GOV header in display order. For Studio Profiles particularly, SMPTE 12M time code information that is not used by the decoding process.

"

19) Replace the following text in subclause 6.1.3.6,

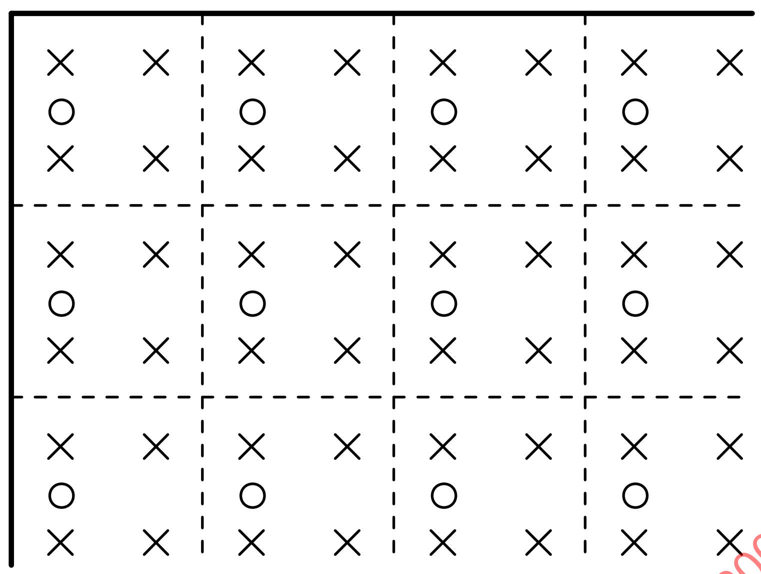
"

6.1.3.6 Format

In this format the Cb and Cr matrices shall be one half the size of the Y-matrix in both horizontal and vertical dimensions. The Y-matrix shall have an even number of lines and samples.

The luminance and chrominance samples are positioned as shown in Figure 6-1.The two variations in the vertical and temporal positioning of the samples for interlaced VOPs are shown in Figure 6-2 and Figure 6-3.

Figure 6-4 shows the vertical and temporal positioning of the samples in a progressive frame.



× Represent luminance samples

○ Represent chrominance samples

Figure 6-1 — The position of luminance and chrominance samples in 4:2:0 data

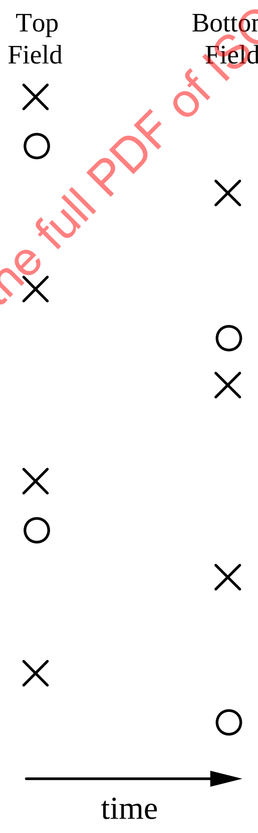


Figure 6-2 — Vertical and temporal positions of samples in an interlaced frame with top_field_first=1

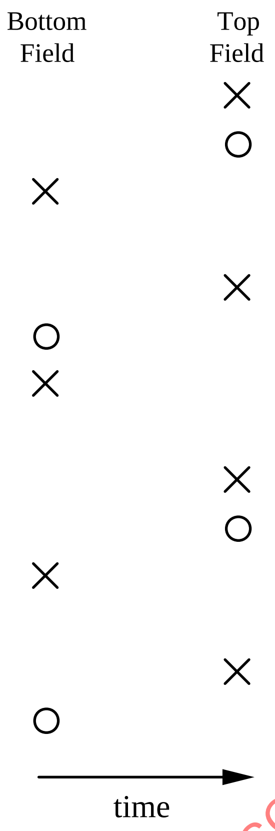


Figure 6-3 — Vertical and temporal position of samples in an interlaced frame with top_field_first=0

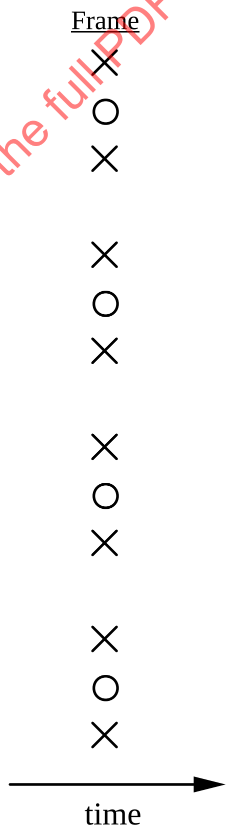


Figure 6-4 — Vertical and temporal positions of samples in a progressive frame

The binary alpha plane for each VOP is represented by means of a bounding rectangle as described in clause F.2, and it has always the same number of lines and pixels per line as the luminance plane of the VOP bounding rectangle. The positions between the luminance and chrominance pixels of the bounding rectangle are defined in this clause according to the 4:2:0 format. For the progressive case, each 2x2 block of luminance pixels in the bounding rectangle associates to one chrominance pixel. For the interlaced case, each 2x2 block of luminance pixels of the same field in the bounding rectangle associates to one chrominance pixel of that field.

In order to perform the padding process on the two chrominance planes, it is necessary to generate a binary alpha plane which has the same number of lines and pixels per line as the chrominance planes. Therefore, when non-scalable shape coding is used, this binary alpha plane associated with the chrominance planes is created from the binary alpha plane associated with the luminance plane by the subsampling process defined below:

For each 2x2 block of the binary alpha plane associated with the luminance plane of the bounding rectangle (of the same frame for the progressive and of the same field for the interlaced case), the associated pixel value of the binary alpha plane associated with the chrominance planes is set to 255 if any pixel of said 2x2 block of the binary alpha plane associated with the luminance plane equals 255.

"

with

"

6.1.3.6 Format

6.1.3.6.1 4:2:0 Format

In this format the Cb and Cr matrices shall be one half the size of the Y-matrix in both horizontal and vertical dimensions. The Y-matrix shall have an even number of lines and samples.

If the matrices represent RGB colour primary matrices, this 4:2:0 format shall not be applied.

NOTE — When interlaced frames are coded as rectangular field VOPs , the VOP reconstructed from each of these field VOPs shall have a Y-matrix with half the number of lines of the corresponding frame. Thus the total number of lines in the Y-matrix of an entire frame shall be divisible by four.

The luminance and chrominance samples are positioned as shown in Figure 6-1.The two variations in the vertical and temporal positioning of the samples for interlaced VOPs are shown in Figure 6-2 and Figure 6-3.

Figure 6-4 shows the vertical and temporal positioning of the samples in a progressive frame.

In each field of an interlaced frame, the chrominance samples do not lie (vertically) mid way between the luminance samples of the field. This is so that the spatial location of the chrominance samples in the frame is the same whether the frame is represented as a single frame-VOP or two field-VOPs.

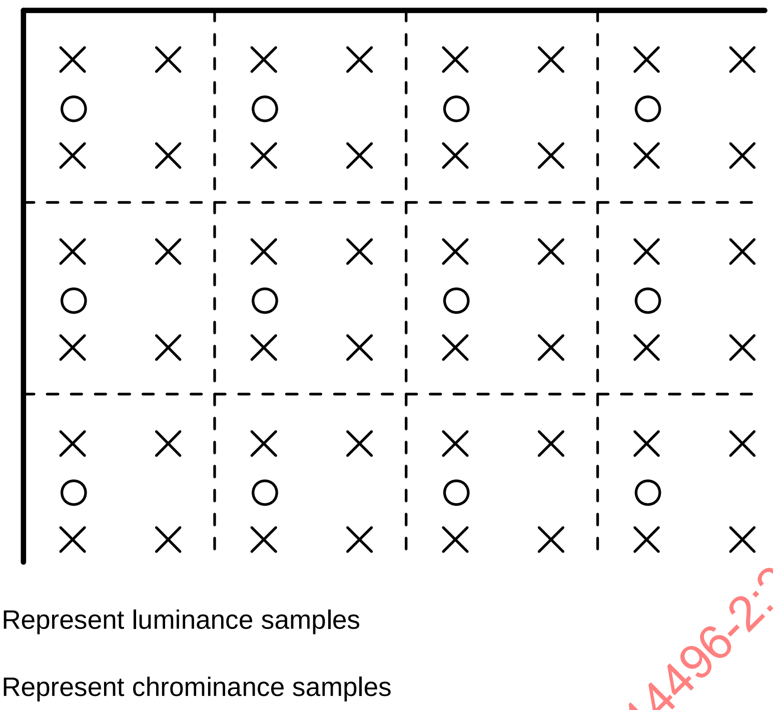


Figure 6-1 — The position of luminance and chrominance samples in 4:2:0 data

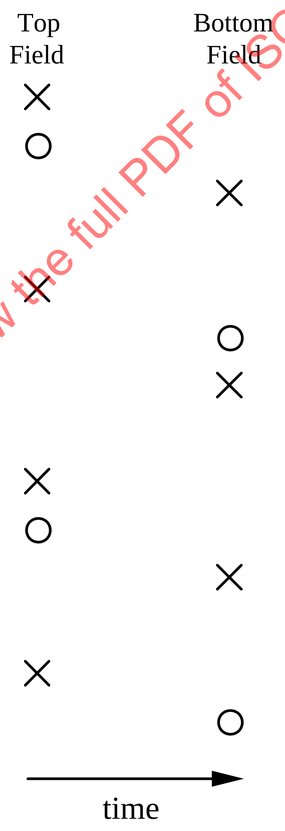


Figure 6-2 — Vertical and temporal positions of samples in an interlaced frame with top_field_first=1

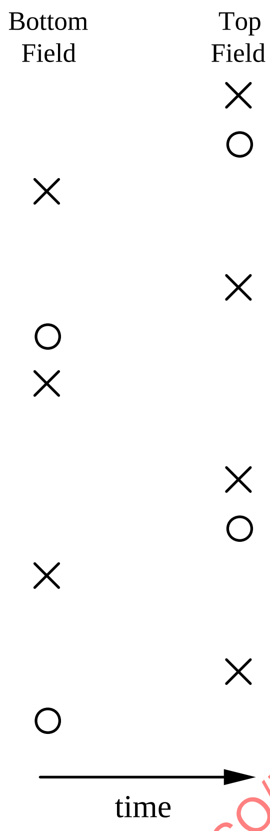


Figure 6-3 — Vertical and temporal position of samples in an interlaced frame with top_field_first=0

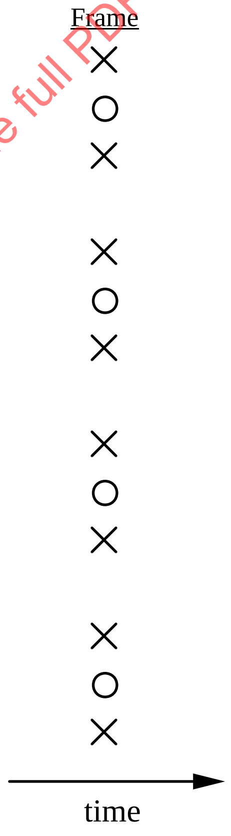


Figure 6-4 — Vertical and temporal positions of samples in a progressive frame

The binary alpha plane for each VOP is represented by means of a bounding rectangle as described in clause F.2, and it always has the same number of lines and pixels per line as the luminance plane of the VOP bounding rectangle. The positions between the luminance and chrominance pixels of the bounding rectangle are defined in this clause according to the 4:2:0 format. For the progressive case, each 2x2 block of luminance pixels in the bounding rectangle associates to one chrominance pixel. For the interlaced case, each 2x2 block of luminance pixels of the same field in the bounding rectangle associates to one chrominance pixel of that field.

In order to perform the padding process on the two chrominance planes, it is necessary to generate a binary alpha plane which has the same number of lines and pixels per line as the chrominance planes. Therefore, when non-scalable shape coding is used, this binary alpha plane associated with the chrominance planes is created from the binary alpha plane associated with the luminance plane by the subsampling process defined below:

For each 2x2 block of the binary alpha plane associated with the luminance plane of the bounding rectangle (of the same frame for the progressive and of the same field for the interlaced case), the associated pixel value of the binary alpha plane associated with the chrominance planes is set to 255 if any pixel of said 2x2 block of the binary alpha plane associated with the luminance plane equals 255.

6.1.3.6.2 4:2:2 Format

In this format the Cb and Cr matrices shall be one half the size of the Y-matrix in the horizontal dimension and the same size as the Y-matrix in the vertical dimension. The Y-matrix shall have an even number of samples.

If the matrices represent RGB colour primar matrices, this 4:2:2 format shall not be applied.

NOTE — When interlaced frames are coded as rectangular field VOPs, the VOP reconstructed from each of these field VOPs shall have a Y-matrix with half the number of lines of the corresponding frame. Thus the total number of lines in the Y-matrix of an entire frame shall be divisible by two.

The luminance and chrominance samples are positioned as shown in Figure AMD1-1.

In order to clarify the organisation, Figure AMD1-2 shows the (vertical) positioning of the samples when the frame is separated into two fields.

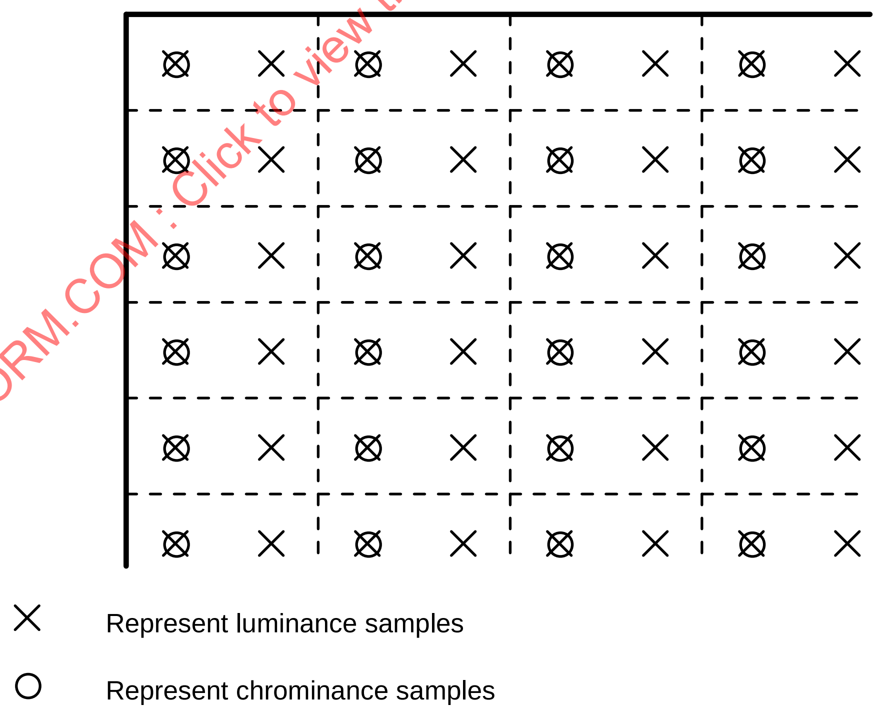


Figure AMD1-1 — The position of luminance and chrominance samples. 4:2:2 data.

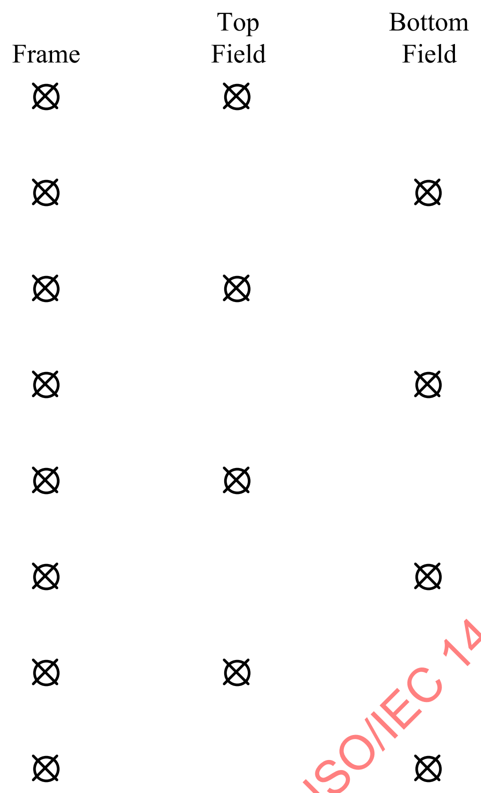


Figure AMD1-2 — Vertical positions of samples with 4:2:2 and 4:4:4 data

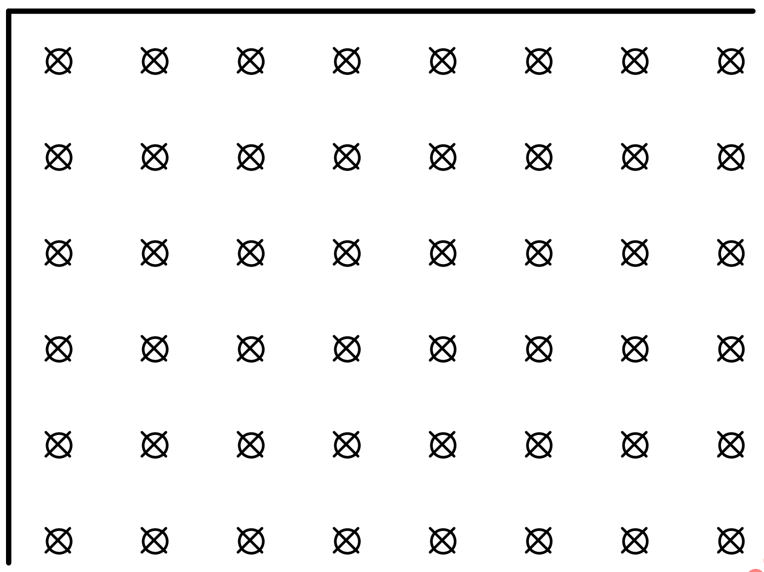
6.1.3.6.3 4:4:4 Format

In this format the Cb and Cr matrices shall be the same size as the Y-matrix in the horizontal and the vertical dimensions.

If the matrices are treated as RGB colour primary matrices, the matrices shall follow this format.

NOTE — When interlaced frames are coded as field rectangular VOPs, the VOP reconstructed from each of these field VOPs shall have a Y-matrix with half the number of lines of the corresponding frame. Thus the total number of lines in the Y-matrix of an entire frame shall be divisible by two.

The luminance and chrominance samples are positioned as shown in Figures AMD1-2 and AMD1-3.



× Represent luminance samples

○ Represent chrominance samples

Figure AMD1-3 — The position of luminance and chrominance samples. 4:4:4 data.

"

20) Replace the following text in subclause 6.1.3.8,

"

A macroblock contains a section of the luminance component and the spatially corresponding chrominance components. The term macroblock can either refer to source and decoded data or to the corresponding coded data elements. A skipped macroblock is one for which no information is transmitted. Presently there is only one chrominance format for a macroblock, namely, 4:2:0 format. The orders of blocks in a macroblock is illustrated below:

A 4:2:0 Macroblock consists of 6 blocks. This structure holds 4 Y, 1 Cb and 1 Cr Blocks and the block order is depicted in Figure 6-5.

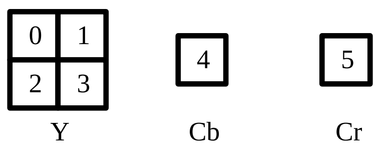


Figure 6-5 — 4:2:0 Macroblock structure

The organisation of VOPs into macroblocks is as follows.

For the case of a progressive VOP, the interlaced flag (in the VOP header) is set to “0” and the organisation of lines of luminance VOP into macroblocks is called frame organization and is illustrated in Figure 6-6. In this case, frame DCT coding is employed.

For the case of interlaced VOP, the interlaced flag is set to “1” and the organisation of lines of luminance VOP into macroblocks can be either frame organization or field organization and thus both frame and field DCT coding may be used in the VOP.

- In the case of frame DCT coding, each luminance block shall be composed of lines from two fields alternately. This is illustrated in Figure 6-6.

- In the case of field DCT coding, each luminance block shall be composed of lines from only one of the two fields. This is illustrated in Figure 6-7.

Only frame DCT coding is applied to the chrominance blocks. It should be noted that field based predictions may be applied for these chrominance blocks which will require predictions of 8x4 regions (after half-sample filtering).

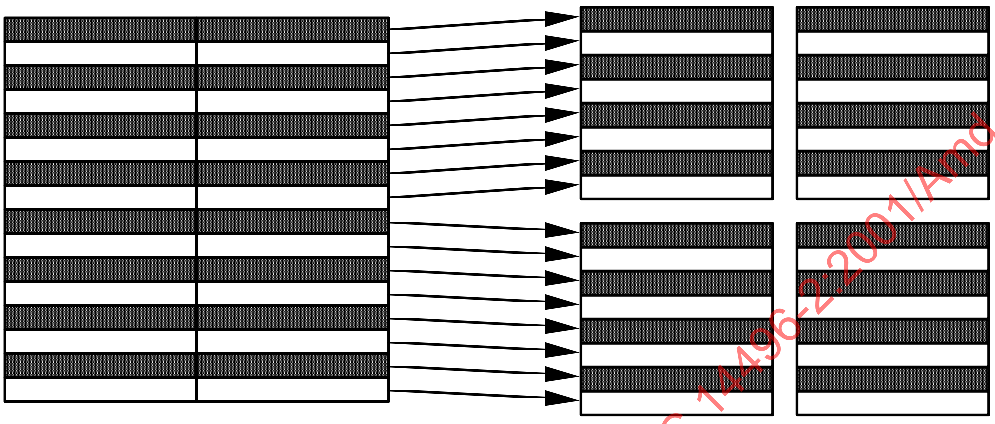


Figure 6-6 — Luminance macroblock structure in frame DCT coding

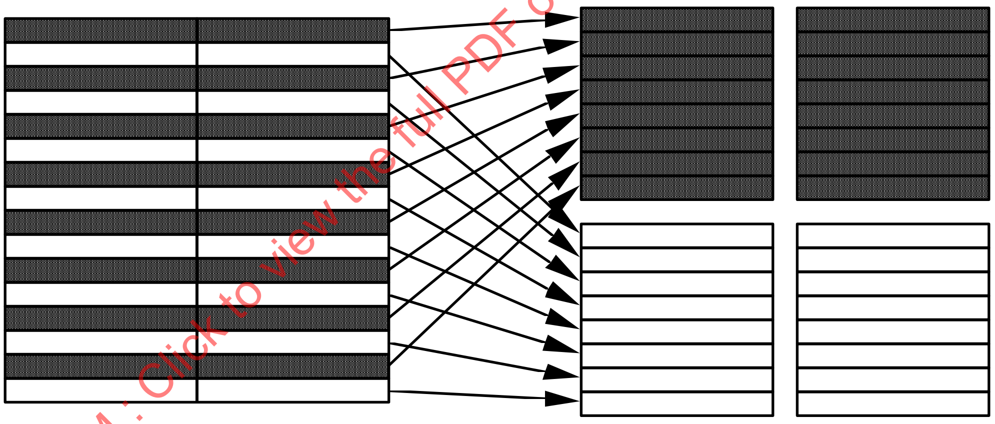


Figure 6-7 — Luminance macroblock structure in field DCT coding

"

with
"

A macroblock contains a section of the luminance component and the spatially corresponding chrominance components. The term macroblock can either refer to source and decoded data or to the corresponding coded data elements. A skipped macroblock is one for which no information is transmitted. There are three chrominance formats for a macroblock, namely, 4:2:0, 4:2:2 and 4:4:4 formats. The order of blocks in a macroblock shall be different for each different chrominance format and are illustrated below:

A 4:2:0 Macroblock consists of 6 blocks. This structure holds 4 Y, 1 Cb and 1 Cr Blocks and the block order is depicted in Figure 6-5.

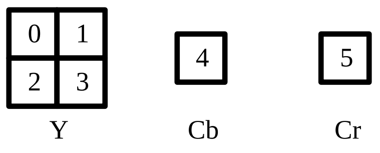


Figure 6-5 — 4:2:0 Macroblock structure

A 4:2:2 Macroblock consists of 8 blocks. This structure holds 4 Y, 2 Cb and 2 Cr Blocks and the block order is depicted in Figure AMD1-4.

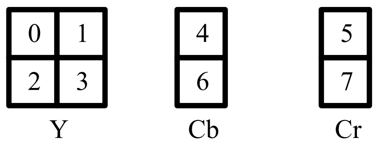


Figure AMD1-4 — 4:2:2 Macroblock structure

A 4:4:4 Macroblock consists of 12 blocks. This structure holds 4 Y, 4 Cb and 4 Cr (or 4 G, 4 B and 4 R) Blocks and the block order is depicted in Figure AMD1-5.

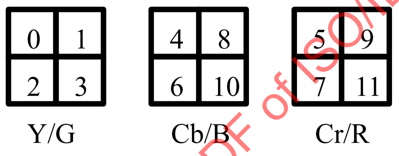


Figure AMD1-5 — 4:4:4 Macroblock structure

In frame VOPs, where both frame and field DCT coding may be used, the internal organisation within the macroblock is different in each case.

- In the case of frame DCT coding, each block shall be composed of lines from two fields alternately. This is illustrated in Figure 6-6.
- In the case of field DCT coding, each block shall be composed of lines from only one of the two fields. This is illustrated in Figure 6-7.

In the case of chrominance blocks the structure depends upon the chrominance format that is being used. In the case of 4:2:2 and 4:4:4 formats (where there are two blocks in the vertical dimension of the macroblock) the chrominance blocks are treated in exactly the same manner as the luminance blocks. However, in the 4:2:0 format the chrominance blocks shall always be organised in frame structure for the purposes of DCT coding. It should however be noted that field based predictions may be made for these blocks which will, in the general case, require that predictions for 8x4 regions (after half-sample filtering) must be made.

In field pictures, each picture only contains lines from one of the fields. In this case each block consists of lines taken from successive lines in the picture as illustrated by Figure 6-6.

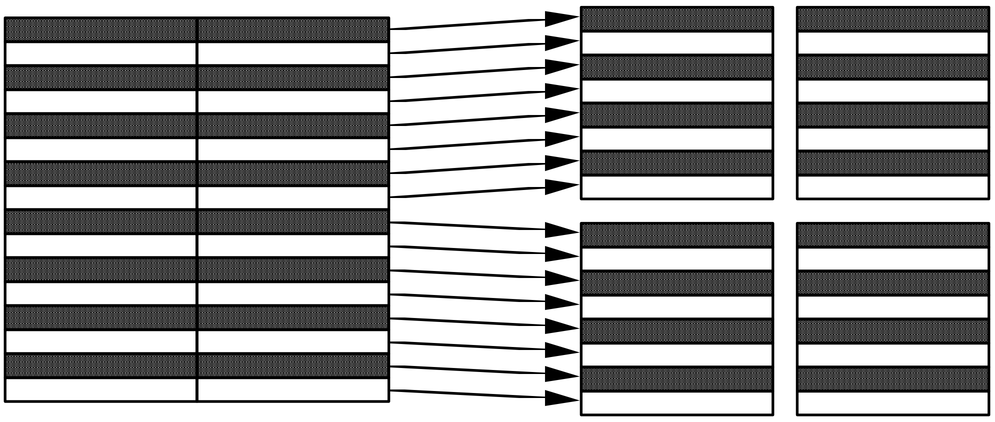


Figure 6-6 — Luminance macroblock structure in frame DCT coding

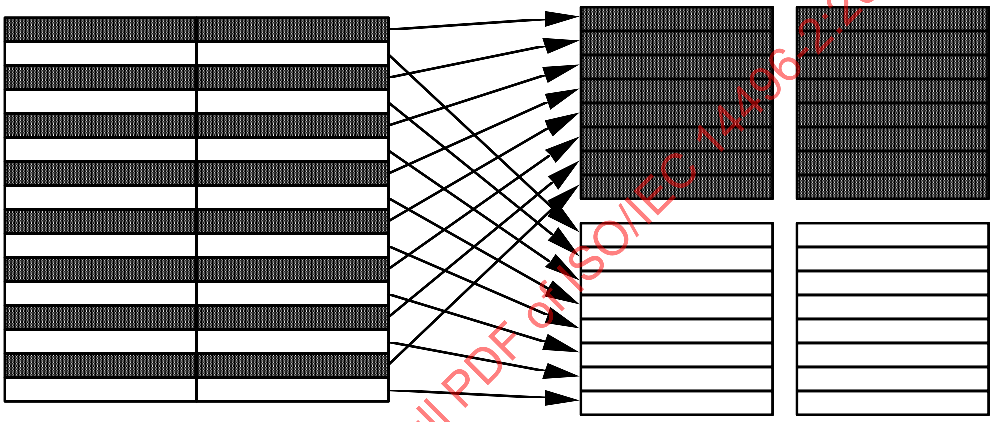


Figure 6-7 — Luminance macroblock structure in field DCT coding

"

21) Add the following subclause 6.1.3.10 after subclause 6.1.3.9:

"

6.1.3.10 Field

A field consists of every other line of samples in the three rectangular matrices of integers representing a frame.

A frame is the union of a top field and a bottom field. The top field is the field that contains the top-most line of each of the three matrices. The bottom field is the other one.

Only when profile_and_level_indication indicates the studio profile, a coded VOP may be a frame VOP or a field VOP. A reconstructed VOP is either a reconstructed frame (when decoding a frame VOP), or one field of a reconstructed frame (when decoding a field VOP).

6.1.3.10.1 Field VOPs

If field VOPs are used then they shall occur in pairs (one top field followed by one bottom field, or one bottom field followed by one top field) and together constitute a coded frame. The two field VOPs that comprise a coded frame shall be encoded in the bitstream in the order in which they shall occur at the output of the decoding process.

When the first VOP of the coded frame is a P-field VOP, then the second VOP of the coded frame shall also be a P- field VOP. Similarly when the first VOP of the coded frame is a B-field VOP the second VOP of the coded frame shall also be a B-field VOP.

When the first VOP of the coded frame is a I-field VOP, then the second VOP of the frame shall be either an I-field VOP or a P-field VOP. If the second VOP is a P-field VOP then certain restrictions apply,.

6.1.3.10.2 Frame VOPs

When coding interlaced sequences using frame VOPs, the two fields of the frame shall be interleaved with one another and then the entire frame is coded as a single frame-VOP.

"

22) Add the following subclauses 6.1.3.11 after subclause 6.1.3.10:

"

6.1.3.11 Slice

A slice is a series of an arbitrary number of consecutive macroblocks. The first and last macroblocks of a slice shall not be skipped macroblocks. Every slice shall contain at least one macroblock. Slices shall not overlap. The position of slices may change from picture to picture.

The first and last macroblock of a slice shall be in the same horizontal row of macroblocks.

Slices shall occur in the bitstream in the order in which they are encountered, starting at the upper-left of the picture and proceeding by raster-scan order from left to right and top to bottom (illustrated in the Figures of this clause as alphabetical order).

6.1.3.11.1 The general slice structure

In the most general case it is not necessary for the slices to cover the entire picture. Figure AMD1-6 shows this case. Those areas that are not enclosed in a slice are not encoded and no information is encoded for such areas (in the specific picture).

If the slices do not cover the entire picture then it is a requirement that if the picture is subsequently used to form predictions then predictions shall only be made from those regions of the picture that were enclosed in slices. It is the responsibility of the encoder to ensure this.

This specification does not define what action a decoder shall take in the regions between the slices.

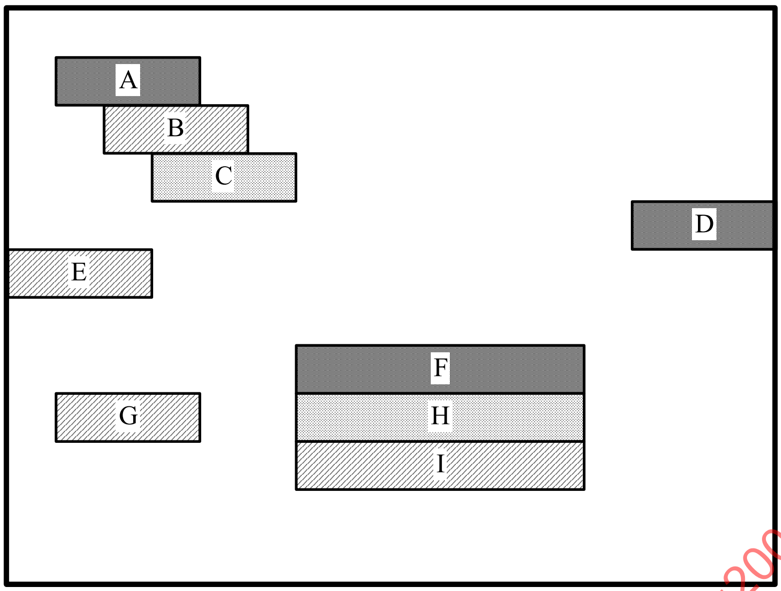


Figure AMD1-6 — The most general slice structure.

6.1.3.11.2 Restricted slice structure

In certain defined levels of defined profiles a restricted slice structure illustrated in Figure AMD1-7 shall be used. In this case every macroblock in the picture shall be enclosed in a slice.

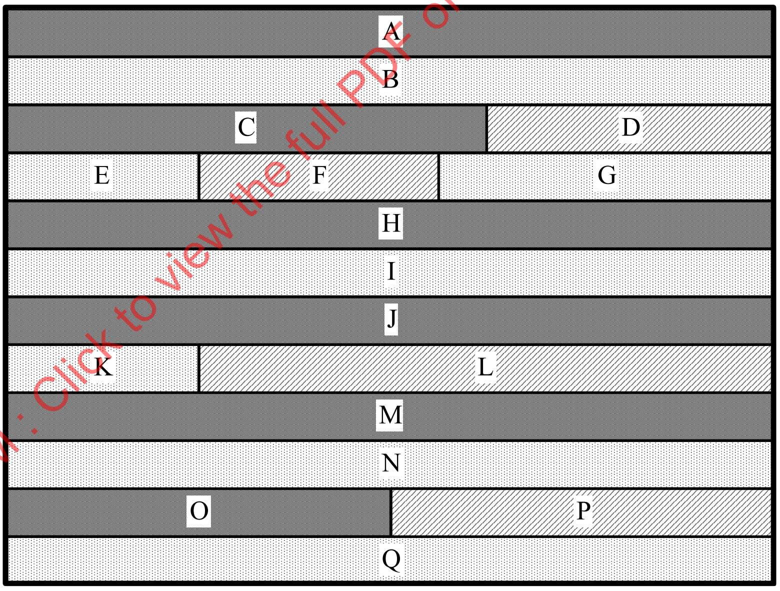


Figure AMD1-7 — Restricted slice structure.

Where a defined level of a defined profile requires that the slice structure obeys the restrictions detailed in this clause, the term “restricted slice structure” may be used.

"

23) Add the following text in subclause 6.2.1 before paragraph 5 (after Table 6-2):

"

Only when profile_and_level_indication indicates a studio profile, byte alignment shall be achieved by inserting bits with the value zero before the start code prefix such that the first bit of the start code prefix is the first (most significant) bit of a byte.

"

24) Replace Table 6-3 in subclause 6.2.1 with the following:

"

Table 6-3 — Start code values

name	**start code value (hexadecimal)**
video_object_start_code	00 through 1F
video_object_layer_start_code	20 through 2F
reserved	30 through AF
visual_object_sequence__start_code	B0
visual_object_sequence_end_code	B1
user_data_start_code	B2
group_of_vop_start_code	B3
video_session_error_code	B4
visual_object_start_code	B5
vop_start_code	B6
slice_start_code	B7
extension_start_code	B8
reserved	B9
fba_object_start_code	BA
fba_object_plane_start_code	BB
mesh_object_start_code	BC
mesh_object_plane_start_code	BD
still_texture_object_start_code	BE
texture_spatial_layer_start_code	BF
texture_snr_layer_start_code	C0
texture_tile_start_code	C1
texture_shape_layer_start_code	C2
reserved	C3-C5
System start codes (see Note)	C6 through FF
NOTE — System start codes are defined in ISO/IEC 14496-1:1999.	

"

25) Replace VisualObjectSequence() in subclause 6.2.2 with the following:

"

VisualObjectSequence() {	**No. of bits**	**Mnemonic**
do {		
visual_object_sequence_start_code	32	bslbf
profile_and_level_indication	8	uimsbf
if (profile_and_level_indication == 11100001-11101000) {		
next_start_code_studio()		
extension_and_user_data(0)		
StudioVisualObject()		
} else {		
while (next_bits() == user_data_start_code) {		
user_data()		
}		
VisualObject()		
}		
} while (nextbits() != visual_object_sequence_end_code)		
visual_object_sequence_end_code	32	bslbf
}		

"

26) Add the following subclause 6.2.13 after subclause 6.2.12:

"

6.2.13 Studio Video Object

6.2.13.1 Studio Visual Object

StudioVisualObject() {	**No. of bits**	**Mnemonic**
visual_object_start_code	32	bslbf
visual_object_verid	4	uimsbf
visual_object_type	4	uimsbf
next_start_code_studio()		
extension_and_user_data(1)		
if (visual_object_type == “video ID“) {		
video_object_start_code	32	bslbf
StudioVideoObjectLayer()		
} else {		
/* Other visual object types are not supported in StudioVisualObject() */		
}		
}		

6.2.13.2 Extension and user data

extension_and_user_data(i) {	**No. of bits**	**Mnemonic**
while ((next_bits() == extension_start_code) ||		
(next_bits() == user_data_start_code)) {		
if ((i==2 || i==4) && (next_bits() == extension_start_code))		
extension_data(i)		
if (next_bits() == user_data_start_code)		
user_data_studio()		
}		
}		

6.2.13.2.1 Extension data

extension_data(i) {	**No. of bits**	**Mnemonic**
while (next_bits()== extension_start_code) {		
extension_start_code	32	bslbf
/* NOTE - i never takes the value 0 because extension_data() is never called in a VisualObjectSequence() */		
/* NOTE - i never takes the value 1 because extension_data() is never called in a StudioVisualObject() */		
if (i == 2) { /* Called in StudioVideoObjectLayer() */		
if (next_bits()== “Sequence Display Extension ID”)		
sequence_display_extension()		
else if (next_bits() == “Quant Matrix Extension ID”)		
quant_matrix_extension()		
else if (nextbits() == “VLC Code Extension ID”)		
vlc_code_extension()		
}		
/* NOTE - i never takes the value 3 because extension_data() is never called in a Group_of_StudioVideoObjectPlane() */		
if (i == 4) { /* Called in VideoObjectPlane() */		
if (nextbits() == “Quant Matrix Extension ID”)		
quant_matrix_extension()		
else if (nextbits() == “Copyright Extension ID”)		
copyright_extension()		
else if (nextbits() == “Picture Display Extension ID”)		
picture_display_extension()		
else if(nextbits() ==“Camera Prameters Extension ID”)		
camera_parameters_extension()		
else if (nextbits() == “ITU-T Extension ID”)		
ITU-T_extension()		
else if (nextbits() == “VLC Code Extension ID”)		
vlc_code_extension()		
}		
}		

6.2.13.2.2 User data Studio

user_data_studio() {	**No. of bits**	**Mnemonic**
user_data_start_code	32	bslbf
while(next_bits() != ‘0000 0000 0000 0000 0000 0001’) {		
user_data	8	uimsbf
}		
next_start_code_studio()		
}		

6.2.13.2.3 Sequence display extension

sequence_display_extension() {	**No. of bits**	**Mnemonic**
extension_start_code_identifier	4	uimsbf
video_format	3	uimsbf
video_range	1	bslbf
colour_description	1	uimsbf
if (colour_description) {		
colour_primaries	8	uimsbf
transfer_characteristics	8	uimsbf
matrix_coefficients	8	uimsbf
}		
display_horizontal_size	14	uimsbf
marker_bit	1	bslbf
display_vertical_size	14	uimsbf
next_start_code_studio()		
}		

6.2.13.2.4 Quant matrix extension

quant_matrix_extension() {	**No. of bits**	**Mnemonic**
extension_start_code_identifier	4	uimsbf
load_intra_quantiser_matrix	1	uimsbf
if (load_intra_quantiser_matrix)		
intra_quantiser_matrix[64]	8 * 64	uimsbf
load_non_intra_quantiser_matrix	1	uimsbf
if (load_non_intra_quantiser_matrix)		
non_intra_quantiser_matrix[64]	8 * 64	uimsbf
load_chroma_intra_quantiser_matrix	1	uimsbf
if (load_chroma_intra_quantiser_matrix)		
chroma_intra_quantiser_matrix[64]	8 * 64	uimsbf
load_chroma_non_intra_quantiser_matrix	1	uimsbf
if (load_chroma_non_intra_quantiser_matrix)		
chroma_non_intra_quantiser_matrix[64]	8 * 64	uimsbf
if (video_object_layer_shape == 'grayscale') {		
for(i=0; i<aux_comp_count; i++) {		
load_intra_quantiser_matrix_grayscale[i]	1	uimsbf
if (load_intra_quantiser_matrix_grayscale[i])		
intra_quantiser_matrix_grayscale [i][64]	8 * 64	uimsbf
load_non_intra_quantiser_matrix_grayscale[i]	1	uimsbf
if (load_non_intra_quantiser_matrix_grayscale[i])		
non_intra_quantiser_matrix_grayscale [i][64]	8 * 64	uimsbf
}		
}		
next_start_code_studio()		
}		

6.2.13.2.5 Picture display extension

picture_display_extension() {	**No. of bits**	**Mnemonic**
extension_start_code_identifier	4	uimsbf
for (i=0; i<number_of_frame_centre_offsets; i++) {		
frame_centre_horizontal_offset	16	simsbf
marker_bit	1	bslbf
frame_centre_vertical_offset	16	simsbf
marker_bit	1	bslbf
}		
next_start_code_studio()		
}		

6.2.13.2.6 Copyright extension

copyright_extension() {	**No. of bits**	**Mnemonic**
extension_start_code_identifier	4	uimsbf
copyright_flag	1	uimsbf
copyright_identifier	8	uimsbf
original_or_copy	1	uimsbf
reserved	7	bslbf
marker_bit	1	bslbf
copyright_number_1	20	uimsbf
marker_bit	1	bslbf
copyright_number_2	22	uimsbf
marker_bit	1	bslbf
copyright_number_3	22	uimsbf
next_start_code_studio()		
}		

6.2.13.2.7 Camera Parameters extension

camera_parameters_extension() {	**No. of bits**	**Mnemonic**
extension_start_code_identifier	4	uimsbf
reserved	1	uimsbf
camera_id	7	simsbf
marker_bit	1	bslbf
height_of_image_device	22	uimsbf
marker_bit	1	bslbf
focal_length	22	uimsbf
marker_bit	1	bslbf
f_number	22	uimsbf
marker_bit	1	bslbf
vertical_angle_of_view	22	uimsbf
marker_bit	1	bslbf
camera_position_x_upper	16	simsbf
marker_bit	1	bslbf
camera_position_x_lower	16	
marker_bit	1	bslbf
camera_position_y_upper	16	simsbf
marker_bit	1	bslbf
camera_position_y_lower	16	
marker_bit	1	bslbf
camera_position_z_upper	16	simsbf
marker_bit	1	bslbf
camera_position_z_lower	16	
marker_bit	1	bslbf
camera_direction_x	22	simsbf
marker_bit	1	bslbf
camera_direction_y	22	simsbf
marker_bit	1	bslbf
camera_direction_z	22	simsbf
marker_bit	1	bslbf
image_plane_vertical_x	22	simsbf
marker_bit	1	bslbf
image_plane_vertical_y	22	simsbf
marker_bit	1	bslbf
image_plane_vertical_z	22	simsbf
marker_bit	1	bslbf
reserved	32	bslbf
next_start_code_studio()		
}		

6.2.13.2.8 ITU-T extension

ITU-T extension () {	No. of bits	Mnemonic
extension_start_code_identifier	4	uimsbf
while (nextbits() != '0000 0000 0000 0000 0000 0001') {		
ITU-T_data	1	bslbf
}		
next_start_code_studio()		
}		
NOTE – The construct with the while-statement prevents start code emulation.		

6.2.13.2.9 VLC code extension

vlc_code_extension() {	No. of bits	Mnemonic
extension_start_code_identifier	4	uimsbf
load_vlc_code	4	uimsbf
if (load_intra_vlc_code == 1) {		
for (j=0; j< 12; j++) {		
for (i=0; i< 22; i++) {		
Intra_vlc_length[j][i]	4	uimsbf
Intra_vlc_code[j][i]	16	uimsbf
}		
}		
}		
if (load_inter_vlc_code == 1) {		
for (j=0; j< 12; j++) {		
for (i=0; i< 22; i++) {		
inter_vlc_length[j][i]	4	uimsbf
inter_vlc_code[j][i]	16	uimsbf
}		
}		
}		
next_start_code_studio()		
}		

6.2.13.3 Studio Video Object Layer

StudioVideoObjectLayer() {	No. of bits	Mnemonic
video_object_layer_start_code	32	bslbf
random_accessible_vol	1	bslbf
video_object_type_indication	8	uimsbf
video_object_layer_verid	4	uimsbf
video_object_layer_shape	2	uimsbf
video_object_layer_shape_extension	4	uimsbf
progressive_sequence	1	bslbf
if (video_object_layer_shape != “binary only”) {		
rgb_components	1	uimsbf
chroma_format	2	uimsbf

bits_per_pixel	4	uimsbf
}		
if (video_object_layer_shape == “rectangular”) {		
marker_bit	1	bslbf
video_object_layer_width	14	uimsbf
marker_bit	1	bslbf
video_object_layer_height	14	uimsbf
marker_bit	1	bslbf
}		
aspect_ratio_info	4	uimsbf
if (aspect_ratio_info == “extended_PAR”) {		
par_width	8	uimsbf
par_height	8	uimsbf
}		
frame_rate_code	4	uimsbf
first_half_bit_rate	15	uimsbf
marker_bit	1	bslbf
latter_half_bit_rate	15	uimsbf
marker_bit	1	bslbf
first_half_vbv_buffer_size	15	uimsbf
marker_bit	1	bslbf
latter_half_vbv_buffer_size	3	uimsbf
first_half_vbv_occupancy	11	uimsbf
marker_bit	1	blsbf
latter_half_vbv_occupancy	15	uimsbf
marker_bit	1	blsbf
low_delay	1	uimsbf
mpeg2_stream	1	uimsbf
if (video_object_layer_shape == ”grayscale”) {		
for(i=0; i<aux_comp_count; i++) {		
alpha_bits_per_pixel[i]	4	uimsbf
minimum_alpha_level[i]	12	uimsbf
maximum_alpha_level[i]	12	uimsbf
}		
composition_method	1	bslbf
linear_composition	1	bslbf
}		
if (video_object_layer_shape != “binary only”) {		
sprite_enable	1	bslbf
if (sprite_enable) {		
sprite_width	20	uimsbf
marker_bit	1	bslbf
sprite_height	20	uimsbf
marker_bit	1	bslbf
sprite_left_coordinate	20	uimsbf
marker_bit	1	bslbf
sprite_top_coordinate	20	uimsbf
marker_bit	1	bslbf
no_of_sprite_warping_points	6	uimsbf

sprite_warping_accuracy	2	uimsbf
sprite_brightness_change	1	bslbf
sprite_defocusing	1	bslbf
sprite_lens_distortion	1	bslbf
}		
}		
next_start_code_studio()		
extension_and_user_data(2)		
if (sprite_enable)		
StudioVideoObjectPlane()		
do {		
if (next_bits() == group_of_vop_start_code)		
Group_of_StudioVideoObjectPlane()		
StudioVideoObjectPlane()		
} while ((next_bits()== group_of_vop_start_code) || (next_bits() == vop_start_code))		
}		

6.2.13.4 Group of Studio Video Object Plane

Group_of_StudioVideoObjectPlane() {	**No. of bits**	**Mnemonic**
group_vop_start_codes	32	bslbf
time_code_smpte12m	64	
closed_gov	1	bslbf
broken_link	1	bslbf
next_start_code_studio()		
extension_and_user_data(3)		
}		

6.2.13.5 Studio Video Object Plane

StudioVideoObjectPlane() {	**No. of bits**	**Mnemonic**
vop_start_code	32	bslbf
time_code_smpte12m	64	
temporal_reference	10	bslbf
vop_structure	2	uimsbf
vop_coding_type	2	uimsbf
vop_coded	1	bslbf
if (vop_coded == ’0’) {		
next_start_code_studio()		
extension_and_user_data(4)		
return()		
}		
if (video_object_layer_shape != “rectangular”) {		
if (!(sprite_enable && vop_coding_type == “I“)) {		
vop_width	14	uimsbf
marker_bit	1	bslbf
vop_height	14	uimsbf

marker_bit	1	bslbf
vop_horizontal_mc_spatial_ref	14	simsbf
marker_bit	1	bslbf
vop_vertical_mc_spatial_ref	14	simsbf
}		
}		
top_field_first	1	bslbf
repeat_first_field	1	bslbf
progressive_frame	1	bslbf
if (vop_coding_type == "I")		
intra_predictors_reset	1	bslbf
}		
if (video_object_layer_shape != "binary only") {		
alternate_scan	1	bslbf
frame_pred_frame_dct	1	bslbf
dct_precision	2	uimsbf
intra_dc_precision	2	uimsbf
q_scale_type	1	bslbf
if ((vop_coding_type != "I")		
&& !(sprite_enable && vop_coding_type == "S")) {		
vop_fcode[0][0]	4	uimsbf
vop_fcode[0][1]	4	uimsbf
vop_fcode[1][0]	4	uimsbf
vop_fcode[1][1]	4	uimsbf
dead_zone_disable	1	bslbf
}		
}		
if (video_object_layer_shape == "grayscale") {		
for(i=0; i<aux_comp_count; i++) {		
alpha_dct_precision[i]	2	uimsbf
alpha_intra_dc_precision[i]	2	uimsbf
alpha_q_scale_type[i]	1	bslbf
}		
}		
if (sprite_enable && vop_coding_type == "S") {		
if (no_sprite_points > 0)		
sprite_trajectory()		
if (sprite_brightness_change)		
vop_sprite_brightness_change	1	bslbf
if (vop_sprite_brightness_change)		
brightness_change_factor()		
if (sprite_defocusing)		
defocusing_control()		
if (sprite_lens_distortion)		
lens_distortion_parameter()		
next_start_code_studio()		
return()		
}		
composite_display_flag	1	bslbf

if (composite_display_flag) {		
v_axis	1	bslbf
field_sequence	3	uimsbf
sub_carrier	1	bslbf
burst_amplitude	7	uimsbf
sub_carrier_phase	8	uimsbf
}		
while (nextbits() == ‘1’) {		
extra_bit_picture /* with the value ‘1’ */	1	uimsbf
extra_information_picture	8	uimsbf
}		
extra_bit_picture /* with the value ‘0’ */	1	uimsbf
next_start_code_studio()		
extension_and_user_data(4)		
do {		
StudioSlice()		
} while (nextbits() == slice_start_code)		
}		

6.2.13.6 Studio sprite coding

sprite_trajectory() {	**No. of bits**	**Mnemonic**
for (i=0; i < no_of_sprite_warping_points; i++) {		
warping_mv_code(du[i])		
warping_mv_code(dv[i])		
}		
}		

warping_mv_code(d) {	**No. of bits**	**Mnemonic**
dmv_length	2-19	uimsbf
if (dmv_length != ‘00’)		
dmv_code	1-21	uimsbf
marker_bit	1	bslbf
}		

brightness_change_factor() {	**No. of bits**	**Mnemonic**
brightness_change_factor_size	1-4	uimsbf
brightness_change_factor_code	5-10	uimsbf
}		

defocusing_control() {	**No. of bits**	**Mnemonic**
defocusing_control_parameter	9-12	uimsbf
marker_bit	1	bslbf
}		

lens_distortion_parameter() {	**No. of bits**	**Mnemonic**
lens_distortion_parameter_1	16	bslbf
marker_bit	1	bslbf
lens_distortion_parameter_2	16	bslbf
marker_bit	1	bslbf
lens_center_horizontal	14	bslbf
marker_bit	1	bslbf
lens_center_vertical	14	bslbf
marker_bit	1	bslbf
}		

6.2.13.7 Studio Slice

StudioSlice() {	**No. of bits**	**Mnemonic**
slice_start_code	32	uimsbf
macroblock_number	1-14	vlclbf
if (video_object_layer_shape != “binary only”)		
quantiser_scale_code	5	uimsbf
if(video_object_layer_shape==”grayscale”)		
for(i=0; i<aux_comp_count; i++)		
alpha_quantiser_scale_code[i]	5	uimsbf
if (nextbits() == 1) {		
slice_extension_flag	1	bslbf
intra_slice	1	uimsbf
slice_VOP_id_enable	1	uimsbf
slice_VOP_id	6	uimsbf
while (nextbits() == 1) {		
extra_bit_slice	1	uimsbf
extra_information_slice	8	uimsbf
}		
}		
extra_bit_slice	1	uimsbf
do {		
StudioMacroblock()		
} while (next_bits() != ‘000 0000 0000 0000 0000 0000’)		
next_start_code_studio()		
}		

6.2.13.8 Studio Macroblock

StudioMacroblock() {	**No. of bits**	**Mnemonic**
if (vop_coding_type != “B”) {		
if (video_object_layer_shape != “rectangular”)		
Studio_mb_binary_shape_coding ()		
if (video_object_layer_shape != “binary only”) {		
if (!transparent_mb()) {		
if (vop_coding_type != “I”)		
not_coded	1	bslbf
if (vop_coding_type == “I” || !not_coded) {		
compression_mode	1	bslbf
if (compression_mode == "DCT") { /* DCT */		
StudioMacroblock_modes()		
if (macroblock_quant)		
quantiser_scale_code	5	uimsbf
if (macroblock_motion_forward)		
motion_vectors(0)		
if (macroblock_pattern)		
coded_block_pattern()		
for (i = 0; i < block_count; i++)		
if(!transparent_block(i))		
StudioBlock(i)		
} /* -end- DCT */		
else { /* DPCM */		
dpcm_scan_order	1	bslbf
StudioDPCMBlock() // Y or G		
StudioDPCMBlock() // U or B		
StudioDPCMBlock() // V or R		
} /* -end- DPCM */		
}		
}		
}		
}		
if(video_object_layer_shape==“grayscale” && !transparent_mb()) {		
for (j=0 ; j<aux_comp_count ; j++) {		
if(macroblock_intra || compression_mode == "DPCM") {		
coda_i[j]	1	bslbf
if(coda_i[j]==”coded”) {		
alpha_compression_mode[j]	1	bslbf
if(alpha_compression_mode[j]== "DCT") {		
alpha_macroblock_quant[j]	1	bslbf
if (alpha_macroblock_quant[j])		
alpha_quantiser_scale_code[j]	5	uimsbf
for(i=0;i<alpha_block_count;i++)		
if(!transparent_block(i))		
StudioAlphaBlock(j,i)		

} /* -end- DCT */		
else { /* DPCM */		
alpha_dpcm_scan_order[j]	1	bslbf
StudioDPCMBlock() // alpha block		
} /* -end- DPCM */		
}		
} else { /* inter macroblock */		
coda_pb[j]	1-2	vlclbf
if(coda_pb[j]==”coded”){		
alpha_compression_mode[j]	1	bslbf
if(alpha_compression_mode[j]== "DCT") {		
cbpa[j]	1-6	vlclbf
if (cbpa[j] != ‘no DCT coeff’) {		
alpha_macroblock_quant[j]	1	bslbf
if (alpha_macroblock_quant[j])		
alpha_quantiser_scale_code[j]	5	uimsbf
}		
for(i=0 ;i<alpha_block_count ;i++)		
if(!transparent_block(i))		
StudioAlphaBlock(j,i)		
} else {		
alpha_dpcm_scan_order[j]	1	bslbf
StudioDPCMBlock(j) // Alpha		
}		
}		
} /* -end- inter */		
} /* -roop- aux_comp_count */		
} /* -end- grayscale */		
}		
NOTE The value of block_count is 6 in the 4:2:0 format. The value of alpha_block_count is 4.		

6.2.13.8.1 Studio MB Binary Shape Coding

Studio_mb_binary_shape_coding () {	**No. of bits**	**Mnemonic**
bab_type	1-6	vlclbf
if (vop_coding_type == 'P'){		
if (bab_type == 1){		
mvs_x	1-18	vlclbf
mvs_y	1-18	vlclbf
}		
}		
if (bab_type == 4) {		
inferior_symbol_macroblock	1	bslbf
cbbp	3-7	vlclbf
for (i=0: i<4; i++){		
if (coded_binary_block_pattern[i]==’1’){		

inferior_symbol_block	1	bslbf
scan_direction	1	bslbf
backward_load_flag	1	bslbf
clp	3-24	vlclbf
for (j=0; j<8; j++){		
if (coded_line_pattern[j]=='1')		
lbp	1-23	vlclbf
}		
}		
}		
}		
}		

6.2.13.8.2 Studio Macroblock modes

StudioMacroblock_modes() {	**No. of bits**	**Mnemonic**
macroblock_type	1-9	vlclbf
if (macroblock_motion_forward) {		
if (vop_structure == 'frame') {		
if (frame_pred_frame_dct == 0)		
frame_motion_type	2	uimsbf
} else {		
field_motion_type	2	uimsbf
}		
}		
if ((vop_structure == "Frame picture") &&		
(frame_pred_frame_dct == 0) &&		
(macroblock_intra || macoblock_pattern)){		
dct_type	1	uimsbf
}		
}		

6.2.13.8.3 Motion vectors

motion_vectors (s) {	**No. of bits**	**Mnemonic**
if (motion_vector_count == 1) {		
if (mv_format == field)		
motion_vertical_field_select[0][s]	1	uimsbf
motion_vector(0, s)		
} else {		
motion_vertical_field_select[0][s]	1	uimsbf
motion_vector(0, s)		
motion_vertical_field_select[1][s]	1	uimsbf
motion_vector(1, s)		
}		
}		

6.2.13.8.4 Motion vector

motion_vector (r, s) {	**No. of bits**	**Mnemonic**
motion_code[r][s][0]	1-11	vlclbf
if ((vop_fcode[s][0] != 1) && (motion_code[r][s][0] != 0))		
motion_residual[r][s][0]	1-8	uimsbf
motion_code[r][s][1]	1-11	vlclbf
if ((vop_fcode[s][1] != 1) && (motion_code[r][s][1] != 0))		
motion_residual[r][s][1]	1-8	uimsbf
}		

6.2.13.8.5 Coded block pattern

coded_block_pattern () {	**No. of bits**	**Mnemonic**
coded_block_pattern_420	3-9	vlclbf
if (chroma_format == 4:2:2)		
coded_block_pattern_1	2	uimsbf
if (chroma_format == 4:4:4)		
coded_block_pattern_2	6	uimsbf
}		

6.2.13.9 Studio Block

The detailed syntax for the term "DCT coefficient" is fully described in clause 7.

StudioBlock(i) {	**No. of bits**	**Mnemonic**
if (pattern_code[i]) {		
if (macroblock_intra) {		
if (i<4) {		
dct_dc_size_luminance	2-14	vlclbf
if(dct_dc_size_luminance != 0)		
dct_dc_differential	1-15	vlclbf
if (dct_dc_size_luminance > 8)		
marker_bit	1	bslbf
} else {		
dct_dc_size_chrominance	2-15	vlclbf
if(dct_dc_size_chrominance !=0)		
dct_dc_differential	1-15	vlclbf
if (dct_dc_size_chrominance > 8)		
marker_bit	1	bslbf
}		
} else {		
First DCT coefficient	2-24	vlcfbf
}		
while (next_bits() != End of block)		
Subsequent DCT coefficients	3-24	vlclbf
End of block	3-15	vlcfbf
}		
}		

6.2.13.9.1 Studio Alpha Block

The syntax for DCT coefficient decoding is the same as for block(i) in 6.2.13.9.

StudioAlphaBlock(j, i) {	**No. of bits**	**Mnemonic**
last = 0		
if (alpha_pattern_code[j][i]) {		
if (macroblock_intra || compression_mode == "DPCM") {		
dct_dc_size_alpha	2-14	vlclbf
if(dct_dc_size_alpha != 0)		
dct_dc_differential	1-15	vlclbf
if (dct_dc_size_alpha > 8)		
marker_bit	1	bslbf
} else {		
First DCT coefficient	2-24	vlcfbf
}		
while (next_bits() != End of block)		
Subsequent DCT coefficients	3-24	vlclbf
End of block	3-15	vlcfbf
}		
}		

6.2.13.10 Studio DPCM Block

StudioDPCMBlock() {	**No. of bits**	**Mnemonic**
block_mean	8-12	uimsbf
rice_parameter	4	uimsbf
for(i=0 ;i<8 or 16 ; i++) {		
for(j=0 ;j<8 or 16;j++) {		
rice_prefix_code	1-12	vlclbf
if (**rice_prefix_code** == ‘0000 0000 0001’)		
dpcm_residual	4-12	uimsbf
else		
rice_suffix_code	0-12	uimsbf
}		
}		
}		

"

27) Add the following text at the end of subclause 6.3.1:

"

Only when profile_and_level_indication indicates a studio profile, extension layers can be inserted in the bitstream. At each point where extensions are allowed in the bitstream any number of the extensions from the defined allowable set may be included. However each type of extension shall not occur more than once.

In the case that a decoder encounters an extension with an extension identification that is described as “reserved” in this specification the decoder shall discard all subsequent data until the next start code. This requirement allows future definition of compatible extensions to this specification.

Table AMD1-1 — extension_start_code_identifier codes.

extension_start_code_identifier	Name
0000	reserved
0001	reserved
0010	Sequence Display Extension ID
0011	Quant Matrix Extension ID
0100	Copyright Extension ID
0101	VLC code Extension ID
0110	reserved
0111	Picture Display Extension ID
1000 - 1010	reserved
1011	Camera Parameters Extension ID
1100	ITU-T extension ID
1101	reserved
…	…
1111	reserved

"

28) Add the following subclause 6.3.13 after subclause 6.3.12:

"

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 AMD1-2.

Table AMD1-2 -- 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 V2-39
0011	reserved
0100	object type listed in Table AMD1-48
0101 – 1111	reserved

visual_object_type: The visual_object_type is a 4-bit code given in Table AMD1-3 which identifies the type of the visual object.

Table AMD1-3 -- 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 AMD1-1.

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 AMD1-4. If the sequence_display_extension() is not present in the bitstream then the video format may be assumed to be "Unspecified video format".

Table AMD1-4 -- 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 AMD1-5.

Table AMD1-5 -- Colour Primaries

Value	Primaries		
0	(forbidden)		
1	Recommendation ITU-R BT.709		
	primary	x	y
	green	0,300	0,600
	blue	0,150	0,060
	red	0,640	0,330
	white D65	0,3127	0,3290
2	Unspecified Video Image characteristics are unknown.		
3	reserved		
4	Recommendation ITU-R BT.470-2 System M		
	primary	x	y
	green	0,21	0,71
	blue	0,14	0,08
	red	0,67	0,33
	white C	0,310	0,316
5	Recommendation ITU-R BT.470-2 System B, G		
	primary	x	y
	green	0,29	0,60
	blue	0,15	0,06
	red	0,64	0,33
	white D65	0,313	0,329
6	SMPTE 170M		
	primary	x	y
	green	0,310	0,595
	blue	0,155	0,070
	red	0,630	0,340
	white D65	0,3127	0,3290
7	SMPTE 240M (1987)		
	primary	x	y
	green	0,310	0,595
	blue	0,155	0,070
	red	0,630	0,340
	white D65	0,3127	0,3291
8	Generic film (colour filters using Illuminant C)		
	primary	x	y
	green	0,243	0,692 (Wratten 58)
	blue	0,145	0,049 (Wratten 47)
	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 AMD1-6.

Table AMD1-6 -- Transfer Characteristics

Value	**Transfer Characteristic**
0	(forbidden)
1	Recommendation ITU-R BT.709 $V = 1{,}099\ L_C^{0{,}45} – 0{,}099$ for $1 \geq L_C \geq 0{,}018$ $V = 4{,}500\ L_C$ for $0{,}018 > L_C \geq 0$
2	Unspecified Video Image characteristics are unknown.
3	reserved
4	Recommendation ITU-R BT.470-2 System M Assumed display gamma 2,2
5	Recommendation ITU-R BT.470-2 System B, G Assumed display gamma 2,8
6	SMPTE 170M $V = 1{,}099\ L_C^{0{,}45} – 0{,}099$ for $1 \geq L_C \geq 0{,}018$ $V = 4{,}500\ L_C$ for $0{,}018 > L_C \geq 0$
7	SMPTE 240M (1987) $V = 1{,}1115\ L_C^{0{,}45} – 0{,}1115$ for $L_C \geq 0{,}0228$ $V = 4{,}0\ L_C$ for $0{,}0228 > L_C$
8	Linear transfer characteristics i.e. $V = L_C$
9	Logarithmic transfer characteristic (100:1 range) $V = 1.0\text{-}Log_{10}(Lc)/2$ for $1 = L_C = 0.01$ $V = 0.0$ for $0.01 > L_C$
10	Logarithmic transfer characteristic (316.22777:1 range) $V = 1.0\text{-}Log_{10}(Lc)/2.5$ for $1 = L_C = 0.0031622777$ $V = 0.0$ 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 AMD1-7.

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.

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 AMD1-7 -- Matrix Coefficients

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

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().

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 AMD1-8 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 1/16ths of a **frame** line (not a line in the field).

3 In the example of Figure AMD1-8 both frame_centre_horizontal_offset and frame_centre_vertical_offset have negative values.

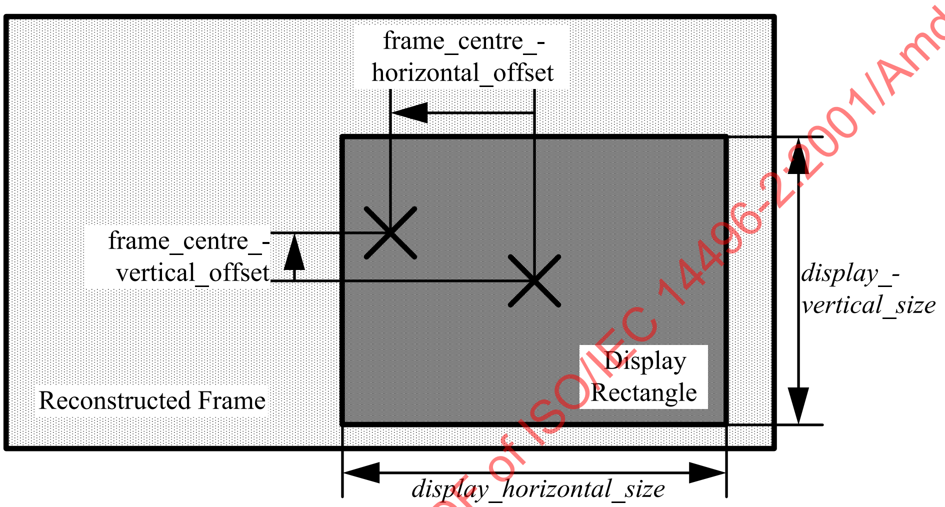


Figure AMD1-8 -- Frame centre offset parameters

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.

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 AMD1-9 explains these terms pictorially.

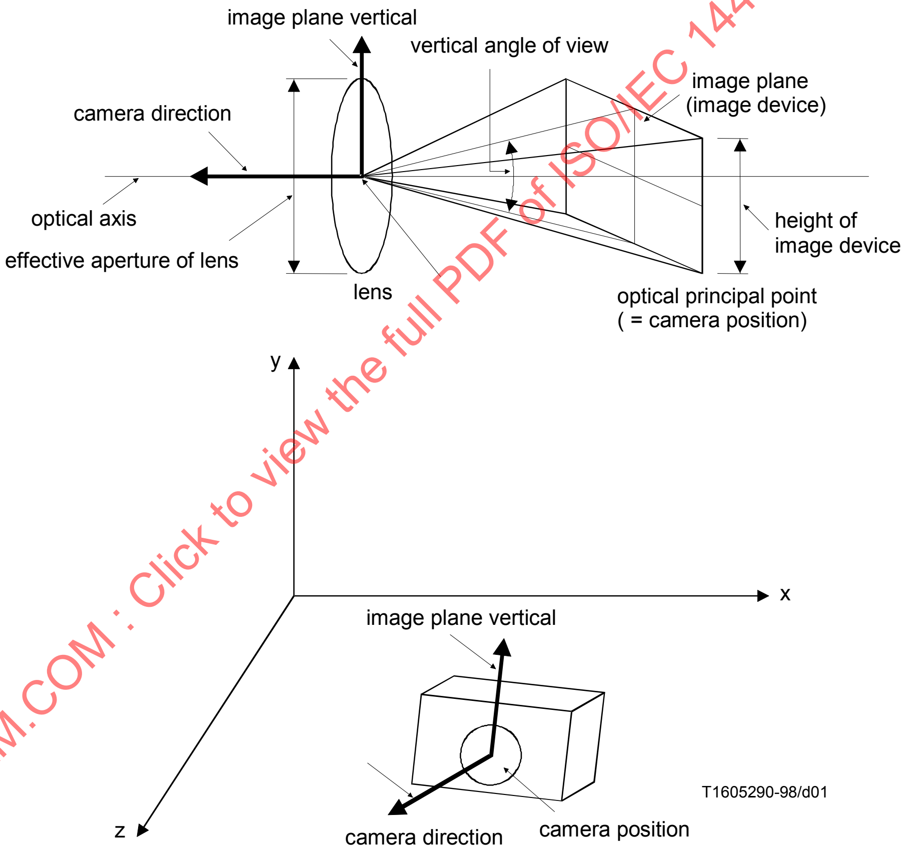


Figure AMD1-9 -- Camera parameters

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

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 AMD1-8.

Table AMD1-8 -- Meaning of load_vlc_code

load_vlc_code	Meaning
0000	forbidden

0001	load_intra_vlc_code = 1
	load_inter_vlc_code = 0

0010	load_intra_vlc_code = 0
	load_inter_vlc_code = 1

0011	load_intra_vlc_code = 1
	load_inter_vlc_code = 1

0100 – 1111	reserved

If load_intra_vlc_code is set to '1', the VLC codes defined in Table AMD1-55 to Table AMD1-66 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 AMD1-55 to Table AMD1-66.

If load_inter_vlc_code is set to '1', the VLC codes defined in Table AMD1-67 to Table AMD1-78 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 AMD1-67 to Table AMD1-78.

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 AMD1-55 to Table AMD1-66. 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 AMD1-55 to Table AMD1-66. 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 AMD1-67 to Table AMD1-78. 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 AMD1-67 to Table AMD1-78. 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 inTable AMD1-9.

Table AMD1-9 -- 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
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 AMD1-10. If both visual_object_verid and video_object_layer_verid exist, the semantics of video_object_layer_verid supersedes the other.

Table AMD1-10 -- 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 V2-39
0011	reserved
0100	object type listed in Table AMD1-48
0101 – 1111	reserved

video_object_layer_shape: This is a 2-bit integer defined in Table AMD1-11. It identifies the shape type of a video object layer.

Table AMD1-11 -- 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 AMD1-12. 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 AMD1-11. 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'.

Table AMD1-12 -- 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 AMD1-13. If rgb_components is set to '1', this shall be set to '11'.

Table AMD1-13 -- 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.

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 AMD1-14 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 AMD1-14 -- 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 AMD1-15.

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.

Table AMD1-15 -- 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 indicating if the decoded bitstream is transcoded from one which conforms to ISO/IEC 13818-2: 1996 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]: This 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.

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 AMD1-16 shows the various choices.

Table AMD1-16 -- 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 AMD1-17 shows the meaning of various codewords

Table AMD1-17 -- Meaning of sprite warping accuracy codewords

code	sprite_warping_accuracy
00	½ pixel
01	¼ 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.

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.

time_code_smpte12m: See annex P.

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.

6.3.13.5 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.

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 AMD1-18.

Table AMD1-18 -- 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 AMD1-19.

Table AMD1-19 -- 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)

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:

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 according to Table AMD1-20. This flag affects the inverse quantisation process as defined in 7.16.4.3.

Table AMD1-20 – DCT precision

dct_precision	base_quantiser
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 AMD1-21.

Table AMD1-21 -- 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 AMD1-42 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 AMD1-22. 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 AMD1-22 – alpha base quantiser

alpha_dct_precision	alpha_base_quantiser
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 AMD1-23.

Table AMD1-23 – 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

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 AMD1-24.

Table AMD1-24 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 AMD1-25.

Table AMD1-25 Definition of sub_carrier_phase.

sub_carrier_phase	Phase
0	$([360^{o} \div 256] * 0)$
1	$([360^{o} \div 256] * 1)$
…	…
255	$([360^{o} \div 256] * 255)$

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 AMD1-93.

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 AMD1-94.

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 AMD1-10.

$$P_s(x,y) = \begin{cases} 1/4a^2 & |x| \le a, |y| \le a \\ 0 & else \end{cases}$$

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.

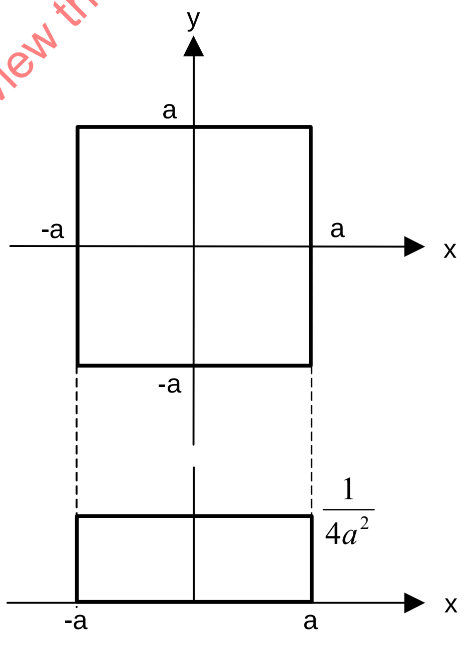


Figure AMD1-10 – Defocusing filter

The length of this value varies according to the sprite warping accuracy. Table AMD1-26 shows the length of the defocusing control parameter.

Table AMD1-26 -- 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

$$u = 1 + R(k_1 \cdot \tfrac{1}{4W^3} + Rk_2 \cdot \tfrac{1}{2W^5})$$
$$R = (\tilde{x} - c_x)^2 + s^2(\tilde{y} - c_y)^2$$

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 AMD1-27, the code itself is simply a binary representation of the macroblock number.

Table AMD1-27 -- 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

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 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 AMD1-28). 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 AMD1-28-- 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 AMD1-88 - Table AMD1-91. 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 AMD1-29. 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 AMD1-80 is used. For P-VOPs, the context-switched table of Table AMD1-81 is used.

Table AMD1-29 -- 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 AMD1-82.

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 AMD1-83, otherwise the VLC table of Table AMD1-82 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 AMD1-84.

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 AMD1-85.

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 AMD1-86.

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 AMD1-50 and Table AMD1-51, selected by vop_coding_type.

macroblock_quant: Derived from macroblock_type according to the Table AMD1-50 and Table AMD1-51. This is set to 1 to indicate that quantiser_scale_code is present in the bitstream.

macroblock_motion_forward: Derived from macroblock_type according to the Table AMD1-50 and Table AMD1-51. This flag affects the bitstream syntax and is used by the decoding process.

macroblock_pattern: Derived from macroblock_type according to the Table AMD1-50 and Table AMD1-51. 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 AMD1-50 and Table AMD1-51. 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 AMD1-30.

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 AMD1-30 -- 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 AMD1-31.

Table AMD1-31 -- 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 AMD1-32.

Table AMD1-32 -- 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 AMD1-30 and Table AMD1-31.

mv_format is derived from field_motion_type or frame_motion_type as indicated in the Table AMD1-30 and Table AMD1-31. 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 Table AMD1-42 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 Table AMD1-92, which is used in motion vector decoding as described in 7.16.5.4.1. (See Table AMD1-42 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 Table AMD1-42 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 AMD1-87.

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:

```
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 Figures 6-5, AMD1-4 and AMD1-5 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 AMD1-33.

Table AMD1-33 -- 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.

6.3.13.9 Studio Block

dct_dc_size_luminance: This is a variable length code as defined in Table AMD1-52 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 AMD1-54-1 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 AMD1-53 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.

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.

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.

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 AMD1-34 -- 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

"

29) Replace paragraph 2 in clause 7 with the following:

"

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 video object decoding for the studio profile is specified in subclause 7.16. The still texture object decoding is described in subclause 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.

"

30) Add the following subclause 7.16 after subclause 7.15:

"

7.16 Video object decoding for the studio profile

This subclause specifies the video object decoding for the studio profile.

7.16.1 Video decoding process

(The identical description to clause 7.1)

7.16.2 Higher syntactic structures

The various parameters and flags in the bitstream for StudioVideoObjectLayer(), Group_of_StudioVideoObjectPlane(), StudioVideoObjectPlane(), StudioSlice(), StudioMacroblock(), StudioBlock() and StudioDPCMBlock(), as well as other syntactic structures related to them shall be interpreted as discussed 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.

Texture data in a macroblock shall have been encoded by either DCT or DPCM.

If compression_mode == 1, texture data in the macroblock shall have been encoded by DCT. The decoding process is indicated in subclause 7.16.4

If compression_mode == 0, texture data in the macroblock shall have been encoded by DPCM. The decoding process is indicated in subclause 7.16.5

Reconstructed fields shall be associated together in pairs to form reconstructed frames. (See “vop_structure” in 6.3.13.5.)

The sequence of reconstructed frames shall be reordered as described in 6.3.13.5.

If progressive_sequence == 1 the reconstructed frames shall be output from the decoding process at regular intervals of the frame period as shown in Figure AMD1-26.

If progressive_sequence == 0 the reconstructed frames shall be broken into a sequence of fields which shall be output from the decoding process at regular intervals of the field period as shown in Figure AMD1-27. In the case that a frame picture has repeat_first_field == 1 the first field of the frame shall be repeated after the second field. (See “repeat_first_field” in 6.3.13.5.)

7.16.3 VOP reconstruction

The luminance and chrominance values of a VOP from the decoded texture and motion information are reconstructed as indicated in this subclause. Figure AMD1-11 represents the process.

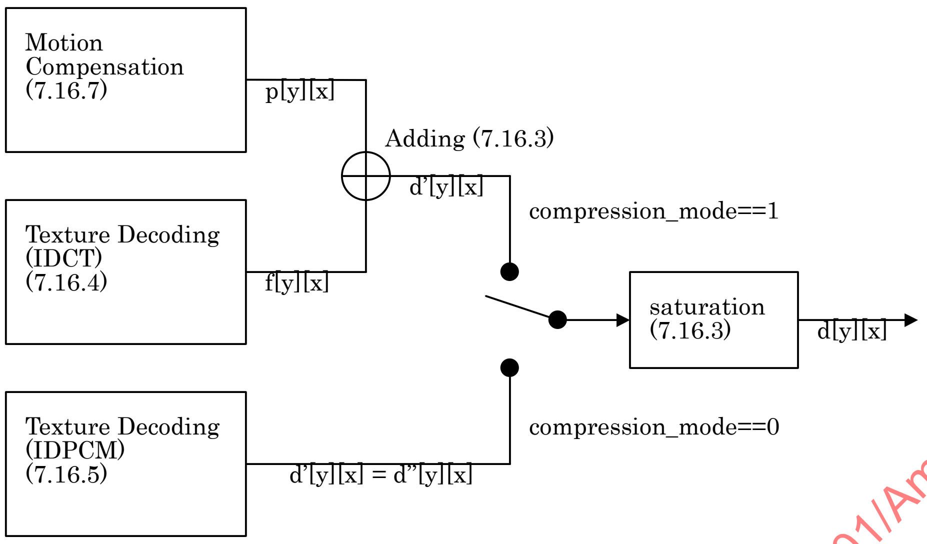


Figure AMD1-11 – VOP reconstruction from the decoded texutre and motion compensated data

1. In case of INTRA macroblocks of compression_mode==1, the luminance and chrominance values f[y][x] from the decoded texture data form the luminance and chrominance values: 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 luminance and chrominance values: d'[y][x] = p[y][x] + f[y][x]

3. In case of INTRA macroblocks of compression_mode==0, the luminance and chrominance values d''[y][x] from the decoded texture data form the luminance and chrominance values: d'[y][x] = d''[y][x].

4. Finally, the calculated luminance and chrominance values are saturated so that

$$d[y][x] = \begin{cases} 2^{bits\_per\_pixel} - 1; & d'[y][x] > 2^{bits\_per\_pixel} - 1 \\ d'[y][x]; & 0 \leq d'[y][x] \leq 2^{bits\_per\_pixel} - 1 \\ 0; & d'[y][x] < 0 \end{cases}$$

NOTE : The saturation defined above limits output sample values for Y, Cr and Cb to the range [0:$2^{\text{bits\_per\_pixel}}$-1]. Therefore, the values which are assigned as the reserved code words for timing reference in ITU-R BT.601 and BT.709 can occasionally occur.

7.16.4 Texture decoding from DCT coefficients

This subclause describes the process used to decode the texture information of a VOP in case of compression_mode==1. The process of video texture decoding from coded DCT coefficients is given in Figure AMD1-12.

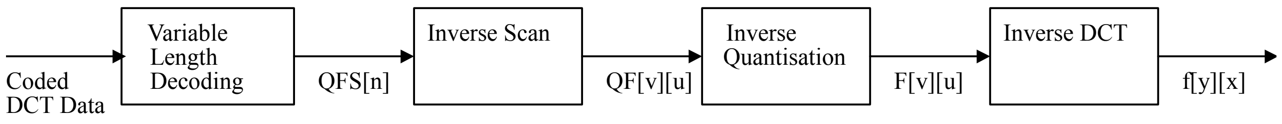


Figure AMD1-12 – Video Texture Decoding Process from DCT coefficients

7.16.4.1 Variable length decoding

This subclause explains the decoding process. Subclause 7.16.4.1.1 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 7.16.4.1.2 specifies the decoding process for all other coefficients; AC coefficients ($n \neq 0$) and DC coefficients in non-intra coded blocks.

Let *cc* denote the colour component. It is related to the block number as specified in Table AMD1-35. In case of rgb_components==0, *cc* is zero for the Y component, one for the C_b component and two for the C_r component. In case of rgb_components==1, zero for the G component, one for the B component and two for the R component.

Table AMD1-35 -- Definition of *cc*, colour component index

	cc		
Block Number	**4:2:0**	**4:2:2**	**4:4:4**
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	1	1	1
5	2	2	2
6	▨	1	1
7	▨	2	2
8	▨	▨	1
9	▨	▨	2
10	▨	▨	1
11	▨	▨	2

7.16.4.1.1 DC coefficients decoding in intra blocks

Differential DC coefficients in blocks in intra macroblocks are decoded as a variable length code denoting dct_dc_size as defined in Table AMD1-52 and Table AMD1-53 in annex B, and a fixed length code dct_dc_differential (Table AMD1-54-1). 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 AMD1-54-1). This is done by appending a fixed length code, dct_dc_differential, of dct_dc_size bits.

In case of rgb_components==0, if *cc* is zero then Table AMD1-52 shall be used for dct_dc_size. If *cc* is non-zero then Table AMD1-53 shall be used for dct_dc_size.

In case of rgb_components==1, Table AMD1-52 shall be used for dct_dct_size for all of the components, *cc*.

Three predictors are maintained, one for each of the colour components, *cc*. Each time a DC coefficient in a block in an intra macroblock is decoded the predictor is added to the differential to recover the actual coefficient. Then the predictor shall be set to the value of the coefficient just decoded. At various times, as described below, the predictors shall be reset. The reset value except when a macroblock is encoded as DPCM residuals is derived from the combination of the parameters bits_per_pixel , dct_precision and intra_dc_precision as;

2 ^ (bits_per_pixel+dct_precision+intra_dc_precision-1)

The predictors shall be reset to the above reset value at the following times:

- Whenever a macroblock is decoded if intra_predictors_reset == 1.
- At the start of a slice.
- Whenever a non-intra macroblock is decoded.
- Whenever a macroblock is skipped. i.e. not_coded == 1.
- Whenever a macroblock and a block is transparent.

When a macroblock is encoded as DPCM residuals, the reset value is derived as;

block_mean x (2 ^ (dct_precision+intra_dc_precision))

Because block_mean is encoded for each component independently, the reset value for each component can be different.

The predictors are denoted *dct_dc_pred*[*cc*].

QFS[0] shall be calculated from dct_dc_size and dct_dc_differential by any process equivalent to:

```
if ( dct_dc_size == 0 ) {
    dct_diff = 0;
} else {
    half_range = 2 ^ ( dct_dc_size - 1 );
    if ( dct_dc_differential >= half_range )
        dct_diff = dct_dc_differential;
    else
        dct_diff = (dct_dc_differential + 1) - (2 * half_range);
}
QFS[0] = dct_dc_pred[cc] + dct_diff;
dct_dc_pred[cc] = QFS[0]
```

NOTE 1 - The symbol ^ *denotes power (not XOR)*.

NOTE 2 - *dct_diff* and *half_range* are temporary variables which are not used elsewhere in this specification.

It is a requirement of the bitstream that QFS[0] shall lie in the range:

0 to ((2^(dct_precision + bits_per_pixel + intra_dc_precision))-1)

7.16.4.1.2 Other coefficients

All coefficients with the exception of the DC intra coefficients shall be decoded using a set of 12 VLC tables, T[0] to T[11] . The default set of the 12 tables is defined in Table AMD1-55 to Table AMD1-66 for intra macroblocks and Table AMD1-67 to 78 for inter macroblocks. The set of tables can be downloaded using the extension syntax of vlc_code_extension().

The decision rule to select one table from the 12 tables for decoding a symbol is defined in this subclause.

In all cases a variable length code shall first be decoded using one of the tables. The decoded value of this code denotes one of three courses of action:

1) End of Block. In this case there are no more coefficients in the block in which case the remainder of the coefficients in the block (those for which no value has yet been decoded) shall be set to zero. This is denoted by “End of block” in the syntax specification of 6.2.13.9.

2) A “normal” coefficient. In which a symbol denoting a value of *run* and/or *level* is decoded followed by a fixed length code of the size indicated by the variable length code according to Table AMD1-36. The variable length code categorizes symbols according to the value of *run* and/or *level* as indicated in Table AMD1-36. For each category a fixed length code is appended to the variable length code to uniquely identify which value in that category actually occurred. The additional codes are defined in Table AMD1-54-2~AMD1-54-4. For decoding group No.1~6, Table AMD1-54-2 shall be used. For decoding group No.7~12, Table AMD1-54-3 shall be used. For decoding group No.13~20, Table AMD1-54-4 shall be used.

3) An “Escape” coded coefficient. In which a value of *level* is fixed length coded as described in 7.16.4.1.3.

The VLC table which is used to decode a variable length code for the current symbol shall be decided from the 12 tables according to the immediately previous symbol which has already been decoded in a zigzag scanning order. After decoding the current symbol, it is used to decide the VLC table for decoding the next symbol from the 12 tables.

The relation of the value of the current symbol and the VLC table used for decoding a variable length code for the next symbol is indicated in Table AMD1-36.

The relation of the value of the previous symbol and the VLC table for decoding the current symbol is shown in Table AMD1-36. A variable length code for the current symbol is decoded using one of the 12 tables, and the variable length code indicates the group (0~21) that the value of the current symbol belongs to. Depending on the group, the size of a fixed length code used to identify the value of the current symbol and the VLC table for decoding the next symbol is decided. For the first non DC coefficient in a block the VLC table T[0] shall be used as the initial table.

Table AMD1-36 -- Categorizing ac symbol into groups

group no.	symbols of DCT AC coefficients (zigzag scanning order)	the size of an additional code	VLC table for the next symbol
0	EOB	0	---
1	0*1	0	T[1]
2	0*2 to 0*3	1	T[1]
3	0*4 to 0*7	2	T[1]
4	0*8 to 0*15	3	T[1]
5	0*16 to 0*31	4	T[1]
6	0*32 to 0*63	5	T[1]
7	0*1[1,-1]	1	T[2]
8	0*2[1,-1] to 0*3[1,-1]	2	T[2]
9	0*4[1,-1] to 0*7[1,-1]	3	T[2]
10	0*8[1,-1] to 0*15[1,-1]	4	T[2]
11	0*16[1,-1] to 0*31[1,-1]	5	T[2]
12	0*32[1,-1] to 0*63[1,-1]	6	T[2]
13	-1, or 1	1	T[3]
14	-3 to –2, or 2 to 3	2	T[4]
15	-7 to –4, or 4 to 7	3	T[5]
16	-15 to –8, or 8 to 15	4	T[6]
17	-31 to –16, or 16 to 31	5	T[7]
18	-63 to –32, or 32 to 63	6	T[8]
19	-127 to –64, or 64 to 127	7	T[9]
20	-255 to –128, or 128 to 255	8	T[10]
21	$-(2^{\text{bits\_per\_pixel+dct\_precision+3}}-1)$ to –256, or 256 to $(2^{\text{bits\_per\_pixel+dct\_precision+3}}-1)$ (escape code)	bits_per_pixel + dct_precision + 4	T[11]

Note : Table T[0] shall be used for the first non DC coefficient in a block.

The VLC code for group '0' of T[0] for inter macroblocks shall not be assigned because there shall be at least one non zero DCT coefficient in an inter block which is indicated as 'coded' by pattern_code[].

The VLC code for group '0'~'13' of T[1] shall not be assigned because there shall be at least one *level* value , the absolute value of which is more than '1', just after group '1'~'6' occurred.

7.16.4.1.3 Escape coding

Level values, the absolute value of that is more then 255, are encoded by using the escape code. The escape code that is assigned to group No.21 is followed by a fixed length code of the length ‘bits_per_pixel+dct_precision+4’. The following equation defines the relation among the fixed length code, the length and *level* to be decoded.

```
    if flc>>(flclen-1) is 1
            level = -1 * ( (  flc^((1<<flclen)-1)  ) + 1)
    else    level = flc
```

Where, flc and flclen denote the fixed length code and its length. The symbol ^ denotes XOR in this subclause.

7.16.4.2 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 QF[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.

Two scan patterns are defined. The scan that shall be used shall be determined by alternate_scan which is encoded in StudioVideoObjectPlane().

Figure AMD1-13 defines *scan*[alternate_scan][*v*][*u*] for the case that alternate_scan is zero. Figure AMD1-14 defines *scan*[alternate_scan][*v*][*u*] for the case that alternate_scan is one.

									u
		0	1	2	3	4	5	6	7
	0	0	1	5	6	14	15	27	28
	1	2	4	7	13	16	26	29	42
	2	3	8	12	17	25	30	41	43
	3	9	11	18	24	31	40	44	53
	4	10	19	23	32	39	45	52	54
	5	20	22	33	38	46	51	55	60
	6	21	34	37	47	50	56	59	61
v	7	35	36	48	49	57	58	62	63

Figure AMD1-13 -- Definition of *scan*[0][*v*][*u*]

									u
		0	1	2	3	4	5	6	7
	0	0	4	6	20	22	36	38	52
	1	1	5	7	21	23	37	39	53
	2	2	8	19	24	34	40	50	54
	3	3	9	18	25	35	41	51	55
	4	10	17	26	30	42	46	56	60
	5	11	16	27	31	43	47	57	61
	6	12	15	28	32	44	48	58	62
v	7	13	14	29	33	45	49	59	63

Figure AMD1-14 -- Definition of *scan*[1][*v*][*u*]

The inverse scan shall be any process equivalent to the following:

```
for ($v$=0; $v$<8; $v$++)
    for ($u$=0; $u$<8; $u$++)
        $QF[v][u] = QFS[scan[\text{alternate\_scan}][v][u]]$
```

NOTE - The scan patterns defined here are often referred to as “zigzag scanning order”.

7.16.4.2.1 Inverse scan for matrix download

When the quantisation matrices are downloaded they are encoded in the bitstream in a scan order that is converted into the two-dimensional matrix used in the inverse quantiser in an identical manner to that used for coefficients.

For matrix download the scan defined by Figure AMD1-13 (i.e. *scan*[0][*v*][*u*]) shall always be used.

Let *W*[*w*][*v*][*u*] denote the weighting matrix in the inverse quantiser (see 7.16.4.3.2.1), and *W'*[*w*][*n*] denote the matrix as it is encoded in the bitstream. The matrix download shall then be equivalent to the following:

```
for ($v$=0; $v$<8; $v$++)
    for ($u$=0; $u$<8; $u$++)
        $W[w][v][u] = W'[w][scan[0][v][u]]$
```

7.16.4.3 Inverse quantisation

The two-dimensional array of coefficients, *QF*[*v*][*u*], is inverse quantised to produce the reconstructed DCT coefficients. This process is essentially a multiplication by the base quantiser and the quantiser step size. The quantiser step size is modified by two mechanisms; a weighting matrix is used to modify the step size within a block and a scale factor 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 weighting matrix). In case of mpeg2_stream==0, the base quantiser is used to adjust the dynamic range of the output of the inverse quantisation to the range $[-2^{\text{bits\_per\_pixel}+6}, 2^{\text{bits\_per\_pixel}+6}-1]$. In case of mpeg2_stream==1, the base quantiser shall not affect the decoding process below.

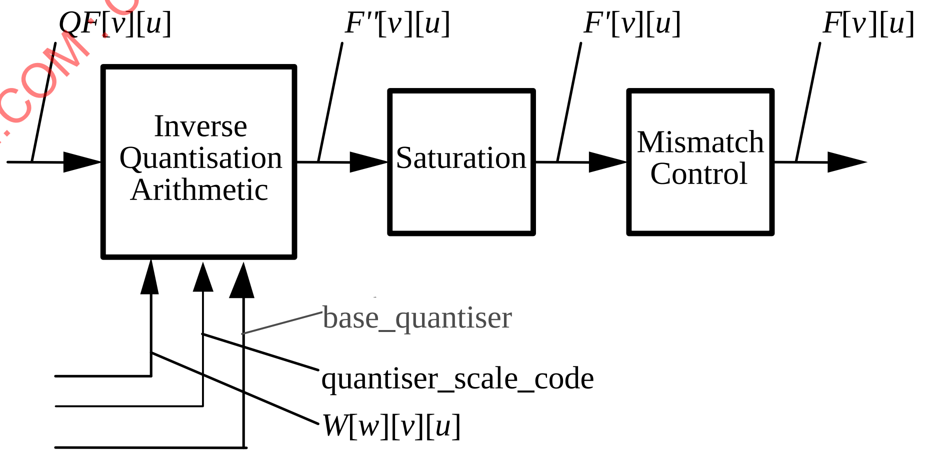


Figure AMD1-15 -- Inverse quantisation process

Figure AMD1-15 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*].

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: 1996.

7.16.4.3.1 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, *intra_dc_mult*, (constant in the sense that it is not modified by either the weighting matrix or the scale factor). The multiplier is related to the parameters base_quantiser and intra_dc_precision that are encoded in StudioVideoObjectPlane(). Table AMD1-37 specifies the relation between intra_dc_precision, base_quantiser and *intra_dc_mult*. This relation changes according to the value of mpeg2_stream. The precision of the intra dc coefficient is defined as 'bits_per_pixel+dct_precision+intra_dc_precision'.

Table AMD1-37 (a) In case of mpeg2_stream==0:

Reletion among intra_dc_precision, base_quantiser and *intra_dc_mult*

intra_dc_precision	*intra_dc_mult*
0	8 x base_quantiser x 8
1	4 x base_quantiser x 8
2	2 x base_quantiser x 8
3	1 x base_quantiser x 8

Table AMD1-37(b) In case of mpeg2_stream==1:

Reletion between intra_dc_precision, *intra_dc_mult*

intra_dc_precision	*intra_dc_mult*
0	8
1	4
2	2
3	1

The reconstructed DC values are computed as follows.

F''[0][0] = intra_dc_mult* QF[0][0]

7.16.4.3.2 Other coefficients

All coefficients other than the DC coefficient of an intra block shall be inverse quantised as specified in this subclause.

7.16.4.3.2.1 Weighting matrices

In the 4:2:0 format, two weighting matrices are used. One shall be used for intra macroblocks and the other for non-intra macroblocks. In the 4:2:2 or 4:4:4 format, four matrices are used allowing different matrices to be used for luminance and chrominance data. Each matrix has a default set of values which may be overwritten by down-loading a user defined matrix as explained in 6.2.13.2.4.

Let the weighting matrices be denoted by *W*[*w*][*v*][*u*] where *w* takes the values 0 to 3 indicating which of the matrices is being used. Table AMD1-38 summarises the rules governing the selection of *w*.

Table AMD1-38 -- Selection of *w*

	4:2:0		**4:2:2 and 4:4:4**	
	luminance (cc = 0)	**chrominance (cc ≠ 0)**	**luminance (cc = 0)**	**chrominance (cc ≠ 0)**
intra blocks (macroblock_intra = 1)	0	0	0	2
non-intra blocks (macroblock_intra = 0)	1	1	1	3

7.16.4.3.2.2 Quantiser scale factor

The quantisation scale factor is decoded as a 5 bit fixed length code, quantiser_scale_code. This indicates the appropriate *quantiser_scale* to apply in the inverse quantisation arithmetic.

q_scale_type (encoded in StudioVideoObjectPlane()) indicates which of two mappings between quantiser_scale_code and *quantiser_scale* shall apply. Table AMD1-39 shows the two mappings between quantiser_scale_code and *quantiser_scale*.

Table AMD1-39 -- Relation between *quantiser_scale* and quantiser_scale_code

quantiser_scale_code	quantiser_scale[q_scale_type]	
	q_scale_type = 0	q_scale_type = 1
0	(forbidden)	
1	2	1
2	4	2
3	6	3
4	8	4
5	10	5
6	12	6
7	14	7
8	16	8
9	18	10
10	20	12
11	22	14
12	24	16
13	26	18
14	28	20
15	30	22
16	32	24
17	34	28
18	36	32
19	38	36
20	40	40
21	42	44
22	44	48
23	46	52
24	48	56
25	50	64
26	52	72
27	54	80
28	56	88
29	58	96
30	60	104
31	62	112

7.16.4.3.2.3 Reconstruction formulae

The following equation specifies the arithmetic to reconstruct *F''*[*v*][*u*] from *QF*[*v*][*u*] (for all coefficients except intra DC coefficients) in the case of mpeg2_stream==0.

$$F''[v][u] = ((2 \times QF[v][u] + k) \times W[w][v][u] \times quantiser\_scale \times base\_quantiser \times 8\ ) / 32$$

where:

$$k = \begin{cases} 0 & \text{intra blocks / non-intra blocks if dead\_zone\_disable==1} \\ Sign(QF[v][u]) & \text{non-intra blocks if dead\_zone\_disable==0} \end{cases}$$

NOTE - The above equation uses the “/” operator as defined in 4.1.

This equation changes in the case of mpeg2_stream==1 as following:

$$F''[v][u] = ((2 \times QF[v][u] + k) \times W[w][v][u] \times quantiser\_scale) / 32$$

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 4.1.

7.16.4.3.3 Saturation

The coefficients resulting from the Inverse Quantisation Arithmetic are saturated to lie in the range $[-2^{\text{bits\_per\_pixel} + 6}, 2^{\text{bits\_per\_pixel} + 6} - 1]$ in the case of mpeg2_stream==0. Thus:

$$F'[v][u] = \begin{cases} 2^{\text{bits\_per\_pixel}+6} - 1 & F''[v][u] > 2^{\text{bits\_per\_pixel}+6} - 1 \\ F''[v][u] & -2^{\text{bits\_per\_pixel}+6} \le F''[v][u] \le 2^{\text{bits\_per\_pixel}+6} - 1 \\ -2^{\text{bits\_per\_pixel}+6} & F''[v][u] < -2^{\text{bits\_per\_pixel}+6} \end{cases}$$

The coefficients resulting from the Inverse Quantisation Arithmetic are saturated to lie in the range $[-2^{\text{bits\_per\_pixel} + 3}, 2^{\text{bits\_per\_pixel} + 3} - 1]$ in the case of mpeg2_stream==1. Thus:

$$F'[v][u] = \begin{cases} 2^{\text{bits\_per\_pixel}+3} - 1 & F''[v][u] > 2^{\text{bits\_per\_pixel}+3} - 1 \\ F''[v][u] & -2^{\text{bits\_per\_pixel}+3} \le F''[v][u] \le 2^{\text{bits\_per\_pixel}+3} - 1 \\ -2^{\text{bits\_per\_pixel}+3} & F''[v][u] < -2^{\text{bits\_per\_pixel}+3} \end{cases}$$

7.16.4.3.4 Mismatch control

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:

$$sum = \sum_{v=0}^{v<8} \sum_{u=0}^{u<8} F'[v][u]$$

$$F[v][u] = F'[v][u] \text{ for all } u,\ v \text{ except } u = v = 7$$

$$F[7][7] = \begin{cases} F'[7][7] & \text{if } sum \text{ is odd} \\ \begin{Bmatrix} F'[7][7] - 1 & \text{if } F'[7][7] \text{ is odd} \\ F'[7][7] + 1 & \text{if } F'[7][7] \text{ is even} \end{Bmatrix} & \text{if } sum \text{ is even} \end{cases}$$

NOTE 1 It may be useful to note that the above correction for *F*[7][7] may simply 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.

7.16.4.3.5 Summary of quantiser process

In summary, the inverse quantisation process is any process numerically equivalent to:

```
for ($v$=0; $v$<8;$v$++) {
    for ($u$=0; $u$<8;$u$++) {
        if ( ($u$==0) && ($v$==0) && (macroblock_intra) ) {
            $F''[v][u] = intra\_dc\_mult * QF[v][u]$;
        } else {
            if ( macroblock_intra ) {
                $F''[v][u]$ = ( $QF[v][u] * W[0][v][u] * quantiser\_scale * base\_quantiser$ * 8 * 2 ) / 32;
            } else {
                $F''[v][u]$ = ( ( ( $QF[v][u]$ *  2 ) + $Sign(QF[v][u])$ ) * $W[1][v][u]$
                                        * $quantiser\_scale * base\_quantiser$ * 8 ) / 32;
            }
        }
    }
}
$sum$ = 0;
for ($v$=0; $v$<8;$v$++) {
    for ($u$=0; $u$<8;$u$++) {
        if ( $F''[v][u] > 2^{\text{bits\_per\_pixel + 6}} - 1$ ) {
            $F'[v][u] = 2^{\text{bits\_per\_pixel + 6}} - 1$;
        } else {
            if ( $F''[v][u] < -2^{\text{bits\_per\_pixel + 6}}$ ) {
                $F'[v][u] = -2^{\text{bits\_per\_pixel + 6}}$ ;
            } 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;
            }
    }
```

7.16.4.4 Inverse DCT

Once the DCT coefficients, F[u][v] are reconstructed, an IDCT transform that conforms to the specifications of Annex A shall be applied to obtain the inverse transformed values, $f[y][x]$. In the case of mpeg2_stream==0, the decimal point of F[u][v] is shifted 3bits to the left in the binary scale in order to adjust the decimal point of the IDCT input. In the case of mpeg2_stream==1, the reconstructed coefficients are directly input to an IDCT function without the shift process. The inverse transformed values shall be saturated so that: $-2^{\text{bits\_per\_pixel}} \leq f[y][x] \leq 2^{\text{bits\_per\_pixel}} - 1$, for all x, y.

7.16.4.4.1 Non-coded blocks and skipped macroblocks

In a macroblock that is not skipped, if pattern_code[i] is one for a given block in the macroblock, then coefficient data is included in the bitstream for that block. This is decoded as specified in the preceding clauses.

However, if pattern_code[i] is zero, or if the macroblock is skipped (not_coded==1), then that block contains no coefficient data. The sample domain coefficients f[y][x] for such a block shall all take the value zero.

7.16.5 Texture decoding from DPCM redisuals

If the DPCM mode is selected each non-transparent block is decoded as follows. First the **rice_parameter** field is decoded. It indicates the length of each subsequent **rice_suffix_code** field. The block is scanned line by line, row by row, and the following process is applied to each pixel. An unsigned residual is retrieved by a combination of the **rice_prefix_code**, **rice_suffix_code** and **dpcm_residual** fields. If **rice_prefix_code** indicates the escape mode, the unsigned residual is given by **dpcm_residual**. Otherwise the unsigned residual is given by (**rice_prefix_code**<<**rice_parameter**)+**rice_suffix_code**.

Then the unsigned residual is mapped to a signed residual according to the following transformation:

x → x>>1 if x is even

x → -x>>1 if x is odd

A prediction is computed based on the pixels that lie directly to the left ‘a’, directly above ‘b’, and directly left above ‘c’. If any of these pixels lies outside of the block boundary it is given the default value of $2^{\text{bits\_per\_pixel-1}}$. The prediction ‘p’ is computed as:

p = a+b-c

if (p < min(a,b)) p = min(a,b)

if (p > max(a,b)) p = max(a,b)

A second prediction p2 is computed as:

p2 = (min(a,b,c)+max(a,b,c)) / 2

If p2 is equal to p, p2 is assigned the value of **block_mean**.

If p2 is larger than p, the sign of the residual is inverted. Otherwise no operation is applied to the residual.

Finally the reconstructed pixel value, d’’[y][x], is obtained by adding the signed residual and the prediction p, modulo $2^{\text{bits\_per\_pixel}}$.

7.16.6 Shape decoding

Binary shape decoding uses a block-based representation. 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.

7.16.6.1 Higher syntactic structures

7.16.6.1.1 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.

7.16.6.1.2 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. If vop_structure is decoded as field structure, both values are definded in the absolute field coordinates.

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. The absolute frame coordinate system must be used for all frame VOPs, while the absolute field coordinate system must be used for all field VOPs. Additionally, the decoded spatial references must have an even value.

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.

7.16.6.2 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.

7.16.6.2.1 Mode decoding

Each bab belongs to one of five types listed in Table AMD1-40. The type information is given by the bab_type field which influences decoding of further shape information.

Table AMD1-40 -- List of bab types

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

7.16.6.2.1.1 I-VOPs

In this specification (Studio Profile), only five bab_types are adopted independently of vop_type.

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 a previously decoded bab_type.

C = 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`). bab_type of the bab outside the current slice 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 AMD1-80. The context C shall be set to zero for all macroblocks in a VOP in case of intra_predictors_reset==1 or for a macroblock located at the start of a slice.

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, which is called HHC. Decoding for HHC is described in subclause 7.16.6.2.5.

7.16.6.2.1.2 P-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 a forward reference VOP. The forward reference VOP is defined as the most recent non-empty (i.e. vop_coded != 0) I- or P-VOP in the past. If the current VOP is a P-VOP, the forward reference VOP is selected as the reference VOP.

If the current VOP is a field VOP of the interlaced sequence, the forward reference VOP is defined as the most recent non empty I- or P- field VOP of the same position (top or bottom field) in the past. When the second field of the first frame in GOV is coded as P-VOP, the reference VOP is the first field coded as I-VOP.

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 AMD1-16 where both rules are applied.

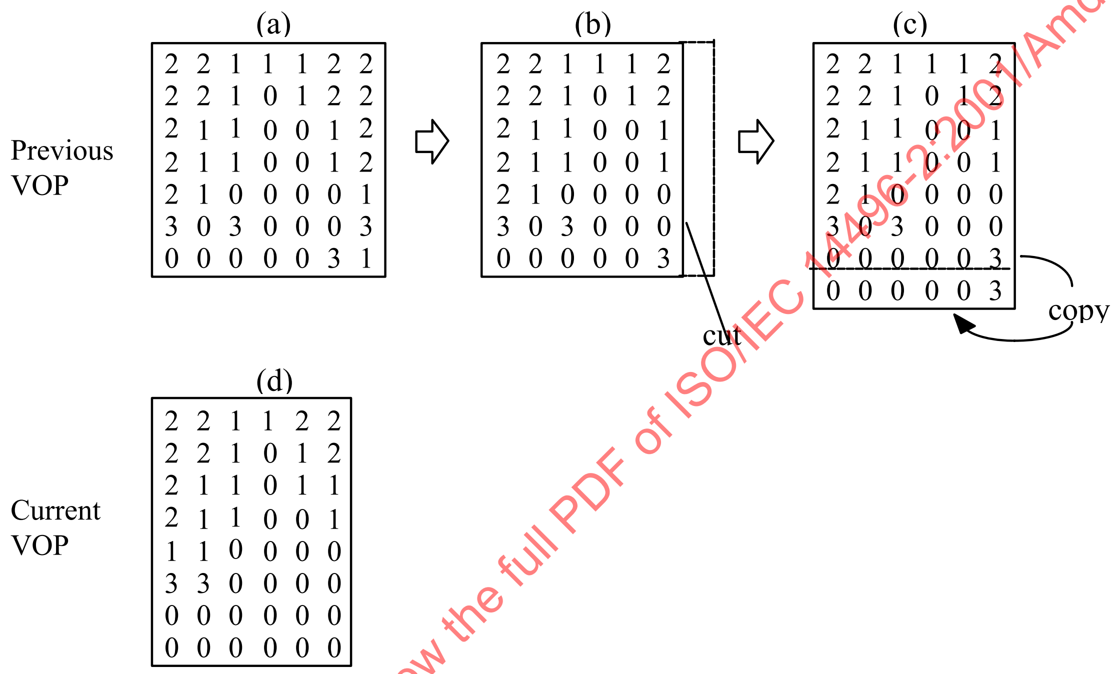


Figure AMD1-16 -- Example of size fitting between current VOP and reference VOP. The numbers represent the type of each bab

The VLC to decode the current bab_type is switched according to the bab_type value of the co-located bab in the reference VOP. These context-switched VLC tables for P-VOPs are given in Table AMD1-81. 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 AMD1-41. Decoding for motion compensation is described in subclause 7.16.6.2.4, and HHC decoding in subclause 7.16.6.2.5.

Table AMD1-41 -- Decoder components applied for each type of bab

bab_type	Motion compensation	HHC
0	yes	no
1	yes	no
2	no	no
3	no	no
4	no	yes

7.16.6.2.2 Binary alpha block motion compensation

Motion Vector of a shape (MVs) is used for motion compensation (MC) of the shape. The value of MVs is reconstructed as described in subclause 7.16.6.2.3. Integer pixel motion compensation is carried out on a 16x16 block basis according to subclause 7.16.6.2.4.

If bab_type is MVs==0 && No Update or MVs!=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, HHC decoding is required.

7.16.6.2.3 Motion vector decoding

If bab_type indicates that MVs!=0, then mvs_x and mvs_y are VLC decoded. For decoding mvs_x, the VLC given in Table AMD1-82 is used. The same table is used for decoding mvs_y, unless the decoded value of mvs_x is zero. If mvs_x == 0, the VLC given in Table AMD1-83 is used for decoding mvs_y. If bab_type indicates that MVs==0, then both mvs_x and mvs_y are set to zero.

7.16.6.2.4 Motion compensation

For inter mode babs (bab_type = 0 or 1), 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 VOP for which vop_coded is not equal to ‘0’. If the displaced position is outside the bounding rectangle, then these pixels are assumed to be “transparent”.

If the macroblock is in the field VOP, a displayed block is copied from previously decoded field I-VOP of the same position (top or bottom field). When the second field of the first frame in GOV is coded as P-VOP, a displayed block is copied from the first field coded as I-VOP.

7.16.6.2.5 HHC decoding

If an inferior_symbol_macroblock is equal to ‘0’ then the inferior symbol means transparent and the superior symbol means opaque. Otherwise, the inferior symbol means opaque and the superior symbol means transparent. After the inferior symbol is decided, the pattern of inferior symbol in the block is decoded as the following hierarchical process.

7.16.6.2.5.1 Coded binary block pattern

The index of cbbp is obtained by decoding the variable length code to produce coded_binary_block_pattern[i] (i=0,1,2,3). If coded_binary_block_pattern[i] is one for a given block in the macroblock then the data of the block layer is included in the bitstream for that block. However, if coded_binary_block_pattern[i] is zero, then that block contains no inferior symbol for the macroblock. The binary alpha coefficients for such shape blocks shall all take the value equal to the superior symbol for the macroblock.

7.16.6.2.5.2 Binary block decoding

For the block whose coded_binary_block_pattern[I] is one, the data of the block layer is decoded. The inferior and the superior symbol for the block is indicated by inferior_symbol_block. If scan_direction is equal to 0, the block is divided into horizontal rows. If scan_direction is equal to 1, the block is divided into vertical columns. If backward_read_flag is set to ‘0’, lbp corresponds to the pixel pattern from left to right pixel in scan_direction=’0’ or from upper to lower pixel in scan_direction=’1’. If backward_read_flag is set to ‘1’, lbp corresponds to the pixel pattern from right to left pixel in scan_type=’0’ or from lower to upper pixel in scan_type=’1’.

The index of clp is obtained by decoding the variable length code to produce coded_line_pattern[i] (I=0,1,2, ,7). If coded_line_pattern[i] is one for a given line in the block then lbp information is included in the bitstream for that line. However, if coded_line_pattern[i] is zero, then that line contains no inferior symbol for the block. The binary alpha coefficient for such shape lines shall all take the value equal to the superior symbol for the block.

The index of lbp is obtained by decoding the variable length code to produce line_bit_pattern[i] (I=0,1,2, ,7). If backward_read_flag is set to ‘1’, line_bit_pattern[i] is set to line_bit_pattern[7-i]. Note that line_bit_pattern[i] equal to ‘1’ means the pixel is the inferior symbol for the block. Otherwise, line_bit_pattern[i] is equal to ‘1’ means the pixel is the superior symbol for the block .

After all pixels are decoded, the inferior symbol and superior symbol are set to either opaque or transparent symbol, respectively.

7.16.6.3 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.

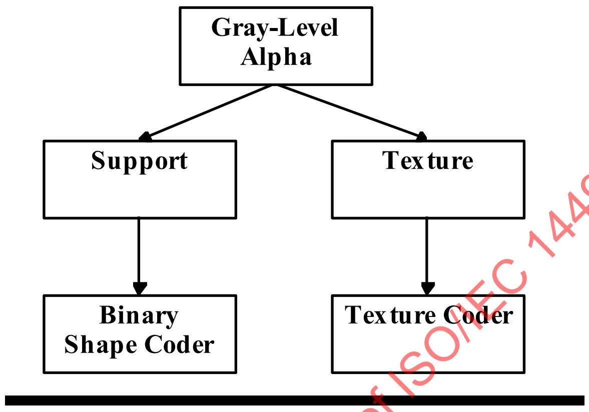


Figure AMD1-17 -- Grayscale shape coding

All samples which are indicated to be transparent by the binary shape data must be set to the value of minimum_alpha_level[i](transparent) in the decoded grayscale alpha plane i. 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 subclause 7.16.4, 7.16.5 and 7.16.7). 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.

7.16.6.3.1 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 inter 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.

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 the value of minimum_alpha_level[i] (transparent). If CODA=”all opaque” (intra, inter macroclocks) or CODA = "residual all zero" (inter macroblocks) then no more alpha data is present. Otherwise, other alpha syntax elements follow, i.e. alpha texture data encoded as either DCT coefficients which are coded and non-transparent or DPCM residuals, as is the case for regular luminance macroblock texture data.

When CODA=”all opaque”, the corresponding decoded alpha macroblock is filled with the value of maximum_alpha_level[i]. This value will be called AlphaOpaqueValue[i].

7.16.6.3.2 Intra Macroblocks coded as DPCM residuals

In case of alpha_compression_mode[i]==1(DPCM mode), the grayscale alpha data shall be decoded as DPCM residuals. DPCM mode can be selected independently of intra/inter mode of the texture data.

7.16.6.3.3 Intra Macroblocks coded as DCT coefficients

When the texture is encoded as DPCM residuals (compression_mode==1) or intra DCT coefficients (compression_mode==0 && macroblock_intra==1), and if alpha_compression_mode[i]==0, the grayscale alpha data is encoded as intra DCT residual.

The intra dc value is decoded in the same way as for luminance.

The DC predictor is used in the same way as for luminance. However, when coda_i indicates that a macroblock is all opaque, the predictor shall be reset so that the next intra block to be decoded is correctly decoded. The reset value is defined as;

AlphaOpaqueValue[i] * (2^(alpah_dct_precision[i]+alpha_intra_dc_precision[i]))

AlphaOpaqueValue[i] is described in subclaused 7.16.6.3.1.

7.16.6.3.4 Inter Macroblocks and Motion Compensation

When the texture data in a P-VOP is encoded as inter macroblock (compression_mode==0 && macroblock_intra==0), and if alpha_compression_mode[i]==0, the alpha macroblock shall be decoded as inter macroblock.

Motion compensation is carried out for inter macroblocks, using the 16x16 or 16x8 luminance motion vectors, in the same way as for luminance data. 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. 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.

cbpa is defined in Table AMD1-88 and alpha_pattern_code is derived from cbps.

Alpha inter DCT coefficients are decoded in the same way as the luminance coefficients. The only exception is that the inverse quantisation process in 7.16.4.2.3 shall perfom as if dead_zone_disable is set to zero independently of the value of dead_zone_disable.

7.16.6.3.5 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.

saturation

The alpha signals are saturated to lie in the range [Tr, Op]. Thus:

```
        { Op          if alpha > Op
alpha = { alpha       if  Tr <= alpha <= Op
        { Tr          if alpha < Tr
```

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 = ((R - (Nalpha-Tr)) * Myuv + ((Nalpha-Tr) * Nyuv)) / R
Palpha = Op

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:

```
        { Myuv                                              ..... Nalpha = Tr
Pyuv  = {
        { (Myuv - BLACK)  -  ( (Myuv - BLACK) * (Nalpha-Tr) ) / R+ Nyuv ..... Nalpha > Tr
```

(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 -Tr - ((Nalpha-Tr)*(Malpha-Tr)) / R

where

BLACK is the common black value of foreground and background objects.

Tr: level for complete transparency

Op: level for complete opacity

R: = Op - Tr (dynamic range of alpha signal)

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).

- 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.

7.16.6.4 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 AMD1-12. For example, a value '0000' indicates the grayscale (alpha) shape. The same support region as described in 7.16.6.2 is used for all auxiliary components, and the decoding procedure is the same as described in 7.16.6.3.

7.16.7 Motion compensation decoding

The motion compensation process forms predictions from previously decoded VOPs which are combined with the coefficient data (from the output of the IDCT) in order to recover the final decoded samples.

In general up to two separate predictions are formed for each block which are combined together to form the final prediction block p[y][x].

In the case of intra coded macroblocks no prediction is formed so that p[y][x] will be zero. The saturation shown in Figure AMD1-11 is still required in order to remove prohibited values from f[y][x].

In the case where a block is not coded, either because the entire macroblock is skipped or the specific block is not coded there is no coefficient data. In this case f[y][x] is zero and the decoded samples are simply the prediction, p[y][x].

All the processes except for resetting motion vector predictors in 7.16.7.4.4 are never executed when a macroblock is completely transparent.

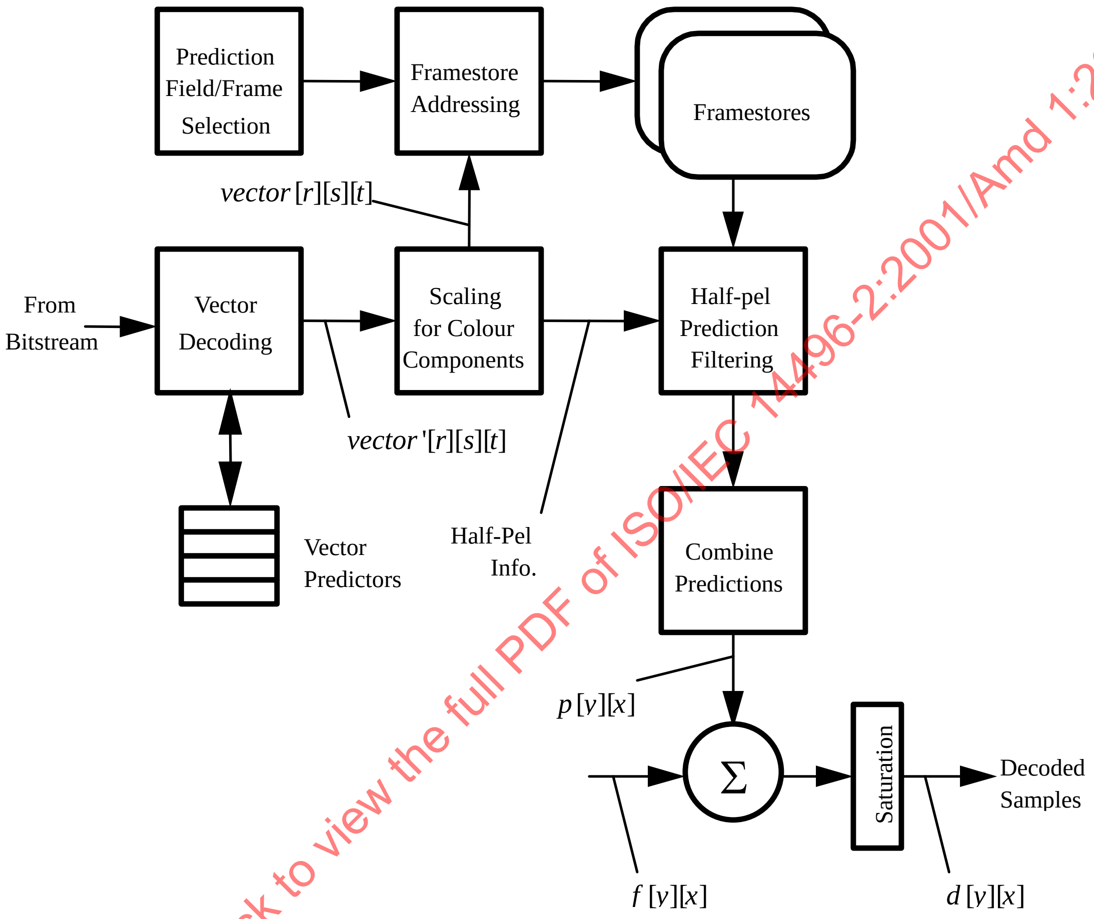


Figure AMD1-18 Simplified motion compensation process

7.16.7.1 Motion compensation decoding of arbitrary shaped VOP

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 or field) 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 7.16.4, 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.

7.16.7.1.1 Padding process

The padding process defines the values of luminance and chrominance samples outside the VOP for prediction of arbitrarily shaped objects. Figure AMD1-19 shows a simplified diagram of this process.

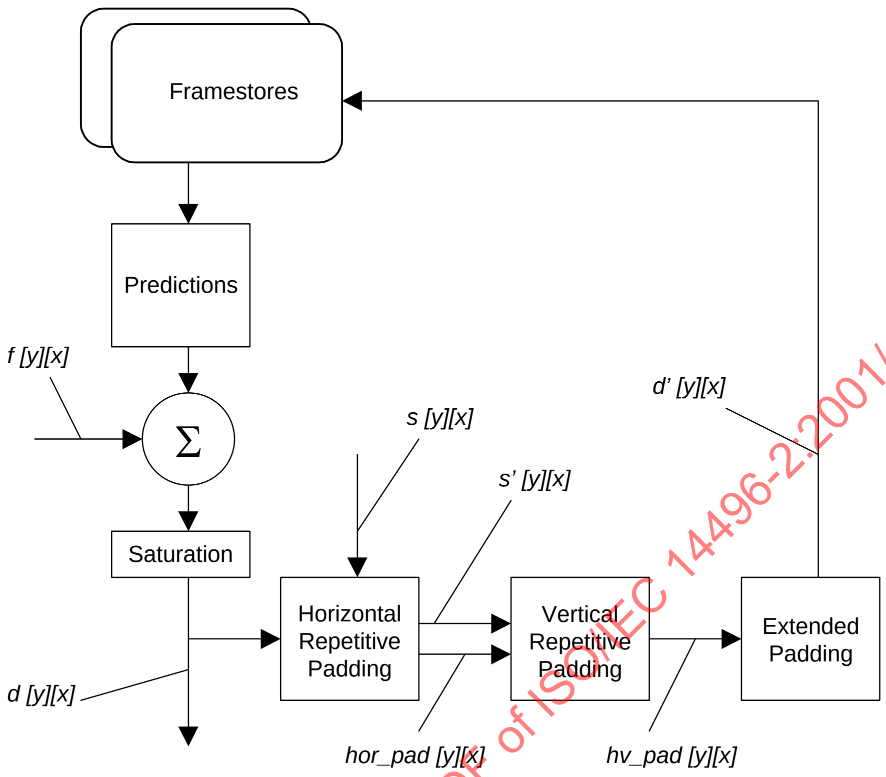


Figure AMD1-19 -- Simplified padding process

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 for 4:2:0 format, 16 x 8 samples for 4:2:2 format, 16 x 16 samples for 4:4:4 format. 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 a multiple of 16, while that of the chrominance components is defined by (vop_width>>1) and (vop_height>>1) extended to multiple of 8.

7.16.7.1.1.1 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.

7.16.7.1.1.2 Vertical repetitive padding

The remaining unfilled transparent horizontal samples (where *s'[y][x]* == 0) from subclause 7.16.7.1.1.1 are padded by a similar process as the horizontal repetitive padding but in the *vertical* direction. The samples already filled in subclause 7.16.7.1.1.1 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.

7.16.7.1.1.3 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 7.16.7.1.1.1 and subclause 7.16.7.1.1.2. If an exterior macroblock is next to more than one boundary macroblock, 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 AMD1-20. 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 $2^{\text{bits\_per\_pixel-1}}$. For 8-bit luminance component and associated chrominance this implies filling with 128.

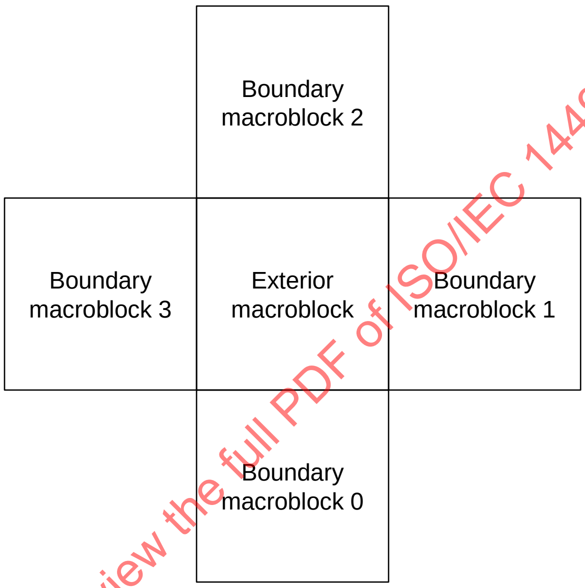


Figure AMD1-20 -- Priority of boundary macroblocks surrounding an exterior macroblock

7.16.7.1.1.4 Padding for chrominance components

Chrominance components are padded according to subclauses 7.16.7.1.1.1 through 7.16.7.1.1.3. 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.

7.16.7.1.1.5 Padding of interlaced macroblocks

Macroblocks of an interlaced VOP (progressive_sequence = 0) are padded according to 7.16.7.1.1.1 through 7.16.7.1.1.3. 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 $2^{\text{bits\_per\_pixel-1}}$. Chrominance components of interlaced VOP are padded according to subclause 7.16.7.1.1.4, however, based on fields. The padding method described in this subclause is not used outside the bounding rectangle of the VOP.

7.16.7.2 Prediction modes

There are two major classifications of the prediction mode:

- field prediction; and
- frame prediction.

In field prediction, predictions are made independently for each field by using data from one or more previously decoded fields. Frame prediction forms a prediction for the frame from one or more previously decoded frames. It must be understood that the fields and frames from which predictions are made may themselves have been decoded as either field VOPs or frame VOPs.

Within a field VOP all predictions are field predictions. However in a frame VOP either field predictions or frame predictions may be used (selected on a macroblock-by macroblock basis).

In addition to the major classification of field or frame prediction a special prediction mode is used:

• *16x8 motion compensation* - In which two motion vectors are used for each macroblock. The first motion vector is used for the upper 16x8 region, the second for the lower 16x8 region. In this specification 16x8 motion compensation shall only be used with field VOPs.

7.16.7.3 Prediction field and frame selection

The selection of which fields and frames shall be used to form predictions shall be made as detailed in this clause.

7.16.7.3.1 Field prediction

In P-VOPs, the two reference fields from which predictions shall made are the most recently decoded reference top field and the most recently decoded reference bottom field. The simplest case illustrated in Figure AMD1-21 shall be used when predicting the first VOP of a coded frame or when using field prediction within a frame-VOP. In these cases the two reference fields are part of the same reconstructed frame.

NOTES -

1 The reference fields may themselves have been reconstructed from two field-VOPs or a single frame-VOP.

2 In the case of predicting a field VOP, the field being predicted may be either the top field or the bottom field.

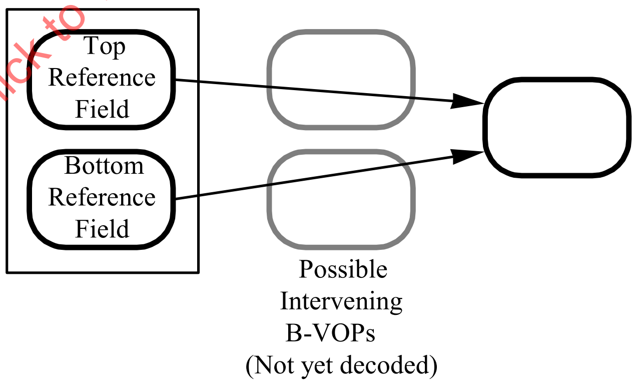


Figure AMD1-21 -- Prediction of the first field or field prediction in a frame-VOP

The case when predicting the second field VOP of a coded frame is more complicated because the two most recently decoded reference fields shall be used, and in this case, the most recent reference field was obtained from decoding the first field VOP of the coded frame. Figure AMD1-22 illustrates the situation when this second VOP is the bottom field. Figure AMD1-23 illustrates the situation when this second VOP is the top field.

NOTE - The earlier reference field may itself have been reconstructed by decoding a field VOP or a frame VOP.

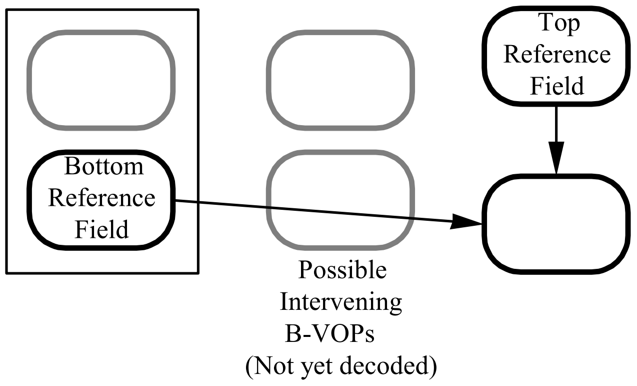


Figure AMD1-22 -- Prediction of the second field-VOP when it is the bottom field

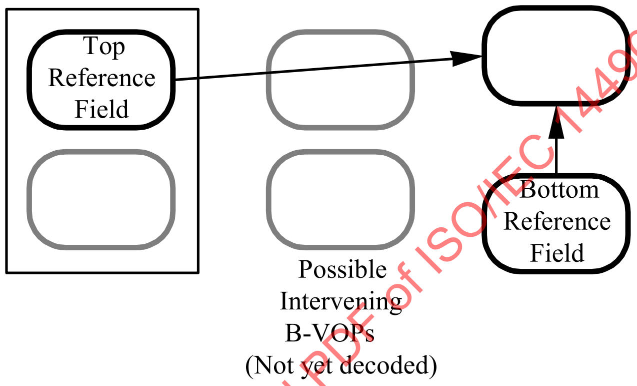


Figure AMD1-23 -- Prediction of the second field-VOP when it is the top field

7.16.7.3.2 Frame prediction

In P-VOPs prediction shall be made from the most recently reconstructed reference frame. This is illustrated in Figure AMD1-24.

NOTE - The reference frame may itself have been reconstructed from two field VOPs or a single frame VOP.

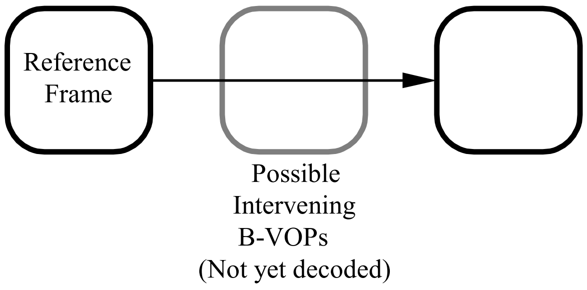


Figure AMD1-24 -- Frame-prediction for P-VOPs

7.16.7.4 Motion vectors

Motion vectors are coded differentially with respect to previously decoded motion vectors in order to reduce the number of bits required to represent them. In order to decode the motion vectors the decoder shall maintain two motion vector predictors (each with a horizontal and vertical component) denoted *PMV*[*r*][*s*][*t*]. For each prediction, a motion vector, *vector'*[*r*][*s*][*t*] is first derived. This is then scaled depending on the sampling structure (4:2:0, 4:2:2 or 4:4:4) to give a motion vector, *vector*[*r*][*s*][*t*], for each colour component. The meanings associated with the dimensions in this array are defined in Table AMD1-42.

Table AMD1-42 -- Meaning of indices in *PMV*[*r*][*s*][*t*], *vector*[*r*][*s*][*t*] and *vector'*[*r*][*s*][*t*]

	0	**1**
r	First motion vector in Macroblock	Second motion vector in Macroblock
s	Forward motion Vector	unused
t	Horizontal Component	Vertical Component

7.16.7.4.1 Decoding the motion vectors

Each motion vector component, *vector'*[*r*][*s*][*t*], shall be calculated by any process that is equivalent to the following one. Note that the motion vector predictors shall also be updated by this process.

```
r_size = vop_fcode[s][t] - 1
f = 1 << r_size
high = ( 16 * f ) -  1;
low = ( (-16) * f );
range = ( 32 * f );

if ( (f  == 1) || (motion_code[r][s][t] == 0) )
    delta = motion_code[r][s][t] ;
else {
    delta = ( ( Abs(motion_code[r][s][t]) - 1 ) * f ) + motion_residual[r][s][t] + 1;
    if (motion_code[r][s][t] < 0)
        delta = - delta;
}

prediction = PMV[r][s][t];
if ( (mv_format == "field") && (t==1) && (VOP_structure == "Frame VOP") )
    prediction = PMV[r][s][t] DIV 2;

vector'[r][s][t]= prediction + delta;
if (vector'[r][s][t] < low )
    vector'[r][s][t] = vector'[r][s][t] + range;
if (vector'[r][s][t] > high)
    vector'[r][s][t] = vector'[r][s][t] - range;

if ( (mv_format == "field") && (t==1) && (VOP_structure == "Frame VOP") )
    PMV[r][s][t] = vector'[r][s][t] * 2;
else
    PMV[r][s][t] = vector'[r][s][t];
```

The parameters in the bitstream shall be such that the reconstructed differential motion vector, *delta*, shall lie in the range [*low*:*high*]. In addition the reconstructed motion vector, *vector'*[*r*][*s*][*t*], and the updated value of the motion vector predictor *PMV*[*r*][*s*][*t*], shall also lie in the range [*low* : *high*]. The allowed range [*low* : *high*] for the motion vectors depends on the parameter vop_fcode[s][t].

r_size, *f*, *delta*, *high* , *low* and *range* are temporary variables that are not used in the remainder of this specification.

motion_code[r][s][t] and motion_residual[r][s][t] are fields recovered from the bitstream. mv_format is recovered from the bitstream using Table AMD1-30 and Table AMD1-31.

r, *s* and *t* specify the particular motion vector component being processed as identified in Table AMD1-42.

vector'[*r*][*s*][*t*] is the final reconstructed motion vector for the luminance component of the macroblock.

7.16.7.4.2 Motion vector restrictions

In frame VOPs, the vertical component of the field motion vectors shall be restricted so that they only cover half the range that is supported by the vop_fcode that relates to those motion vectors. This restriction ensures that the motion vector predictors will always have values that are appropriate for decoding subsequent frame motion vectors. Table AMD1-43 summarises the size of motion vectors that may be coded as a function of vop_fcode.

Table AMD1-43 -- Allowable motion vector range as a function of *vop_fcode*[*s*][*t*]

***vop_fcode*[*s*][*t*]**	**Vertical components (*t*==1) of field vectors in frame VOPs**	**All other cases**
0	(forbidden)	
1	[-4: +3,5]	[-8: +7,5]
2	[-8: +7,5]	[-16: +15,5]
3	[-16: +15,5]	[-32: +31,5]
4	[-32: +31,5]	[-64: +63,5]
5	[-64: +63,5]	[-128: +127,5]
6	[-128: +127,5]	[-256: +255,5]
7	[-256: +255,5]	[-512: +511,5]
8	[-512: +511,5]	[-1024: +1023,5]
9	[-1024: +1023,5]	[-2048: +2047,5]
10-14	(reserved)	
15	(used when a particular *vop_fcode*[*s*][*t*] will not be used)	

7.16.7.4.3 Updating motion vector predictors

Once all of the motion vectors present in the macroblock have been decoded using the process defined in the previous clause it is sometimes necessary to update other motion vector predictors. This is because in some prediction modes fewer than the maximum possible number of motion vectors are used. The remainder of the predictors that might be used in the VOP must retain “sensible” values in case they are subsequently used.

The motion vector predictors shall be updated as specified in Table AMD1-44 and Table AMD1-45. The rules for updating motion vector predictors in the case of skipped macroblocks are specified in 7.16.5.7.

NOTE - It is possible for an implementation to optimise the updating (and resetting) of motion vector predictors depending on the VOP type. For example in a P-VOP the predictors for backwards motion vectors are unused and need not be maintained.

Table AMD1-44 -- Updating of motion vector predictors in frame VOPs

frame_motion_-type	macroblock_motion_-forward	macroblock_-intra	Predictors to Update
Frame-based‡	-	1	*PMV*[1][0][1:0] = *PMV*[0][0][1:0]
Frame-based	1	0	*PMV*[1][0][1:0] = *PMV*[0][0][1:0]
Frame-based‡	0	0	*PMV*[*r*][*s*][*t*] = 0 §
Field-based	1	0	(none)

NOTE - *PMV*[*r*][*s*][1:0] = *PMV*[*u*][*v*][1:0] means that;
PMV[*r*][*s*][1] = *PMV*[*u*][*v*][1] and *PMV*[*r*][*s*][0] = *PMV*[*u*][*v*][0]

‡ **frame_motion_type** is not present in the bitstream but is assumed to be Frame-based

§ (Only occurs in P-VOP) *PMV*[*r*][*s*][*t*] is set to zero (for all *r*, *s* and *t*). See 7.16.7.4.4.

Table AMD1-45 -- Updating of motion vector predictors in field VOPs

field_motion_-type	macroblock_motion_-forward	macroblock_-intra	Predictors to Update
Field-based‡	-	1	*PMV*[1][0][1:0] = *PMV*[0][0][1:0]
Field-based	1	0	*PMV*[1][0][1:0] = *PMV*[0][0][1:0]
Field-based‡	0	0	*PMV*[*r*][*s*][*t*] = 0 §
16x8 MC	1	0	(none)

NOTE - *PMV*[*r*][*s*][1:0] = *PMV*[*u*][*v*][1:0] means that;
PMV[*r*][*s*][1] = *PMV*[*u*][*v*][1] and *PMV*[*r*][*s*][0] = *PMV*[*u*][*v*][0]

‡ **field_motion_type** is not present in the bitstream but is assumed to be Field-based

§ (Only occurs in P-VOP) *PMV*[*r*][*s*][*t*] is set to zero (for all *r*, *s* and *t*). See 7.16.7.4.4.

7.16.7.4.4 Resetting motion vector predictors

All motion vector predictors shall be reset to zero in the following cases:

- At the start of each slice.
- Whenever a macroblock is completely transparent.
- Whenever an intra macroblock is decoded.
- Whenever a macroblock is encoded as DPCM residuals.
- In a P-VOP when a macroblock is skipped (not coded = 1).

7.16.7.4.5 Prediction in P-VOP

In the case that a P field VOP is used as the second field of a frame in which the first field is an I field VOP a series of semantic restrictions apply. These ensure that prediction is only made from the I field VOP. These restrictions are;

- Field prediction in which **motion_vertical_field_select** indicates the same parity as the field being predicted shall not be used.
- There shall be no skipped macroblocks (not_coded = 0).

7.16.7.4.6 Motion vectors for chrominance components

The motion vectors calculated in the previous clauses refer to the luminance component where;

vector[*r*][*s*][*t*] = *vector'*[*r*][*s*][*t*] (for all r, s and t)

For each of the two chrominance components the motion vectors shall be scaled as follows:

4:2:0 Both the horizontal and vertical components of the motion vector are scaled by dividing by two:

vector[*r*][*s*][0] = *vector'*[*r*][*s*][0] / 2;

vector[*r*][*s*][1] = *vector'*[*r*][*s*][1] / 2;

4:2:2 The horizontal component of the motion vector is scaled by dividing by two, the vertical component is not altered:

vector[*r*][s][0] = *vector'*[*r*][s][0] / 2;

vector[*r*][s][1] = *vector'*[r][s][1];

4:4:4 The motion vector is unmodified:

vector[*r*][s][0] = *vector'*[*r*][s][0];

vector[*r*][s][1] = *vector'*[*r*][s][1];

7.16.7.4.7 Semantic restrictions concerning predictions

It is a restriction on the bitstream that reconstructed motion vectors shall not refer to samples outside the decoded area of a reference VOP. For an arbitrary shape VOP, the decoded area refers to the area within the bounding rectangle, padded as described in subclause 7.16.7.1.1. A bounding rectangle is defined by vop_width and vop_height extended to multiple of 16.

7.16.7.5 Forming predictions

Predictions are formed by reading prediction samples from the reference fields or frames. A given sample is predicted by reading the corresponding sample in the reference field or frame offset by the motion vector in the absolute coordinate system.

A positive value of the horizontal component of a motion vector indicates that the prediction is made from samples (in the reference field/frame) that lie to the right of the samples being predicted in the absolute coordinate system.

A positive value of the vertical component of a motion vector indicates that the prediction is made from samples (in the reference field/frame) that lie the below the samples being predicted in the absolute coordinate system.

All motion vectors are specified to an accuracy of one half sample. Thus if a component of the motion vector is odd, the samples will be read from mid-way between the actual samples in the reference field/frame. These half-samples are calculated by simple linear interpolation from the actual samples.

In the case of field-based predictions it is necessary to determine which of the two available fields to use to form the prediction. In the case of field-based prediction and 16x8 MC an additional bit, motion_vertical_field_select, is encoded to indicate which field to use.

If motion_vertical_field_select is zero then the prediction is taken from the top reference field.

If motion_vertical_field_select is one then the prediction is taken from the bottom reference field.

For each prediction block the integer sample motion vectors int_vec[t] and the half sample flags half_flag[t] shall be formed as follows;

```
for (t=0 ; t<2 ; t++) {
    int_vec[t] = vector[r][s][t] DIV 2 ;
    if ((vector[r][s][t] – (2 * int_vec[t]) != 0)
        half_flag[t] = 1 ;
    else
        half_flag[t] = 0 ;
}
```

Then for each sample in the prediction block the samples are read and the half sample prediction applied as follows;

```
if ( ( ! half_flag[0] )&& ( ! half_flag[1]) )
    pel_pred[y][x] = pel_ref[y + int_vec[1]][x + int_vec[0]] ;

if ( ( ! half_flag[0] )&& half_flag[1] )
    pel_pred[y][x] = ( pel_ref[y + int_vec[1]][x + int_vec[0]] +
                       pel_ref[y + int_vec[1]+1][x + int_vec[0]] ) // 2 ;

if ( half_flag[0]&& ( ! half_flag[1]) )
    pel_pred[y][x] = ( pel_ref[y + int_vec[1]][x + int_vec[0]] +
                       pel_ref[y + int_vec[1]][x + int_vec[0]+1] ) // 2 ;

if ( half_flag[0]&& half_flag[1] )
    pel_pred[y][x] = ( pel_ref[y + int_vec[1]][x + int_vec[0]] +
                       pel_ref[y + int_vec[1]][x + int_vec[0]+1] +
                         pel_ref[y + int_vec[1]+1][x + int_vec[0]] +
                           pel_ref[y + int_vec[1]+1][x + int_vec[0]+1] ) // 4 ;
```

where *pel_pred*[*y*][*x*] is the prediction sample being formed and *pel_ref*[*y*][*x*] are samples in the reference field or frame.

7.16.7.6 Motion vector selection

Table AMD1-46 shows the prediction modes used in field VOPs and Table AMD1-47 shows the predictions used in frame VOPs. In each table the motion vectors that are present in the bitstream are listed in the order in which they appear in the bitstream.

Table AMD1-46 -- Predictions and motion vectors in field VOPs

field_ motion_ type	**macroblock_motion_-forward**	**macro-block_-intra**	**Motion vector**	**Prediction formed for**
Field-based‡	-	1	*vector*'[0][0][1:0]	None
Field-based	1	0	*vector*'[0][0][1:0]	whole field, forward
Field-based‡	0	0	*vector*'[0][0][1:0]*§	whole field, forward
16x8 MC	1	0	*vector*'[0][0][1:0] *vector*'[1][0][1:0]	upper 16x8 field, forward lower 16x8 field, forward
NOTE - Motion vectors are listed in the order they appear in the bitstream ‡ **field_motion_type** is not present in the bitstream but is assumed to be Field-based * These motion vectors are not present in the bitstream § The motion vector is taken to be (0; 0) as explained in 7.16.7.4.5.				

Table AMD1-47 -- Predictions and motion vectors in frame VOPs

frame_-motion_-type	**macroblock_motion_-forward**	**macro-block_-intra**	**Motion vector**	**Prediction formed for**
Frame-based‡	-	1	*vector*'[0][0][1:0]	None
Frame-based	1	0	*vector*'[0][0][1:0]	frame, forward
Frame-based‡	0	0	*vector*'[0][0][1:0]*§	frame, forward
Field-based	1	0	*vector*'[0][0][1:0] *vector*'[1][0][1:0]	top field, forward bottom field, forward
NOTE - Motion vectors are listed in the order they appear in the bitstream ‡ **frame_motion_type** is not present in the bitstream but is assumed to be Frame-based * These motion vectors are not present in the bitstream § The motion vector is taken to be (0; 0) as explained in 7.16.7.4.5.				

7.16.7.7 Skipped macroblocks

A skipped macroblock is a macroblock for which no data is encoded, that is part of a coded slice. With the exception of the first non-transparent macroblock in a slice, if not_coded flag in a macroblock is '1', the macroblock is to be skipped. The decoder shall form a prediction for skipped macroblocks which shall then be used as the final decoded sample values.

The process differs between field VOPs and frame VOPs.

There shall be no skipped macroblocks in I-VOPs.

7.16.7.7.1 P field VOP

- The prediction shall be made as if field_motion_type is “Field-based”
- The prediction shall be made from the field of the same parity as the field being predicted.
- Motion vector predictors shall be reset to zero.
- The motion vector shall be zero.

7.16.7.7.2 P frame VOP

- The prediction shall be made as if frame_motion_type is “Frame-based”
- Motion vector predictors shall be reset to zero.
- The motion vector shall be zero.

7.16.7.8 Combining predictions

The final stage is to combine the various predictions together in order to form the final prediction blocks.

It is also necessary to organise the data into blocks that are either field organised or frame organised in order to be added directly to the decoded coefficients.

The transform data is either field organised or frame organised as specified by *dct_type*.

7.16.7.8.1 Simple frame predictions

In the case of simple frame predictions no further processing is required.

The predictions for chrominance components of 4:2:0, 4:2:2 and 4:4:4 formats shall be of size 8 samples by 8 lines, 8 samples by 16 lines and 16 samples by 16 lines respectively.

7.16.7.8.2 Simple field predictions

In the case of simple field predictions (i.e. not 16×8) no further processing is required

In the case of simple field prediction in a frame picture the predictions for chrominance components of 4:2:0, 4:2:2 and 4:4:4 formats for each field shall be of size 8 samples by 4 lines, 8 samples by 8 lines and 16 samples by 8 lines respectively.

7.16.7.8.3 16x8 Motion compensation

In this prediction mode separate predictions are formed for the upper 16x8 region of the macroblock and the lower 16x8 region of the macroblock.

The predictions for chrominance components, for each 16x8 region, of 4:2:0, 4:2:2 and 4:4:4 formats shall be of size 8 samples by 4 lines, 8 samples by 8 lines and 16 samples by 8 lines respectively.

7.16.8 Output of the decoding process

This section describes the output of the theoretical model of the decoding process that decodes bitstreams conforming to this specification.

The decoding process input is one or more coded video bitstreams (one for each of the layers). The video layers are generally multiplexed by the means of a system stream that also contains timing information.

The output of the decoding process is a series of fields or frames that are normally the input of a display process. The order in which fields or frames are output by the decoding process is called the display order, and may be different from the coded order (when B-pictures are used). The display process is responsible for the action of displaying the decoded fields or frames on a display device. If the display device cannot display at the frame rate indicated in the bitstream, the display process may perform frame rate conversion. This specification does not describe a theoretical model of the display process nor the operation of the display process.

Since some of the syntax elements, such as progressive_frame, may be needed by the display process, in this theoretical model of the decoding process, all the syntactic elements that are decoded by the decoding process are output by the decoding process and may be accessed by the display process.

When a progressive sequence is decoded (progressive_sequence is equal to 1), the luminance and chrominance samples of the reconstructed frames are output by decoding process in the form of progressive frames and the output rate is the frame rate. Figure AMD1-25 illustrates this in the case of chroma_format equals to 4:2:0.

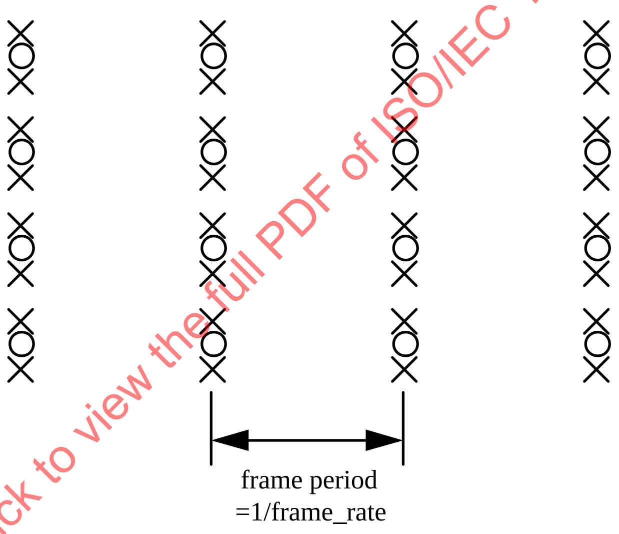


Figure AMD1-25. progressive_sequence == 1

The same reconstructed frame is output one time if repeat_first_field is equal to 0, and two or three consecutive times if repeat_first_field is equal to 1, depending on the value of top_field_first. Figure AMD1-26 illustrates this in the case of chroma_format equals to 4:2:0 and repeat_first_field equals 1.

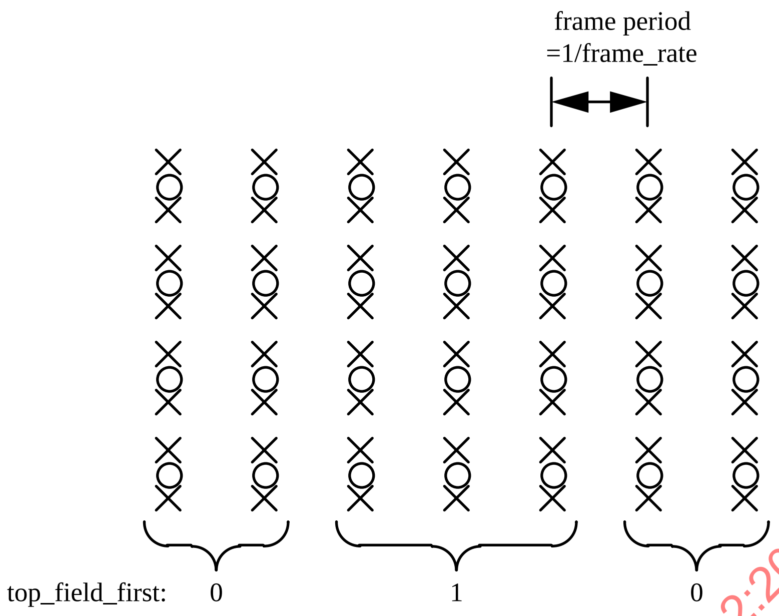


Figure AMD1-26. progressive_sequence == 1, repeat_first_field = 1

When decoding an interlaced sequence (progressive_sequence is equal to 0), the luminance samples of the reconstructed frames are output by the decoding process in the form of interlaced fields at a rate that is twice the frame rate. Figure AMD1-27 illustrates this.

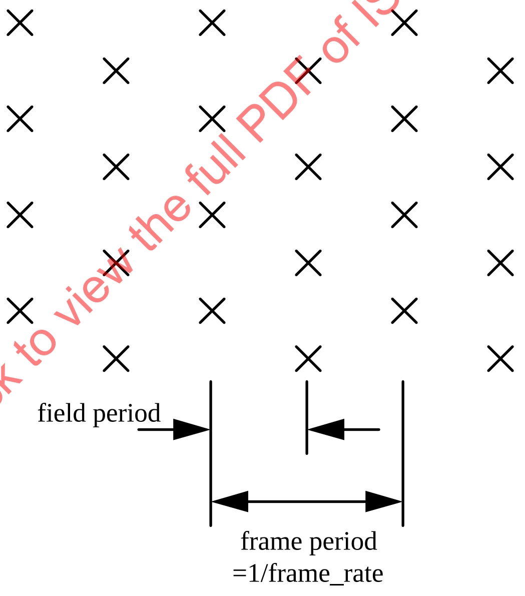


Figure AMD1-27. progressive_sequence == 0

It is a requirement on the bitstream that the fields at the output of the decoding process shall always be alternately top and bottom (note that the very first field of a sequence may be either top or bottom).

If the reconstructed frame is interlaced (progressive_frame is equal to 0), the luminance samples and chrominance samples are output by the decoding process in the form of two consecutive fields. The first field output by the decoding process is the top field or the bottom field of the reconstructed frame, depending on the value of top_field_first.

Although all the samples of progressive frames represent the same instant in time, all the samples are not output at the same time by the decoding process when the sequence is interlaced.

If the reconstructed frame is progressive (progressive_frame is equal to 1), the luminance samples are output by the decoding process in the form of two or three consecutive fields, depending on the value of repeat_first_field.

NOTE - The information that these fields originate from the same progressive frame in the bitstream is conveyed to the display process.

All of the chrominance samples of the reconstructed progressive frame are output by the decoding process at the same time as the first field of luminance samples. This is illustrated in Figures AMD1-28 and AMD1-29.

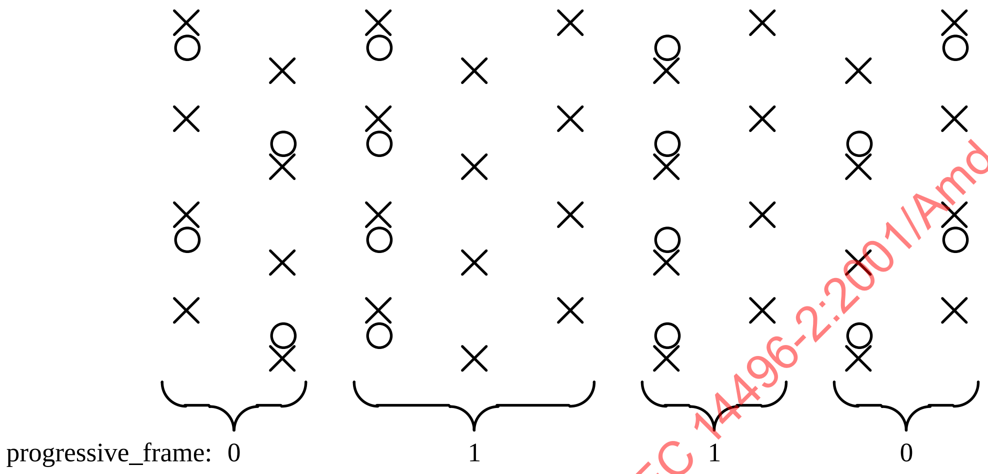


Figure AMD1-28. progressive_sequence == 0 with 4:2:0 chrominance.

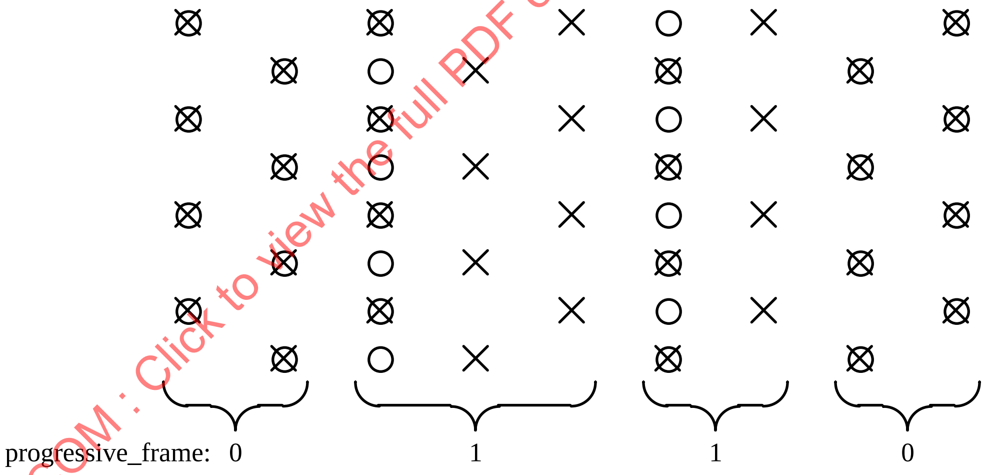


Figure AMD1-29. progressive_sequence == 0 with 4:2:2 or 4:4:4 chrominance.

7.16.9 Sprite decoding

The subclause specifies the additional decoding process for a sprite video object. The sprite decoding can operate in two modes: basic sprite decoding and low-latency sprite decoding. Figure AMD1-30 is a diagram of the sprite decoding process. It is simplified for clarity.

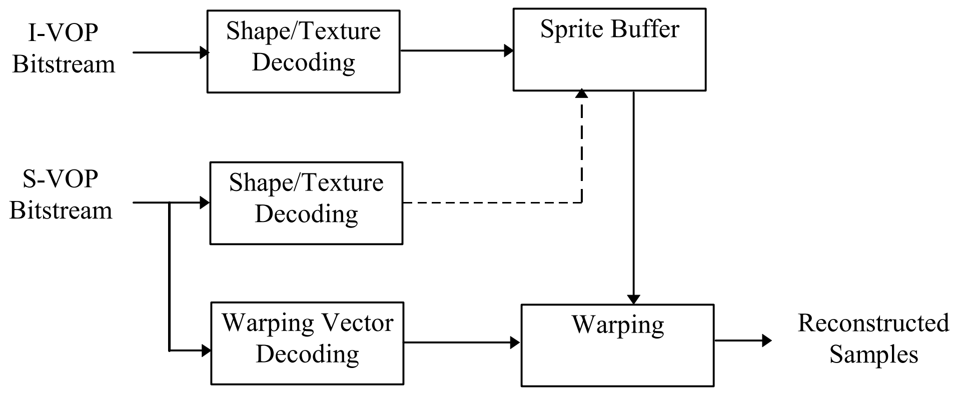


Figure AMD1-30 -- The sprite decoding process

7.16.9.1 Higher syntactic structures

The various parameters in the VOL and VOP bitstreams shall be interpreted as described in clause 6. When sprite_enable == '1', vop_coding_type shall be "I" only for the initial VOP in a VOL for basic sprites (i.e. low_latency_sprite_enable == '0'), and all the other VOPs shall be S-VOPs (i.e. vop_coding_type == "S"). The reconstructed I-VOP in a VOL for basic sprites is not displayed but stored in a sprite memory, and will be used by all the remaining S-VOPs in the same VOL. An S-VOP is reconstructed by applying warping to the VOP stored in the sprite memory, using the warping parameters (i.e. a set of motion vectors) embedded in the VOP bitstream. Alternatively, in a VOL for low-latency sprites (i.e. low_latency_sprite_enable == '1'), these S-VOPs can update the information stored in the sprite memory before applying warping.

7.16.9.2 Sprite Reconstruction

The luminance, chrominance and grayscale alpha data of a sprite are stored in two-dimensional arrays. The width and height of the luminance array are specified by sprite_width and sprite_height respectively. The samples in the sprite luminance, chrominance and grayscale alpha arrays are addressed by two-dimensional integer pairs (i', j') and (i_c', j_c') as defined in the following:

- Top left luminance and grayscale alpha sample
  (i', j') = (sprite_left_coordinate, sprite_top_coordinate)
- Bottom right luminance and grayscale alpha sample
  (i', j') = (sprite_left_coordinate + sprite_width – 1, sprite_top_coordinate + sprite_height – 1)
- Top left chrominance sample
  For 4:2:0 VOPs,
  (i_c', j_c') = (sprite_left_coordinate / 2, sprite_top_coordinate / 2)
  For 4:2:2 VOPs,
  (i_c', j_c') = (sprite_left_coordinate / 2, sprite_top_coordinate)
  For 4:4:4 VOPs,
  (i_c', j_c') = (sprite_left_coordinate, sprite_top_coordinate)
- Bottom right chrominance sample
  For 4:2:0 VOPs,
  (i_c', j_c') = (sprite_left_coordinate / 2 + sprite_width// 2 – 1, sprite_top_coordinate / 2 + sprite_height// 2 – 1).
  For 4:2:2 VOPs,
  (i_c', j_c') = (sprite_left_coordinate / 2 + sprite_width// 2 – 1, sprite_top_coordinate + sprite_height – 1).
  For 4:4:4 VOPs,
  (i_c', j_c') = (sprite_left_coordinate + sprite_width – 1, sprite_top_coordinate + sprite_height – 1).

- Likewise, the addresses of the luminance, chrominance and grayscale alpha samples of the VOP currently being decoded are defined in the following:

- Top left sample of luminance and grayscale alpha
  (i, j) = (0, 0) for rectangular VOPs, and
  (i, j) = (vop_horizontal_mc_spatial_ref, vop_vertical_mc_spatial_ref) for non-rectangular VOPs

- Bottom right sample of luminance and grayscale alpha
  (i, j) = (video_object_layer_width - 1, video_object_layer_height - 1) for rectangular VOPs, and
  (i, j) = (vop_horizontal_mc_spatial_ref + vop_width - 1, vop_vertical_mc_spatial_ref + vop_height - 1) for non-rectangular VOPs

- Top left sample of chrominance
  (i_c, j_c) = (0, 0) for rectangular VOPs, and
  (i_c, j_c) = (vop_horizontal_mc_spatial_ref / 2, vop_vertical_mc_spatial_ref / 2) for non-rectangular 4:2:0 VOPs
  (i_c, j_c) = (vop_horizontal_mc_spatial_ref / 2, vop_vertical_mc_spatial_ref) for non-rectangular 4:2:2 VOPs
  (i_c, j_c) = (vop_horizontal_mc_spatial_ref, vop_vertical_mc_spatial_ref) for non-rectangular 4:4:4 VOPs

- Bottom right sample of chrominance
  (i_c, j_c) = (video_object_layer_width / 2 - 1, video_object_layer_height / 2 - 1) for rectangular 4:2:0 VOPs, and
  (i_c, j_c) = (vop_horizontal_mc_spatial_ref / 2 + vop_width// 2 - 1, vop_vertical_mc_spatial_ref / 2 + vop_height// 2 - 1) for non-rectangular 4:2:0 VOPs
  (i_c, j_c) = (video_object_layer_width / 2 - 1, video_object_layer_height - 1) for rectangular 4:2:2 VOPs, and
  (i_c, j_c) = (vop_horizontal_mc_spatial_ref / 2 + vop_width// 2 - 1, vop_vertical_mc_spatial_ref + vop_height - 1) for non-rectangular 4:2:2 VOPs
  (i_c, j_c) = (video_object_layer_width - 1, video_object_layer_height - 1) for rectangular 4:4:4 VOPs, and
  (i_c, j_c) = (vop_horizontal_mc_spatial_ref + vop_width- 1, vop_vertical_mc_spatial_ref + vop_height - 1) for non-rectangular 4:4:4 VOPs

7.16.9.3 Sprite reference point decoding

The syntactic elements in sprite_trajectory () and below shall be interpreted as specified in clause 6. du[i] and dv[i] (0 =< i < no_sprite_point) specifies the mapping between indexes of some reference points in the VOP and the corresponding reference points in the sprite. These points are referred to as VOP reference points and sprite reference points respectively in the rest of the specification.
The index values for the VOP reference points are defined as:

(i_0, j_0) = (0, 0) when video_object_layer_shape == 'rectangle', and
(vop_horizontal_mc_spatial_ref, vop_vetical_mc_spatial_ref) otherwise,
$(i_1, j_1) = (i_0+W, j_0)$,
$(i_2, j_2) = (i_0, j_0 + H)$,
$(i_3, j_3) = (i_0+W, j_0+H)$

where W = video_object_layer_width and H = video_object_layer_height when video_object_layer_shape == 'rectangle' or W = vop_width and H = vop_height otherwise. Only the index values with subscripts less than no_sprite_point shall be used for the rest of the decoding process.

The index values for the sprite reference points shall be calculated as follows:

$(i_0', j_0') = (s / 2) (2\, i_0 + du[0], 2\, j_0 + dv[0])$

$(i_1', j_1') = (s / 2) (2\, i_1 + du[1] + du[0], 2\, j_1 + dv[1] + dv[0])$

$(i_2', j_2') = (s / 2) (2\, i_2 + du[2] + du[0], 2\, j_2 + dv[2] + dv[0])$

$(i_3', j_3') = (s / 2) (2\, i_3 + du[3] + du[2] + du[1] + du[0], 2\, j_3 + dv[3] + dv[2] + dv[1] + dv[0])$

where i_0', j_0', etc are integers in $\frac{1}{s}$ pel accuracy, where s is specified by sprite_warping_accuracy. Only the index values with substcripts less than no_sprite_point need to be calculated.

When no_of_sprite_warping_points == 2 or 3, the index values for the *virtual sprite points* are additionally calculated as follows:

$$(i_1'', j_1'') = (16\,(i_0 + W') + ((W - W')\,(r\, i_0' - 16\, i_0) + W'\,(r\, i_1' - 16\, i_1))\,//\,W,$$
$$16\, j_0 + ((W - W')\,(r\, j_0' - 16\, j_0) + W'\,(r\, j_1' - 16\, j_1))\,//\,W)$$
$$(i_2'', j_2'') = (16\, i_0 + ((H - H')\,(r\, i_0' - 16\, i_0) + H'\,(r\, i_2' - 16\, i_2))\,//\,H,$$
$$16\,(j_0 + H') + ((H - H')\,(r\, j_0' - 16\, j_0) + H'\,(r\, j_2' - 16\, j_2))\,//\,H)$$

where i_1'', j_1'', i_2'', and j_2'' are integers in $\frac{1}{16}$ pel accuracy, and $r = 16/s$. W' and H' are defined as the smallest integers that satisfy the following condition:

$W' = 2^{\alpha}$, $H' = 2^{\beta}$, $W' \geq W$, $H' \geq H$, $\alpha > 0$, $\beta > 0$, both α and β are integers.

The calculation of i_2'', and j_2'' is not necessary when no_of_sprite_warping_points == 2.

7.16.9.4 Warping

For any pixel (i, j) inside the VOP boundary, $(F(i, j), G(i, j))$ and $(F_c(i_c, j_c), G_c(i_c, j_c))$ are computed as described as follows. These quantities are then used for sample reconstruction as specified in subclause 7.16.9.5. The following notations are used to simplify the description:

$I = i - i_0$,
$J = j - j_0$,

For 4:2:0 VOPs,

$I_c = 4\, i_c - 2\, i_0 + 1$,

$J_c = 4\, j_c - 2\, j_0 + 1$,

For 4:2:2 VOPs,

$I_c = 4\, i_c - 2\, i_0 + 1$, for calculating $F_c(i_c, j_c)$,

$I_c = 2\, i_c - i_0$, for calculating $G_c(i_c, j_c)$,

$J_c = 2\, j_c - 2\, j_0 + 1$, for calculating $F_c(i_c, j_c)$,

$J_c = j_c - j_0$, for calculating $G_c(i_c, j_c)$,

For 4:4:4 VOPs,

$I_c = i_c - i_0$,

$J_c = j_c - j_0$,

When no_of_sprite_warping_point == 0,

$$(F(i, j), G(i, j)) = (si, sj),$$
$$(F_c(i_c, j_c), G_c(i_c, j_c)) = (si_c, sj_c).$$

When no_of_sprite_warping_point == 1,

$$(F(i, j), G(i, j)) = (i_0' + sI, j_0' + sJ),$$
$$(F_c(i_c, j_c), G_c(i_c, j_c)) = (i_0' /// 2 + s\,(i_c - i_0 / 2),\ j_0' /// 2 + s\,(j_c - j_0 / 2)),\ \text{for 4:2:0 VOPs.}$$

$$(F_c(i_c, j_c), G_c(i_c, j_c)) = (i_0' /// 2 + s\,(i_c - i_0 / 2), j_0' + sJ_c),\ \text{for 4:2:2 VOPs.}$$

$$(F_c(i_c, j_c), G_c(i_c, j_c)) = (i_0' + sI_c, j_0' + sJ_c),\ \text{for 4:4:4 VOPs.}$$

When no_of_sprite_warping_points == 2,

$$\begin{aligned}(F(i, j), G(i, j)) &= (\,i_0' + ((-r\,i_0' + i_1'')\,I + (r\,j_0' - j_1'')\,J) /// (W'\,r)\,,\\ &\quad j_0' + ((-r\,j_0' + j_1'')\,I + (-r\,i_0' + i_1'')\,J) /// (W'\,r)),\\ (F_c(i_c, j_c), G_c(i_c, j_c)) &= (((-r\,i_0' + i_1'')\,I_c + (r\,j_0' - j_1'')\,J_c + 2\,W'\,r\,i_0' - 16W') /// (4\,W'\,r),\\ &\quad ((-r\,j_0' + j_1'')\,I_c + (-r\,i_0' + i_1'')\,J_c + 2\,W'\,r\,j_0' - 16W') /// (4\,W'\,r)),\end{aligned}$$

for 4:2:0 VOPs,

$$\begin{aligned}(F_c(i_c, j_c), G_c(i_c, j_c)) &= (((-r\,i_0' + i_1'')\,I_c + (r\,j_0' - j_1'')\,J_c + 2\,W'\,r\,i_0' - 16W') /// (4\,W'\,r),\\ &\quad j_0' + ((-r\,j_0' + j_1'')\,I_c + (-r\,i_0' + i_1'')\,J_c) /// (W'\,r)),\end{aligned}$$

for 4:2:2 VOPs,

$$\begin{aligned}(F_c(i_c, j_c), G_c(i_c, j_c)) &= (\,i_0' + ((-r\,i_0' + i_1'')\,I_c + (r\,j_0' - j_1'')\,J_c) /// (W'\,r)\,,\\ &\quad j_0' + ((-r\,j_0' + j_1'')\,I_c + (-r\,i_0' + i_1'')\,J_c) /// (W'\,r)),\end{aligned}$$

for 4:4:4 VOPs.

According to the definition of W' and H' (i.e. $W' = 2^{\alpha}$ and $H' = 2^{\beta}$), the divisions by "///" in these functions can be replaced by binary shift operations. By this replacement, the above equations can be rewritten as:

$$\begin{aligned}(F(i, j), G(i, j)) &= (\,i_0' + (((-r\,i_0' + i_1'')\,I + (r\,j_0' - j_1'')\,J + 2^{\alpha+\rho-1}) >> (\alpha+\rho))\,,\\ &\quad j_0' + (((-r\,j_0' + j_1'')\,I + (-r\,i_0' + i_1'')\,J + 2^{\alpha+\rho-1}) >> (\alpha+\rho)),\\ (F_c(i_c, j_c), G_c(i_c, j_c)) &= (((-r\,i_0' + i_1'')\,I_c + (r\,j_0' - j_1'')\,J_c + 2\,W'\,r\,i_0' - 16W' + 2^{\alpha+\rho+1}) >> (\alpha+\rho+2),\\ &\quad ((-r\,j_0' + j_1'')\,I_c + (-r\,i_0' + i_1'')\,J_c + 2\,W'\,r\,j_0' - 16W' + 2^{\alpha+\rho+1}) >> (\alpha+\rho+2)),\end{aligned}$$

for 4:2:0 VOPs,

$$\begin{aligned}(F_c(i_c, j_c), G_c(i_c, j_c)) &= (((-r\,i_0' + i_1'')\,I_c + (r\,j_0' - j_1'')\,J_c + 2\,W'\,r\,i_0' - 16W' + 2^{\alpha+\rho+1}) >> (\alpha+\rho+2),\\ &\quad j_0' + (((-r\,j_0' + j_1'')\,I_c + (-r\,i_0' + i_1'')\,J_c + 2^{\alpha+\rho-1})),\end{aligned}$$

for 4:2:2 VOPs,

$$\begin{aligned}(F_c(i_c, j_c), G_c(i_c, j_c)) &= (\,i_0' + (((-r\,i_0' + i_1'')\,I_c + (r\,j_0' - j_1'')\,J_c + 2^{\alpha+\rho-1}) >> (\alpha+\rho))\,,\\ &\quad j_0' + (((-r\,j_0' + j_1'')\,I_c + (-r\,i_0' + i_1'')\,J_c + 2^{\alpha+\rho-1}) >> (\alpha+\rho)),\end{aligned}$$

for 4:4:4 VOPs.

where $2^{\rho} = r$.

When no_of_sprite_warping_points == 3,

$$(F(i, j), G(i, j)) = (i_0' + ((-r\,i_0' + i_1'')\,H'\,I + (-r\,i_0' + i_2'')W'\,J) \mathbin{///} (W'H'r),\; j_0' + ((-r\,j_0' + j_1'')\,H'\,I + (-r\,j_0' + j_2'')W'\,J) \mathbin{///} (W'H'r)),$$

$$(F_c(i_c, j_c), G_c(i_c, j_c)) = (((-r\,i_0' + i_1'')\,H'\,I_c + (-r\,i_0' + i_2'')W'\,J_c + 2\,W'H'r\,i_0' - 16W'H') \mathbin{///} (4W'H'r),\; ((-r\,j_0' + j_1'')\,H'\,I_c + (-r\,j_0' + j_2'')W'\,J_c + 2\,W'H'r\,j_0' - 16W'H') \mathbin{///} (4W'H'r)),$$

for 4:2:0 VOPs,

$$(F_c(i_c, j_c), G_c(i_c, j_c)) = (((-r\,i_0' + i_1'')\,H'\,I_c + (-r\,i_0' + i_2'')W'\,J_c + 2\,W'H'r\,i_0' - 16W'H') \mathbin{///} (4W'H'r),\; j_0' + ((-r\,j_0' + j_1'')\,H'\,I_c + (-r\,j_0' + j_2'')W'\,J_c) \mathbin{///} (W'H'r))$$

for 4:2:2 VOPs,

$$(F_c(i_c, j_c), G_c(i_c, j_c)) = (i_0' + ((-r\,i_0' + i_1'')\,H'\,I_c + (-r\,i_0' + i_2'')W'\,J_c) \mathbin{///} (W'H'r),\; j_0' + ((-r\,j_0' + j_1'')\,H'\,I_c + (-r\,j_0' + j_2'')W'\,J_c) \mathbin{///} (W'H'r)),$$

for 4:4:4 VOPs.

According to the definition of *W'* and *H'*, the computation of these functions can be simplified by dividing the denominator and numerator of the division beforehand by *W'* (when *W'* < *H'*) or *H'* (when *W'* ≥ *H'*). As in the case of no_of_sprite_warping_points == 2, the divisions by "///" in these functions can be replaced by binary shift operations. For example, when *W'* ≥ *H'* (i.e. $\alpha \geq \beta$) the above equations can be rewritten as:

$$(F(i, j), G(i, j)) = (i_0' + (((-r\,i_0' + i_1'')\,I + (-r\,i_0' + i_2'')\,2^{\alpha-\beta}\,J + 2^{\alpha+\rho-1}) >> (\alpha+\rho)),\; j_0' + (((-r\,j_0' + j_1'')\,I + (-r\,j_0' + j_2'')\,2^{\alpha-\beta}\,J + 2^{\alpha+\rho-1}) >> (\alpha+\rho))),$$

$$(F_c(i_c, j_c), G_c(i_c, j_c)) = (((-r\,i_0' + i_1'')\,I_c + (-r\,i_0' + i_2'')\,2^{\alpha-\beta}\,J_c + 2W'r\,i_0' - 16W' + 2^{\alpha+\rho+1}) >> (\alpha+\rho+2),\; ((-r\,j_0' + j_1'')\,I_c + (-r\,j_0' + j_2'')\,2^{\alpha-\beta}\,J_c + 2W'r\,j_0' - 16W' + 2^{\alpha+\rho+1}) >> (\alpha+\rho+2)),$$

for 4:2:0 VOPs,

$$(F_c(i_c, j_c), G_c(i_c, j_c)) = (((-r\,i_0' + i_1'')\,I_c + (-r\,i_0' + i_2'')\,2^{\alpha-\beta}\,J_c + 2W'r\,i_0' - 16W' + 2^{\alpha+\rho+1}) >> (\alpha+\rho+2),\; j_0' + (((-r\,j_0' + j_1'')\,I_c + (-r\,j_0' + j_2'')\,2^{\alpha-\beta}\,J_c + 2^{\alpha+\rho-1}) >> (\alpha+\rho))),$$

for 4:2:2 VOPs,

$$(F_c(i_c, j_c), G_c(i_c, j_c)) = (i_0' + (((-r\,i_0' + i_1'')\,I_c + (-r\,i_0' + i_2'')\,2^{\alpha-\beta}\,J_c + 2^{\alpha+\rho-1}) >> (\alpha+\rho)),\; j_0' + (((-r\,j_0' + j_1'')\,I_c + (-r\,j_0' + j_2'')\,2^{\alpha-\beta}\,J_c + 2^{\alpha+\rho-1}) >> (\alpha+\rho))),$$

for 4:4:4 VOPs.

When no_of_sprite_warping_point == 4,

$$(F(i, j), G(i, j)) = ((a\,I + b\,J + c) \mathbin{///} (g\,I + h\,J + D\,W\,H),\; (d\,I + e\,J + f) \mathbin{///} (g\,I + h\,J + D\,W\,H)),$$

$$(F_c(i_c, j_c), G_c(i_c, j_c)) = ((2\,a\,I_c + 2\,b\,J_c + 4\,c - (g\,I_c + h\,J_c + 2\,D\,W\,H)\,s) \mathbin{///} (4\,g\,I_c + 4\,h\,J_c + 8\,D\,W\,H),\; (2\,d\,I_c + 2\,e\,J_c + 4\,f - (g\,I_c + h\,J_c + 2\,D\,W\,H)\,s) \mathbin{///} (4\,g\,I_c + 4\,h\,J_c + 8\,D\,W\,H)),$$

for 4:2:0 VOPs,

$$(F_c(i_c, j_c), G_c(i_c, j_c)) = ((2\,a\,I_c + 2\,b\,J_c + 4\,c - (g\,I_c + h\,J_c + 2\,D\,W\,H)\,s) \mathbin{///} (4\,g\,I_c + 4\,h\,J_c + 8\,D\,W\,H),\; (d\,I_c + e\,J_c + f) \mathbin{///} (g\,I_c + h\,J_c + D\,W\,H)),$$

for 4:2:2 VOPs,

$$(F_c(i_c, j_c), G_c(i_c, j_c)) = ((a\, I_c + b\, J_c + c) /// (g\, I_c + h\, J_c + D\, W H), (d\, I_c + e\, J_c + f) /// (g\, I_c + h\, J_c + D\, W H)),$$

for 4:4:4 VOPs.

where

$$g = ((i_0' - i_1' - i_2' + i_3')(j_2' - j_3') - (i_2' - i_3')(j_0' - j_1' - j_2' + j_3'))\, H,$$
$$h = ((i_1' - i_3')(j_0' - j_1' - j_2' + j_3') - (i_0' - i_1' - i_2' + i_3')(j_1' - j_3'))\, W,$$
$$D = (i_1' - i_3')(j_2' - j_3') - (i_2' - i_3')(j_1' - j_3'),$$
$$a = D\,(i_1' - i_0')\, H + g\, i_1',$$
$$b = D\,(i_2' - i_0')\, W + h\, i_2',$$
$$c = D\, i_0'\, W\, H,$$
$$d = D\,(j_1' - j_0')\, H + g\, j_1',$$
$$e = D\,(j_2' - j_0')\, W + h\, j_2',$$
$$f = D\, j_0'\, W\, H.$$

A set of parameters that causes the denominator of any of the above equations to be zero for any pixel in an opaque or boundary macroblock is disallowed. The implementor should be aware that a 32bit register may not be sufficient for representing the denominator or the numerator in the above transform functions for affine and perspective transforms. The usage of a 64 bit floating point representation should be sufficient in such cases.

7.16.9.5 Sample reconstruction

The reconstructed value Y of the luminance sample (i, j) in the currently decoded VOP shall be defined as

$$Y = ((s - r_j)((s - r_i)\, Y_{00} + r_i\, Y_{01}) + r_j((s - r_i)\, Y_{10} + r_i\, Y_{11}))\, //\, s^2,$$

where Y_{00}, Y_{01}, Y_{10}, Y_{11} represent the sprite luminance sample at $(F(i, j)////s, G(i, j)////s)$, $(F(i, j)////s + 1, G(i, j)////s)$, $(F(i, j)////s, G(i, j)////s + 1)$, and $(F(i, j)////s + 1, G(i, j)////s + 1)$ respectively, and $r_i = F(i, j) - (F(i, j)////s)s$ and $r_j = G(i, j) - (G(i, j)////s)s$. Figure AMD1-30 illustrates this process.

In case any of Y_{00}, Y_{01}, Y_{10} and Y_{11} lie outside the sprite luminance binary mask, it shall be obtained by the padding process as defined in subclause 7.16.7.1.1.

When brightness_change_in_sprite == 1, the final reconstructed luminance sample (i, j) is further computed as $Y = Y$ * (brightness_change_factor * 0.01 + 1), clipped to the range of [0, 2^(bits_per_pixel-1)].

Similarly, the reconstructed value C of the chrominance sample (i_c, j_c) in the currently decoded VOP shall be defined as

$$C = ((s - r_j)((s - r_i)\, C_{00} + r_i\, C_{01}) + r_j((s - r_i)\, C_{10} + r_i\, C_{11}))\, //\, s^2,$$

where C_{00}, C_{01}, C_{10}, C_{11} represent the sprite chrominance sample at $(F_c(i_c, j_c)////s, G_c(i_c, j_c)////s)$, $(F_c(i_c, j_c)////s + 1, G_c(i_c, j_c)////s)$, $(F_c(i_c, j_c)////s, G_c(i_c, j_c)////s + 1)$, and $(F_c(i_c, j_c)////s + 1, G_c(i_c, j_c)////s + 1)$ respectively, and $r_i = F_c(i_c, j_c) - (F_c(i_c, j_c)////s)s$ and $r_j = G_c(i_c, j_c) - (G_c(i_c, j_c)////s)s$. In case any of C_{00}, C_{01}, C_{10} and C_{11} lies outside the sprite chrominance binary mask, it shall be obtained by the padding process as defined in subclause 7.16.7.1.1.

The same method is used for the reconstruction of grayscale alpha and luminance samples. The reconstructed value A of the grayscale alpha sample (i, j) in the currently decoded VOP shall be defined as

$$A = ((s - r_j)((s - r_i)\, A_{00} + r_i\, A_{01}) + r_j((s - r_i)\, A_{10} + r_i\, A_{11}))\, //\, s^2,$$

where A_{00}, A_{01}, A_{10}, A_{11} represent the sprite grayscale alpha sample at $(F(i, j)////s, G(i, j)////s)$, $(F(i, j)////s + 1, G(i, j)////s)$, $(F(i, j)////s, G(i, j)////s + 1)$, and $(F(i, j)////s + 1, G(i, j)////s + 1)$ respectively, and $r_i = F(i, j) - (F(i, j)////s)s$ and $r_j = G(i, j) - (G(i, j)////s)s$. In case any of A_{00}, A_{01}, A_{10} and A_{11} lies outside the sprite luminance binary mask, it shall be obtained by the padding process as defined in subclause 7.16.7.1.1.

The reconstructed values of luminance binary mask samples *BY*(*i*,*j*) shall be computed following the identical process for the luminance samples. However, corresponding binary mask sample values shall be used in place of luminance samples Y_{00}, Y_{01}, Y_{10}, Y_{11}. Assume the binary mask sample opaque is equal to 255 and the binary mask sample transparent is equal to 0. If the computed value is bigger or equal to 128, *BY*(*i*, *j*) is defined as opaque. Otherwise, *BY* (*i*, *j*) is defined as transparent. The chrominance binary mask samples shall be reconstructed by decimating the corresponding 2 x 2 adjacent luminance binary mask samples as specified in subclause 7.16.7.1.1.4.

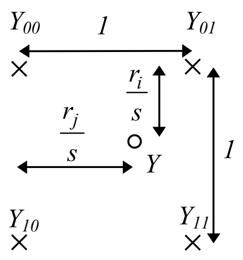


Figure AMD1-31 -- Pixel value interpolation (it is assumed that sprite samples are located on an integer grid)

"

31) Add the following Table in subclause 9.1 after tableV2-39

"

Table AMD1-48 — Tools for ISO/IEC 14496-2:2001 Amendment 1 Visual Object Types

	Visual Object Type	
Visual Tools	**Simple Studio**	**Core Studio**
Basic -Progressive/Interlaced -Frame/Field Structure -Slice Structure -Studio DPCM Block -Studio Binary Shape -Studio Grayscale Shape	X	X
I-VOP	X	X
P-VOP		X
Studio Sprite		X

NOTE 3 — The allowed values of 'chroma_format' in StudioVideoObjectLayer() are defined in the level definition.

NOTE 4 — The allowed values of 'bits_per_pixel' and 'alpha_bits_per_pixel' in StudioVideoObjectLayer() are defined in the level definition

"

32) Add the following Table and text at the end of subclause 9.2:

"

Table AMD1-49 — ISO/IEC 14496-2:2001 Amendment 1 Visual Profiles

	Object Types **Profiles**	**Simple Studio**	**Core Studio**
AMD1-1	Simple Studio	X	
AMD1-2	Core Studio	X	X

Note that object types that are not listed in this table are not decordable by decoders complying to the Profiles listed in this table.

Note that the Profiles listed in this table can be grouped into Natural Visual.

"

33) Add the following subclause A.1.1 in clause A.1:

A.1.1 Discrete cosine transform for the Studio Profile

The NxN two dimensional DCT is defined as:

$$F(u,v) = \frac{2}{N} C(u)C(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos\frac{(2x+1)u\pi}{2N} \cos\frac{(2y+1)v\pi}{2N}$$

with u, v, x, y = 0, 1, 2, … N-1

where x, y are spatial coordinates in the sample domain

u, v are coordinates in the transform domain

$$C(u), C(v) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } u,v = 0 \\ 1 & \text{otherwise} \end{cases}$$

The inverse DCT (IDCT) is defined as:

$$f(x,y) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u)C(v)F(u,v) \cos\frac{(2x+1)u\pi}{2N} \cos\frac{(2y+1)v\pi}{2N}$$

If each pixel is represented by *n* bits per pixel, the input to the forward transform and output from the inverse transform is represented with (*n*+1) bits. The bit precision of the DCT coefficients changes in accordance with the value of mpeg2_stream defined in StudioVideoObjectLayer(). In case of mpeg2_stream = 0, The coefficients are represented in (n+7) bits including three fractional bits. The dynamic range of the coefficients is [-2^{n+6}:$+2^{n+6}-1$]. In case of mpeg2_stream = 1, The coefficients are represented in (n+4) bits. The dynamic range is [-2^{n+3}:$+2^{n+3}-1$].

The N by N inverse discrete transform shall conform to IEEE Standard Specification for the Implementations of 8 by 8 Inverse Discrete Cosine Transform, Std 1180-1990, December 6, 1990.

NOTE 1 Clause 2.3 Std 1180-1990 “Considerations of Specifying IDCT Mismatch Errors” requires the specification of periodic intra-picture coding in order to control the accumulation of mismatch errors. Every macroblock is required to be refreshed before it is coded 132 times as predictive macroblocks. Macroblocks in B-pictures (and skipped macroblocks in P-pictures) are excluded from the counting because they do not lead to the accumulation of mismatch errors. This requirement is the same as indicated in 1180-1990 for visual telephony according to ITU-T Recommendation H.261.

NOTE 2 Whilst the IEEE IDCT standard mentioned above is a necessary condition for the satisfactory implementation of the IDCT function it should be understood that this is not sufficient. In particular, attention is drawn to the following sentence from subclause 5.4: “Where arithmetic precision is not specified, such as the calculation of the IDCT, the precision shall be sufficient so that significant errors do not occur in the final integer values.”

34) Add the following clause B.3 after clause B.2:

"

B.3 Variable length codes for the studio profile

B.3.1 Macroblock type

The properties of the macroblock are determined by the macroblock type VLC according to these tables.

Table AMD1-50 -- Variable length codes for macroblock_type in I-VOPs

macroblock_type VLC code	macroblock_quant	macroblock_motion_forward	macroblock_motion_backward	macroblock_pattern	macroblock_intra	Description
1	0	0	0	0	1	Intra
01	1	0	0	0	1	Intra, Quant

Table AMD1-51 -- Variable length codes for macroblock_type in P-VOPs

macroblock_type VLC code	macroblock_quant	macroblock_motion_forward	macroblock_motion_backward	macroblock_pattern	macroblock_intra	Description
1	0	1	0	1	0	MC, Coded
01	0	0	0	1	0	No MC, Coded
001	0	1	0	0	0	MC, Not Coded
0001 1	0	0	0	0	1	Intra
0001 0	1	1	0	1	0	MC, Coded, Quant
0000 1	1	0	0	1	0	No MC, Coded, Quant
0000 01	1	0	0	0	1	Intra, Quant

B.3.2 DCT coefficients

Table AMD1-52 -- Variable length codes for dct_dc_size_luminance

Variable length code	dct_dc_size_luminance
001110	0
00110	1
0000	2
0010	3
111	4
101	5
011	6
010	7
100	8
110	9
0001	10
0011110	11
00111110	12
001111110	13
0011111110	14
00111111110	15
001111111110	16
0011111111110	17
0011111111111	18

Table AMD1-53 -- Variable length codes for dct_dc_size_chrominance

Variable length code	dct_dc_size_chrominance
0000	0
0010	1
111	2
101	3
011	4
010	5
100	6
110	7
0001	8
00110	9
001110	10
0011110	11
00111110	12
001111110	13
0011111110	14
00111111110	15
001111111110	16
0011111111110	17
0011111111111	18

Table AMD1-54-1 — Differential DC additional codes

Additional code	Differential DC	Size
000000000000000000 to 011111111111111111 *	-131072 to -262143	18
00000000000000000 to 01111111111111111 *	-65536 to -131071	17
0000000000000000 to 0111111111111111 *	-32768 to -65535	16
000000000000000 to 011111111111111 *	-16384 to -32767	15
00000000000000 to 01111111111111 *	-8192 to -16383	14
0000000000000 to 0111111111111 *	-4096 to -8191	13
000000000000 to 011111111111 *	-2048 to -4095	12
00000000000 to 01111111111 *	-1024 to -2047	11
0000000000 to 0111111111 *	-512 to -1023	10
000000000 to 011111111 *	-256 to -511	9
00000000 to 01111111	-255 to -128	8
0000000 to 0111111	-127 to -64	7
000000 to 011111	-63 to -32	6
00000 to 01111	-31 to -16	5
0000 to 0111	-15 to -8	4
000 to 011	-7 to -4	3
00 to 01	-3 to -2	2
0	-1	1
	0	0
1	1	1
10 to 11	2 to 3	2
100 to 111	4 to 7	3
1000 to 1111	8 to 15	4
10000 to 11111	16 to 31	5
100000 to 111111	32 to 63	6
1000000 to 1111111	64 to 127	7
10000000 to 11111111	128 to 255	8
100000000 to 111111111 *	256 to 511	9
1000000000 to 1111111111 *	512 to 1023	10
10000000000 to 11111111111 *	1024 to 2047	11
100000000000 to 111111111111 *	2048 to 4095	12
1000000000000 to 1111111111111 *	4096 to 8191	13
10000000000000 to 11111111111111 *	8192 to 16383	14
100000000000000 to 111111111111111 *	16384 to 32767	15
1000000000000000 to 1111111111111111 *	32768 to 65535	16
10000000000000000 to 11111111111111111 *	65536 to 131071	17
100000000000000000 to 111111111111111111 *	131072 to 262143	18

In cases where dct_dc_size is greater than 8, marked '*' in Table AMD1-54-1, a marker bit is inserted after the dct_dc_additional_code to prevent start code emulations.

Table AMD1-54-2 — Additional codes for group No.1~6 (zero run length)

Size	Additional code	zero run length
0	---	1
1	0 to 1	2 to 3
2	00 to 11	4 to 7
3	000 to 111	8 to 15
4	0000 to 1111	16 to 31
5	00000 to 11111	32 to 63

Table AMD1-54-3 — Additional codes for group No.7~12 (zero run length and +/-1 level)

Size	Additional code	zero run length	level(+1/-1)
6	000000 to 111110	32 to 63	-1
5	00000 to 11110	16 to 31	-1
4	0000 to 1110	8 to 15	-1
3	000 to 110	4 to 7	-1
2	00 to 10	2 to 3	-1
1	0	1	-1
0	---	---	---
1	1	1	+1
2	01 to 11	2 to 3	+1
3	001 to 111	4 to 7	+1
4	0001 to 1111	8 to 15	+1
5	00001 to 11111	16 to 31	+1
6	000001 to 111111	32 to 63	+1

Table AMD1-54-4 — Additional codes for group No.13~20 (level value)

Size	Additional code	level value
8	00000000 to 01111111	-255 to -128
7	0000000 to 0111111	-127 to -64
6	000000 to 011111	-63 to -32
5	00000 to 01111	-31 to -16
4	0000 to 0111	-15 to -8
3	000 to 011	-7 to -4
2	00 to 01	-3 to -2
1	0	-1
0	---	---
1	1	1
2	10 to 11	2 to 3
3	100 to 111	4 to 7
4	1000 to 1111	8 to 15
5	10000 to 11111	16 to 31
6	100000 to 111111	32 to 63
7	1000000 to 1111111	64 to 127
8	10000000 to 11111111	128 to 255

Table AMD1-55 — VLC Table T[0] for intra macroblock

symbol group	variable length code
0	0011
1	0010
2	1001110
3	100111101
4	10011110010
5	100111100110
6	1001111001110
7	1000
8	10010
9	011011
10	0110100
11	0110101
12	10011111
13	11
14	000
15	010
16	101
17	0111
18	01100
19	100110
20	1001111000
21	1001111001111

Table AMD1-56 — VLC Table T[1] for intra macroblock

symbol group	variable length code
0	---
1	---
2	---
3	---
4	---
5	---
6	---
7	---
8	---
9	---
10	---
11	---
12	---
13	---
14	0
15	10
16	110
17	1110
18	11110
19	111110
20	1111110
21	1111111

Table AMD1-57 — VLC Table T[2] for intra macroblock

symbol group	variable length code
0	110
1	01110
2	11111
3	0111101
4	011110000
5	01111000111110
6	0111100011111111
7	010
8	100
9	0110
10	1110
11	011111
12	0111100010
13	00
14	101
15	11110
16	01111001
17	01111000110
18	011110001110
19	011110001111110
20	0111100011111110
21	0111100011110

Table AMD1-58 — VLC Table T[3] for intra macroblock

symbol group	variable length code
0	10001
1	1001
2	10111
3	1000001
4	1000000000
5	1000000001101111
6	1000000001101110
7	011
8	0100
9	1010
10	10110
11	100000001
12	10000000011010
13	00
14	11
15	0101
16	100001
17	10000001
18	10000000010
19	100000000111
20	100000000110110
21	1000000001100

Table AMD1-59 — VLC Table T[4] for intra macroblock

symbol group	variable length code
0	1000110
1	1001
2	11111
3	1111011
4	111101010
5	111101011010110
6	111101011010111
7	110
8	1110
9	100010
10	1000111
11	1111010111
12	11110101101010
13	00
14	01
15	101
16	10000
17	111100
18	11110100
19	11110101100
20	111101011011
21	1111010110100

Table AMD1-60 — VLC Table T[5] for intra macroblock

symbol group	variable length code
0	0000101001
1	00000
2	0000110
3	0000101010
4	000010100010
5	0000101000111100
6	0000101000111101
7	00010
8	0000100
9	00001011
10	00001010000
11	000010100011100
12	000010100011101
13	11
14	01
15	10
16	001
17	00011
18	0000111
19	0000101011
20	000010100011111
21	0000101000110

Table AMD1-61 — VLC Table T[6] for intra macroblock

symbol group	variable length code
0	11000011110
1	1101
2	11000001
3	1100001110
4	11000011010
5	1100001111111010
6	1100001111111011
7	110001
8	11000000
9	1100001100
10	110000111110
11	11000011111111
12	1100001111111100
13	001
14	000
15	01
16	10
17	111
18	11001
19	11000010
20	11000011011
21	1100001111110

Table AMD1-62 — VLC Table T[7] for intra macroblock

symbol group	variable length code
0	0011011110
1	00111
2	001100101
3	00110110
4	00110011
5	00110111111110
6	001101111111111
7	0011010
8	001100100
9	0011011101
10	00110111110
11	001101111110
12	001101111111110
13	0010
14	0000
15	11
16	01
17	10
18	0001
19	0011000
20	0011011100
21	0011011111110

Table AMD1-63 — VLC Table T[8] for intra macroblock

symbol group	variable length code
0	1100000001
1	110001
2	1100000011
3	11000000100
4	11000010
5	11000000101000
6	110000001010011
7	11000001
8	110000110
9	1100001110
10	1100001111
11	110000001011
12	110000001010010
13	1101
14	111
15	001
16	10
17	01
18	000
19	11001
20	1100000000
21	1100000010101

Table AMD1-64 — VLC Table T[9] for intra macroblock

symbol group	variable length code
0	001001011111000
1	0010001
2	0010010110
3	0010011
4	00100001
5	001001011110
6	001001011111011
7	00100100
8	001000001
9	001001010
10	00100101110
11	001001011111001
12	001001011111010
13	00101
14	0011
15	110
16	000
17	10
18	01
19	111
20	001000000
21	0010010111111

Table AMD1-65 — VLC Table T[10] for intra macroblock

symbol group	variable length code
0	000111101101011
1	0001110
2	00011011
3	1101
4	000111101100
5	00011110110100
6	000111101101010
7	0001100
8	00011010
9	000111100
10	0001111010
11	00011110111
12	00011111011011
13	00000
14	00001
15	1100
16	111
17	001
18	10
19	01
20	00010
21	00011111

Table AMD1-66 – VLC Table T[11] for intra macroblock

symbol group	variable length code
0	010110001111011
1	0101101
2	010110000
3	010110001011
4	0101100011101
5	01011000111100
6	010110001111010
7	010111
8	01011001
9	01011000110
10	010110001010
11	0101100011100
12	0101100011111
13	0100
14	0110
15	01010
16	0111
17	0011
18	0010
19	000
20	1
21	01011000100

Table AMD1-67 — VLC Table T[0] for inter macroblock

symbol group	variable length code
0	---
1	11000
2	11001
3	1101011
4	110101000
5	11010100110
6	1101010011110
7	111
8	011
9	0000
10	0101
11	01001
12	11011
13	10
14	001
15	0001
16	01000
17	110100
18	11010101
19	1101010010
20	110101001110
21	1101010011111

Table AMD1-68 — VLC Table T[1] for inter macroblock

symbol group	variable length code
0	---
1	---
2	---
3	---
4	---
5	---
6	---
7	---
8	---
9	---
10	---
11	---
12	---
13	---
14	0
15	10
16	110
17	1110
18	11110
19	111110
20	1111110
21	1111111

Table AMD1-69 — VLC Table T[2] for inter macroblock

symbol group	variable length code
0	000
1	00111
2	10111
3	001101
4	00110001
5	00110000110
6	00110000111110
7	100
8	010
9	011
10	0010
11	10110
12	001100000
13	11
14	1010
15	0011001
16	0011000010
17	001100001110
18	001100001111110
19	0011000011111110
20	0011000011111111
21	0011000011110