Table 6-7 -- Colour Primaries

Value	Primaries		
0	(forbidden)		
1	ITU-R Recommendation 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	ITU-R Recommendation 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	ITU-R Recommendation 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	0,3127	0,3290
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	0,3290
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 video_signal_type() is not present in the bitstream or colour_description is zero the chromaticity is assumed to be that corresponding to colour_primaries having the value 1.

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

Table 6-8 -- Transfer Characteristics

Value	Transfer Characteristic
0	(forbidden)
1	ITU-R Recommendation 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	ITU-R Recommendation BT.470-2 System M Assumed display gamma 2.2
5	ITU-R Recommendation 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) $V = 1.0 - Log_{10}(Lc)/2$ for $1 = L_c = 0.01$ $V = 0.0$ for $0.01 > L_c$
10	Logarithmic transfer characteristic (316.22777:1 range) $V = 1.0 - Log_{10}(Lc)/2.5$ for $1 = L_c = 0.0031622777$ $V = 0.0$ for $0.0031622777 > L_c$
11-255	reserved

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

matrix_coefficients: This 8-bit integer describes the matrix coefficients used in deriving luminance and chrominance signals from the green, blue, and red primaries, and is defined in Table 6-9.

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.

Table 6-9 -- Matrix Coefficients

Value	**Matrix**
0	(forbidden)
1	ITU-R Recommendation BT.709 $E'_{Y} = 0{,}7152\, E'_{G} + 0{,}0722\, E'_{B} + 0{,}2126\, E'_{R}$ $E'_{PB} = -0{,}386\, E'_{G} + 0{,}500\, E'_{B} -0{,}115\, E'_{R}$ $E'_{PR} = -0{,}454\, E'_{G} - 0{,}046\, E'_{B} + 0{,}500\, E'_{R}$
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	ITU-R Recommendation 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 video_signal_type() 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.

In the case that video_signal_type() 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).

6.3.2.1 User data

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.

6.3.3 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 marks a new video object layer.

video_object_layer_id: This is given by the last 4-bits of the video_object_layer_start_code. The video_object_layer_id uniquely identifies a video object layer.

short_video_header: The short_video_header is an internal flag which is set to 1 when an abbreviated header format is used for video content. This indicates video data which begins with a short_video_start_marker rather than a longer start code such as visual_object_ start_code. The short header format is included herein to provide forward compatibility with video codecs designed using the earlier video coding specification ITU-T Recommendation H.263. All decoders which support video objects shall support both header formats (short_video_header equal to 0 or 1) for the subset of video tools that is expressible in either form.

video_plane_with_short_header(): This is a syntax layer encapsulating a video plane which has only the limited set of capabilities available using the short header format.

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 he indicated object type only, e.g. Simple Object or Core Object, as shown in Table 6-10.

Table 6-10 – 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 FBA	00001110
Reserved	00001111 – 11111111

is_object_layer_identifier: This is a 1-bit code which when set to '1' indicates that version identification and priority is specified for the visual object layer. When set to '0', no version identification or priority needs to be specified.

video_object_layer_verid: This is a 4-bit code which identifies the version number of the video object layer. Its meaning is defined in Table 6-11. If both visual_object_verid and video_object_layer_verid exist, the semantics of video_object_layer_verid supersedes the other. When this field does not exist, the value of video_object_layer_verid is substituted by the value of visual_object_verid.

Table 6-11 – 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 - 1111	Reserved

video_object_layer_priority: This is a 3-bit code which specifies the priority of the video object layer. It takes values between 1 and 7, with 1 representing the highest priority and 7, the lowest priority. The value of zero is reserved.

aspect_ratio_info: This is a four-bit integer which defines the value of pixel aspect ratio. Table 6-12 shows the meaning of the code. If aspect_ratio_info indicates extended PAR, pixel_aspect_ratio is represented by par_width and par_height. The par_width and par_height shall be relatively prime.

Table 6-12 -- 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 picure)
0101	40:33 (525-type stretched for 16:9 picure)
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.

vol_control_parameters: This a one-bit flag which when set to '1' indicates presence of the following parameters: chroma_format, low_delay, and vbv_parameters.

chroma_format: This is a two bit integer indicating the chrominance format as defined in the Table 6-13.

Table 6-13 -- Meaning of chroma_format

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

low_delay : This is a one-bit flag which when set to '1' indicates the VOL contains no B-VOPs. If this flag is not present in the bitstream, the default value is 0 for visual object types that support B-VOP otherwise it is 1.

vbv_parameters: This is a one-bit flag which when set to '1' indicates presence of following VBV parameters: first_half_bit_rate, latter_half_bit_rate, first_half_vbv_buffer_size, latter_half_vbv_buffer_size, first_half_vbv_occupancy and latter_half_vbv_occupancy. 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 resync_marker 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 $R_{vol}(t) <= 400 \times \mathit{bit\_rate}$ 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.

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA V. AGAWANE
ANSI Store order X037445 Downloaded: 7/18/2007 3:58:21 PM ET
Single user license only. Copying and networking prohibited.

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_half_vbv_occupancy and the latter_half_vbv_occupancy in order to avoid the resync_marker 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.

video_object_layer_shape: This is a 2-bit integer defined in Table 6-14. It identifies the shape type of a video object layer.

Table 6-14 -- 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 V2 - 1. 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 V2 - 1. More applications are possible by selection of the USER DEFINED type.

Table V2 - 1 -- 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
1101-1111	t.b.d.	t.b.d.	t.b.d.	t.b.d.

vop_time_increment_resolution: This is a 16-bit unsigned integer that indicates the number of evenly spaced subintervals, called ticks, within one modulo time. One modulo time represents the fixed interval of one second. The value zero is forbidden.

fixed_vop_rate: This is a one-bit flag which indicates that all VOPs are coded with a fixed VOP rate. It shall only be '1' if and only if all the distances between the display time of any two successive VOPs in the display order in the video object layer are constant. In this case, the VOP rate can be derived from the fixed_vop_time_increment. If it is '0' the display time between any two successive VOPs in the display order can be variable thus indicated by the time stamps provided in the VOP header.

fixed_vop_time_increment: This value represents the number of ticks between two successive VOPs in the display order. The length of a tick is given by vop_time_increment_resolution. It can take a value in the range of [0,vop_time_increment_resolution). The number of bits representing the value is calculated as the minimum number of unsigned integer bits required to represent the above range. fixed_vop_time_increment shall only be present if fixed_vop_rate is '1' and its value must be identical to the constant given by the distance between the display time of any two successive VOPs in the display order. In this case, the fixed VOP rate is given as (vop_time_increment_resolution / fixed_vop_time_increment). A zero value is forbidden.

EXAMPLE
VOP time = tick × vop_time_increment
= vop_time_increment / vop_time_increment_resolution

Table 6-15 – Examples of vop_time_increment_resolution, fix_vop_time_increment, and vop_time_increment

Fixed VOP rate = 1/VOP time	vop_time_increment_ resolution	fixed_vop_time_ increment	vop_time_increment
15Hz	15	1	0, 1, 2, 3, 4,…
7.5Hz	15	2	0, 2, 4, 6, 8,…
29.97…Hz	30000	1001	0, 1001, 2002, 3003,…
59.94…Hz	60000	1001	0, 1001, 2002, 3003,…

video_object_layer_width: The video_object_layer_width is a 13-bit unsigned integer representing the width of the displayable part of the luminance component 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. A zero value is forbidden.

video_object_layer_height: The video_object_layer_height is a 13-bit unsigned integer representing the height of the displayable part of the luminance component in pixel units. The height of the encoded luminance component of VOPs in macroblocks is (video_object_layer_height+15)/16. The displayable part is top-aligned in the encoded VOPs. A zero value is forbidden.

interlaced: This is a 1 bit flag which, when set to "1" indicates that the VOP may contain interlaced video. When this flag is set to "0", the VOP is of non-interlaced (or progressive) format.

obmc_disable: This is a one-bit flag which when set to '1' disables overlapped block motion compensation.

sprite_enable: When video_object_layer_verid == '0001', this is a one-bit flag which when set to '1' indicates the usage of static (basic or low latency) sprite coding. When video_object_layer_verid == '0002', this is a two-bit unsigned integer which indicates the usage of static sprite coding or global motion compensation (GMC). Table V2 - 2 shows the meaning of various codewords. An S-VOP with sprite_enable == "GMC" is referred to as an S (GMC)-VOP in this document.

Table V2 - 2 – Meaning of sprite_enable codewords

sprite_enable (video_object_layer_ verid == '0001')	sprite_enable (video_object_layer_ verid == '0002')	Sprite Coding Mode
0	00	sprite not used
1	01	static (Basic/Low Latency)
--	10	GMC (Global Motion Compensation)
--	11	Reserved

sprite_width: This is a 13-bit unsigned integer which identifies the horizontal dimension of the sprite.

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/OSPO/JA Y. ADAMANE
ANSI Store order #X37445 Downloaded 7/18/2007 3:58:21 PM ET
Single user license only. Copying and networking prohibited.

sprite_height: This is a 13-bit unsigned integer which identifies the vertical dimension of the sprite.

sprite_left_coordinate: This is a 13-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 13-bit signed integer which defines the top edge of the sprite. The value of sprite_top_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 'static' or 'GMC', warping is identity (stationary sprite) and no coordinates need to be coded. When its value is 4, a perspective transform is used. When its value is 1,2 or 3, an affine transform is used. Further, the case of value 1 is separated as a special case from that of values 2 or 3. Table 6-16 shows the various choices. Note that the value of 4 is disallowed when sprite_enable == 'GMC'.

Table 6-16 – Number of point and implied warping function

Number of points	warping function
0	Stationary
1	Translation
2,3	Affine
4	Perspective
5-63	Reserved

sprite_warping_accuracy – This is a 2-bit code which indicates the quantisation accuracy of motion vectors used in the warping process for sprites and GMC. Table 6-17 shows the meaning of various codewords.

Table 6-17 – 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.

low_latency_sprite_enable: This is a one-bit flag which when set to "1" indicates the presence of low_latency sprite, alternatively, a value of "0" means basic sprite.

not_8_bit: This one bit flag is set when the video data precision is not 8 bits per pixel and visual object type is N-bit.

sadct_disable: This is a one-bit flag specifying the inverse transforms to be used for texture decoding. If 'sadct_disable' is set to '1', standard inverse DCT as described in version 1 is applied to all 8x8-blocks. When set to '0', flag 'sadct_disable' indicates that different types of inverse DCT are used in an adaptive way: standard inverse DCT is applied to those 8x8-blocks where all 64 pels are opaque, whereas inverse shape-adaptive DCT (SA-DCT) and inverse ΔDC-SA-DCT – an extended version of SA-DCT - are used in inter- and intra-coded 8x8-blocks with at least one transparent and one opaque pel.

quant_precision: This field specifies the number of bits used to represent quantiser parameters. Values between 3 and 9 are allowed. When not_8_bit is zero, and therefore quant_precision is not transmitted, it takes a default value of 5.

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. When not_8_bit is zero and bits_per_pixel is not present, the video data precision is always 8 bits per pixel, which is equivalent to specifying a value of 8 in this field. The same number of bits per pixel is used in the luminance and two chrominance planes. The alpha plane, used to specify shape of video objects, is always represented with 8 bits per pixel.

no_gray_quant_update: This is a one bit flag which is set to '1' when a fixed quantiser is used for the decoding of grayscale alpha data. When this flag is set to '0', the grayscale alpha quantiser is updated on every macroblock by generating it anew from the luminance quantiser value, but with an appropriate scale factor applied. See the description in subclause 7.5.4.3.

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 7.5.4.6.

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 7.5.4.6.

quant_type: This is a one-bit flag which when set to '1' that the first inverse quantisation method and when set to '0' indicates that the second inverse quantisation method is used for inverse quantisation of the DCT coefficients. Both inverse quantisation methods are described in subclause 7.4.4. For the first inverse quantisation method, two matrices are used, one for intra blocks the other for non-intra blocks.

The default matrix for intra blocks 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 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_quant_mat: This is a one-bit flag which is set to '1' when intra_quant_mat follows. If it is set to '0' then there is no change in the values that shall be used.

intra_quant_mat: This is a list of 2 to 64 eight-bit unsigned integers. The new values are in zigzag scan order and replace the previous values. A value of 0 indicates that no more values are transmitted and the remaining, non-transmitted values are set equal to the last non-zero value. The first value shall always be 8 and is not used in the decoding process.

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

nonintra_quant_mat: This is a list of 2 to 64 eight-bit unsigned integers. The new values are in zigzag scan order and replace the previous values. A value of 0 indicates that no more values are transmitted and the remaining, non-transmitted values are set equal to the last non-zero value. The first value shall not be 0.

load_intra_quant_mat_grayscale: This is a one-bit flag which is set to '1' when intra_quant_mat_grayscale follows. If it is set to '0' then there is no change in the quantisation matrix values that shall be used.

intra_quant_mat_grayscale: This is a list of 2 to 64 eight-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_quant_mat.

load_nonintra_quant_mat_grayscale: This is a one-bit flag which is set to '1' when nonintra_quant_mat_grayscale[i] follows for grayscale alpha or auxiliary component i=0,1,2. If it is set to '0' then there is no change in the quantisation matrix values that shall be used.

intra_quant_mat_grayscale[i]: This is a list of 2 to 64 eight-bit unsigned integers defining the grayscale intra alpha quantisation matrix to be used for grayscale alpha or auxiliary component i=0,1,2. The semantics and the default quantisation matrix are identical to those of intra_quant_mat.

nonintra_quant_mat_grayscale: This is a list of 2 to 64 eight-bit unsigned integers defining the grayscale nonintra alpha quantisation matrix[i] to be used for grayscale alpha or auxiliary component i=0,1,2. The semantics and the default quantisation matrix are identical to those ofnonintra_quant_mat.

nonintra_quant_mat_grayscale[i]: This is a list of 2 to 64 eight-bit unsigned integers defining the grayscale nonintra alpha quantisation matrix to be used for grayscale alpha or auxiliary component i=0,1,2. The semantics and the default quantisation matrix are identical to those ofnonintra_quant_mat.

quarter_sample: This is a one-bit flag which when set to '0' indicates that half sample mode and when set to '1' indicates that quarter sample mode shall be used for motion compensation of the luminance component.

complexity_estimation_disable: This is a one-bit flag which, when set to '1', disables complexity estimation header in each VOP.

estimaton method: Setting of the of the estimation method, it is „00" for Version 1 and "01" for version 2.

shape_complexity_estimation_disable: This is a one-bit flag which when set to '1' disables shape complexity estimation.

opaque: Fag enabling transmission of the number of luminance and chrominance blocks coded using opaque coding mode in % of the total number of blocks (bounding rectangle).

transparent: Flag enabling transmission of the number of luminance and chrominance blocks coded using transparent mode in % of the total number of blocks (bounding rectangle).

intra_cae: Flag enabling transmission of the number of luminance and chrominance blocks coded using IntraCAE coding mode in % of the total number of blocks (bounding rectangle).

inter_cae: Flag enabling transmission of the number of luminance and chrominance blocks coded using InterCAE coding mode in % of the total number of blocks (bounding rectangle).

no_update: Flag enabling transmission of the number of luminance and chrominance blocks coded using no update coding mode in % of the total number of blocks (bounding rectangle).

upsampling: Flag enabling transmission of the number of luminance and chrominance blocks which need upsampling from 4-4- to 8-8 block dimensions in % of the total number of blocks (bounding rectangle).

version2_complexity_estimation_disable: Flag to disable version 2 parameter set.

sadct: Flag enabling transmission of the number of luminance and chrominance blocks coded using SADCT coding mode in % of the total number of blocks (bounding box). When estimation_method == '00' the value of sadct is set to '0'.

quarterpel: Flag enabling transmission of the number of luminance and chrominance block predicted by a quarter-pel vector on one or two dimensions (horizontal and vertical) in % of the total number of blocks (bounding box). When estimation_method == '00' the value of quarterpel is set to '0'.

texture_complexity_estimation_set_1_disable: Flag to disable texture parameter set 1.

intra_blocks: Flag enabling transmission of the number of luminance and chrominance Intra or Intra+Q coded blocks in % of the total number of blocks (bounding rectangle).

inter_blocks: Flag enabling transmission of the number of luminance and chrominance Inter and Inter+Q coded blocks in % of the total number of blocks (bounding rectangle).

inter4v_blocks: Flag enabling transmission of the number of luminance and chrominance Inter4V coded blocks in % of the total number of blocks (bounding rectangle).

not_coded_blocks: Flag enabling transmission of the number of luminance and chrominance Non Coded blocks in % of the total number of blocks (bounding rectangle).

texture_complexity_estimation_set_2_disable: Flag to disable texture parameter set 2.

dct_coefs: Flag enabling transmission of the number of DCT coefficients % of the maximum number of coefficients (coded blocks).

dct_lines: Flag enabling transmission of the number of DCT8x1 in % of the maximum number of DCT8x1 (coded blocks).

vlc_symbols: Flag enabling transmission of the average number of VLC symbols formacroblock.

vlc_bits: Flagenabling transmission of the average number of bits for each symbol.

motion_compensation_complexity_disable: Flag to disable motion compensation parameter set.

apm (Advanced Prediction Mode): Flag enabling transmission of the number of luminance block predicted using APM in % of the total number of blocks for VOP (bounding rectangle).

npm (Normal Prediction Mode): Flag enabling transmission of the number of luminance and chrominance blocks predicted using NPM in % of the total number of luminance and chrominance for VOP (bounding rectangle).

interpolate_mc_q: Flag enabling transmission of the number of luminance and chrominance interpolated blocks in % of the total number of blocks for VOP (bounding rectangle).

forw_back_mc_q: Flag enabling transmission of the number of luminance and chrominance predicted blocks in % of the total number of blocks for VOP (bounding rectangle).

halfpel2: Flag enabling transmission of the number of luminance and chrominance block predicted by a half-pel vector on one dimension (horizontal or vertical) in % of the total number of blocks (bounding rectangle).

halfpel4: Flag enabling transmission of the number of luminance and chrominance block predicted by a half-pel vector on two dimensions (horizontal and vertical) in % of the total number of blocks (bounding rectangle).

resync_marker_disable: This is a one-bit flag which when set to '1' indicates that there is no resync_marker in coded VOPs. This flag can be used only for the optimization of the decoder operation. Successful decoding can be carried out without taking into account the value of this flag.

data_partitioned: This is a one-bit flag which when set to '1' indicates that the macroblock data is rearranged differently, specifically, motion vector data is separated from the texture data (i.e., DCT coefficients).

reversible_vlc: This is a one-bit flag which when set to '1' indicates that the reversible variable length tables (Table B-23, Table B-24 and Table B-25) should be used when decoding DCT coefficients. These tables can only be used when data_partition flag is enabled. Note that this flag shall be treated as '0' in B-VOPs. Use of escape sequence (Table B-24 and Table B-25) for encoding the combinations listed in Table B-23 is prohibited.

newpred_enable: This is a one-bit flag which, when set to '1', indicates that the NEWPRED mode is enabled. When video_object_layer_verid is equal to '0001', and therefore newpred enable is not transmitted, it takes a default value of zero.

requested_upstream_message_type: This is a two-bits flag which indicates which type of upstream message is needed by the encoder. The syntax and semantics of the upstream message are described in subclause 6.2.12 and 6.3.12.

01: need NP_ACK message to be returned for each NEWPRED segment
10: need NP_NACK message to be returned for each NEWPRED segment
11: need both NP_ACK and NP_NACK messages to be returned for each NEWPRED segment
00: reserved

newpred_segment_type: This is a one-bits flag which indicates the unit of selecting reference VOP (NEWPRED segment).

0: Video Packet
1: VOP

reduced_resolution_vop_enable: This is a one-bit flag which indicates that the reduced resolution VOP tool is enabled when set to '1'. When video_object_layer_verid is equal to '0001', and therefore reduced_resolution_vop_enable is not transmitted, it takesa default value of zero.

scalability: This is a one-bit flag which when set to '1' indicates that the current layer uses scalable coding. If the current layer is used as base-layer then this flag is set to '0'. Additionally, this flag shal be set to '0' for S(GMC)-VOPs.

hierarchy_type: The hierarchical relation between the associated hierarchy layer and its hierarchy embedded layer is defined as shown in Table 6-18.

Table 6-18 -- Code table for hierarchy_type

Description	Code
ISO/IEC 14496-2 Spatial Scalability	0
ISO/IEC 14496-2 Temporal Scalability	1

ref_layer_id: This is a 4-bit unsigned integer with value between 0 and 15. It indicates the layer to be used as reference for prediction(s) in the case of scalability.

ref_layer_sampling_direc: This is a one-bit flag which when set to '1' indicates that the resolution of the reference layer (specified by reference_layer_id) is higher than the resolution of the layer being coded. If it is set to '0' then the reference layer has the same or lower resolution than the resolution of the layer being coded.

hor_sampling_factor_n: This is a 5-bit unsigned integer which forms the numerator of the ratio used in horizontal spatial resampling in scalability. The value of zero is forbidden.

hor_sampling_factor_m: This is a 5-bit unsigned integer which forms the denominator of the ratio used in horizontal spatial resampling in scalability. The value of zero is forbidden.

vert_sampling_factor_n: This is a 5-bit unsigned integer which forms the numerator of the ratio used in vertical spatial resampling in scalability. The value of zero is forbidden.

vert_sampling_factor_m: This is a 5-bit unsigned integer which forms the denominator of the ratio used in vertical spatial resampling in scalability. The value of zero is forbidden.

enhancement_type: This is a 1-bit flag which is set to '1' when the current layer enhances the partial region of the reference layer. If it is set to '0' then the current layer enhances the entire region of the reference layer. The default value of this flag is '0'.

use_ref_shape: This is a one bit flag which indicates the procedure to decode binary shape for spatial scalability. If it is set to '0', scalable shape coding should be used. If it is set to '1' and enhancement_type is set to '0', no shape data is decoded and the up-sampled binary shape of reference_layer should be used for enhancement layer. If enhancement_type is set to '1' and this flag is set to '1', binary shape of the enhancement layer should be decoded as the same non-scalable decoding process. When video_object_layer_verid == '0001', the value of use_ref_shape is set to '1'.

use_ref_texture: Reserved flag for future extension. This flag shall be 0 when video_object_layer_ver_id is "0001" or "0010".

shape_hor_sampling_factor_n: This is a 5-bit unsigned integer which forms the numerator of the ratio used in horizontal spatial resampling in shape scalability. The value of zero is forbidden.

shape_hor_sampling_factor_m: This is a 5-bit unsigned integer which forms the denominator of the ratio used in horizontal spatial resampling in shape scalability. The value of zero is forbidden.

shape_vert_sampling_factor_n: This is a 5-bit unsigned integer which forms the denominator of the ratio used in vertical spatial resampling in shape scalability. The value of zero is forbidden.

shape_vert_sampling_factor_m: This is a 5-bit unsigned integer which forms the denominator of the ratio used in vertical spatial resampling in shape scalability. The value of zero is forbidden.

stuffing_start_code: This is the bit string '000001C3' in hexadecimal. It is used in conjunction with possibly following stuffing_byte(s) for the purpose of stuffing bits to guaranty the VBV buffer regulation.

stuffing_byte: This is the 8-bit string in which the value is '11111111'.

6.3.4 Group of Video Object Plane

group_of_vop_start_code: The group_of_vop_start_code is the bit string '000001B3' in hexadecimal. It identifies the beginning of a GOV header.

time_code: This is a 18-bit integer containing the following: time_code_hours, time_code_minutes, marker_bit and time_code_seconds as shown in Table 6-19. The parameters correspond to those defined in the IEC standard publication 461 for "time and control codes for video tape recorders". The time code specifies the modulo part (i.e. the full second units) of the time base for the first object plane (in display order) after the GOV header.

Table 6-19 -- Meaning of time_code

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

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 GOV 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 pictures 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 plane 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-VOP following the group of plane 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.

6.3.5 Video Object Plane and Video Plane with Short Header

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

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 coded VOP (S). The meaning of vop_coding_type is defined in Table 6-20.

Table 6-20 -- Meaning of vop_coding_type

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

modulo_time_base: This value represents the local time base in one second resolution units (1000 milliseconds). It consists of a number of consecutive '1' followed by a '0'. Each '1' represents a duration of one second that have elapsed. For I-, S(GMC)-, and P-VOPs of a non scalable bitstream and the base layer of a scalable bitstream, the number of '1's indicate the number of seconds elapsed since the synchronization point marked by time_code of the previous GOV header or by modulo_time_base of the previously decoded I-, S(GMC)-, or P-VOP, in decoding order. For B-VOP of a non scalable bitstream and a base layer of a scalable bitstream, the number of '1's indicates the number of seconds elapsed since the synchronization point marked in the previous GOV header, or I-VOP, S(GMC)-VOP, or P-VOP, in display order. For I-, P-, or B-VOPs of enhancement layer of scalable bitstream, the number of '1's indicate the number of seconds elapsed since the synchronization point marked in the previous GOV header, I-VOP, P-VOP, or B-VOP, in display order

vop_time_increment: This value represents the absolute vop_time_increment from the synchronization point marked by the modulo_time_base measured in the number of clock ticks. It can take a value in the range of [0,vop_time_increment_resolution). The number of bits representing the value is calculated as the minimum number of unsigned integer bits required to represent the above range. The local time base in the units of seconds is recovered by dividing this value by the vop_time_increment_resolution.

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 rules apply: If binary shape or alpha plane does exist for the VOP (i.e. video_object_layer_shape != "rectangular"), it shall be completely transparent. If binary shape or alpha plane does not exist for the VOP (i.e. video_object_layer_shape == "rectangular"), the luminance and chrominance planes of the reconstructed VOP shall be filled with the forward reference VOP as defined in subclause 7.6.7.

vop_rounding_type: This is a one-bit flag which signals the value of the parameter rounding_control used for pixel value interpolation in motion compensation for P- and S(GMC)- VOPs. When this flag is set to '0', the value of rounding_control is 0, and when this flag is set to '1', the value of rounding_control is 1. When vop_rounding_type is not present in the VOP header, the value of rounding_control is 0.

vop_reduced_resolution: This single bit flag signals whether the VOP is encoded in spatially reduced resolution or not. When vop_reduced_resolution is not transmitted, it takes a default value of zero. When this flag is set to '1', the VOP is encoded spatially reduced resolution and referred as Reduced Resolution VOP. Reduced Resolution VOP shall be decoded by the texture decoding process including upsampling of IDCT output, and motion compensation decoding process of Reduced Resolution VOP. In this case, the width of the encoded luminance component of the VOP in macroblocks is (video_object_layer_width+31)/32 and the the height of the encoded luminance component of the VOP in macroblocks is (video_object_layer_height+31)/32. The displayable part is top and left-aligned in the encoded VOP. This flag shall not be the '1' when the (video_object_layer_width+15)/16 is not the multiple of 2 or the (video_object_layer_width+15)/16 is not the multiple of 2. When this flag is set to "0" or this flag is not present, the VOP is encoded in normal spatial resolution and shall be decoded by the normal decoding process.

vop_width: This is a 13-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 13-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 13-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 13-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.

background_composition: This flag only occurs when scalability flag has a value of "1. This flag is used in conjunction with enhancement_type flag. If enhancement_type is "1", hierarchy_type is "1", and this flag is "1", background composition specified in subclause 8.1 is performed. If enhancement type is "1", hierarchy_type is "1" and this flag is "0", any method can be used to make a background for the enhancement layer.

When hierarchy_type is '0', video_object_layer_shape is "binary" (object based spatial scalability), enhancement_type is "1", and background_compositin flag is "1", background composition specified in subclause 8.4 is performed and when enhancement_type is "1" and this flag is '0', any method can be used to make a background for the enhancement layer.

change_conv_ratio_disable: This is a 1-bit flag which when set to '1' indicates that conv_ratio is not sent at the macroblock layer and is assumed to be 1 for all the macoblocks of the VOP. When set to '0', the conv_ratio is coded at macroblock layer.

vop_constant_alpha: This bit is used to indicate the presence of vop_constant_alpha_value. When this is set to one, vop_consant_alpha_value is included in the bitstream.

vop_constant_alpha_value: This is an unsigned integer which indicates the scale factor to be applied as a post processing phase of binary or grayscale shape decoding. See subclause 7.5.4.2.

intra_dc_vlc_thr: This is a 3-bit code allows a mechanism to switch between two VLC's for coding of Intra DC coefficients as per Table 6-21.

Table 6-21 -- Meaningof intra_dc_vlc_thr

index	meaning of intra_dc_vlc_thr	code
0	Use Intra DC VLC for entire VOP	000
1	Switch to Intra AC VLC at running Qp >=13	001
2	Switch to Intra AC VLC at running Qp >=15	010
3	Switch to Intra AC VLC at running Qp >=17	011
4	Switch to Intra AC VLC at running Qp >=19	100
5	Switch to Intra AC VLC at running Qp >=21	101
6	Switch to Intra AC VLC at running Qp >=23	110
7	Use Intra AC VLC for entire VOP	111

Where running Qp is defined as the DCT quantisation parameter for luminance and chrominance used for immediately previous coded macroblock, except for the first coded macroblock in a VOP or a video packet. At the first coded macroblock in a VOP or a video packet, the running Qp is defined as the quantisation parameter value for the current macroblock.

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

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

sprite_transmit_mode: This is a 2-bit code which signals the transmission mode of the sprite object. At video object layer initialization, the code is set to "piece" mode. When all object and quality update pieces are sent for the entire video object layer, the code is set to the "stop" mode. When an object piece is sent, the code is set to "piece" mode. When an update piece is being sent, the code is set to the "update" mode. When all sprite object pieces andquality update pieces for the current VOP are sent, the code is set to "pause" mode. Table 6-22 shows the different sprite transmit modes.

Table 6-22 -- Meaning ofsprite transmit modes

code	sprite_transmit_mode
00	stop
01	piece
10	update
11	pause

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/[illegible]
Single user licence only. Copying and networking prohibited.

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

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

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

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

ref_select_code: This is a 2-bit unsigned integer which specifies prediction reference choices for P- and B-VOPs in enhancement layer with respect to decoded reference layer identified by ref_layer_id. The meaning of allowed values is specified in Table 7-13 and Table 7-14.

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

macroblock_number: This is a variable length code with length between 1 and 14 bits. It identifies the macroblock number within a VOP. The number of the top-left macroblock in a VOP shall be zero. The macroblock number increases from left to right and from top to bottom. The actual length of the code depends on the total number of macroblocks in the VOP calculated according to Table 6-23, the code itself is simply a binary representation of the macroblock number. If reduced_resolution_vop_enable is equal to one, the length of macroblock_number code shall be determined by as if the total number of macroblocks in a VOP in Table 6-2-3 be equal to ((video_object_layer_width+15)/16) * ((video_object_layer_height+15)/16).

Table 6-23 -- Length of macroblock_number code

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

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

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

use_intra_dc_vlc: The value of this internal flag is set to 1 when the values of intra_dc_thr and the DCT quantiser for luminance and chrominance indicate the usage of the intra DC VLCs shown in Table B-13 - Table B-15 for the decoding of intra DC coefficients. Otherwise, the value of this flag is set to 0.

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

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

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

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

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

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

6.3.5.1 Definition of DCECS variable values

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

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

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

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

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/[illegible]
ANSI Store order X[illegible] Downloaded [illegible] PM ET
Single user license only. Copying and networking prohibited.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6.3.5.2 Video Plane with Short Header

video_plane_with_short_header() – This data structure contains a video plane using an abbreviated header format. Certain values of parameters shall have pre-defined and fixed values for any video_plane_with_short_header, due to the limited capability of signaling information in the short header format. These parameters having fixed values are shown is Table 6-24.

Table 6-24 -- Fixed Settings for video_plane_with_short_header()

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

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

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

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

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

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

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA V. AGAWANE
Single user license only. Copying and networking prohibited.

source_format: This is an indication of the width and height of the rectangular videc plane represented by the video_plane_with_short_header. The meaning of this fielc is shown in Table 6-25. Each of these source formats has the same VOP time increment resolution which is equal to 30000/1001 (approximately 29.97) Hz and the same width:height pixel aspect ratio (288/3):(352/4), which equals 12:11 in relatively prime numbers and which defines a CIF picture as having a width:height picture aspect ratio of 4:3.

Table 6-25 -- Parameters Defined by source_format Field

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

picture_coding_type: This bit indicates the vop_coding_type. When equal to zero, he vop_coding_type is "I", and when equal to one, the vop_coding_type is "P".

four_reservec_zero_bits: This is a four-bit fied containng bits which are reserved for future use and equal to zero.

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

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

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

num_gobs_in_vop: This is the number of GOBs in the VOP. This parameter is derived from the source_format as shown in Table 6-25.

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

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

gob_number: This is a five-bit number which indicates which GOB is being processed in the VOP. Its value may either be read following a gob_resync_marker or may be inferred from the progress of macroblock decoding. All GOBs shall appear in the bitstream of each video_plane_with_short_header(), and the GOBs shall appear in a

© ISO/IEC 2001 – All rights reserved
Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA V. AGAWANE
ANSI Store order [illegible] Downloaded: 7/18/2007 [illegible] PM ET
Single user license only. Copying and networking prohibited.

strictly increasing order in the bitstream. In other words, if a gob_number is read from the bitstream after a gob_resync_marker, its value must be the same as the value that would have been inferred in the absence of the gob_resync_marker.

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

num_macroblocks_in_gob: This is the number of macroblocks in each group of blocks (GOB) unit. This parameter is derived from the source_format as shown in Table 6-25. The count of stuffing macroblocks is not included in this value.

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

6.3.5.3 Shape coding

bab_type: This is a variable length code between 1 and 7 bits. It indicates the coding mode used for the bab. There are seven bab_types as depicted in Table 6-26. The VLC tables used depend on the decoding context i.e. the bab_types of blocks already received. For I-VOPs, the context-switched VLC table of Table B-27 is used. For P-VOPs, B-VOPs, and S(GMC)-VOPs , the context switched table of Table B-28 is used.

Table 6-26 -- List of bab_types and usage

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

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

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

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

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA V. AGAWANE
Single user license only. Copying and networking prohibited.

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

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

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

enh_bab_type: This is a variable length code between 1 and 3 bits. It indicates the bab coding mode used in binary shape enhancement layer coding. There are four enh_bab_types as depicted in Table V2 - 3. The VLC tables used depend on the decoding context i.e. the bab_types of blocks in lower reference layer.

Table V2 - 3 -- List of enh_bab_types and usage

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

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

enh_binary_arithmetic_code() -- This is a binary arithmetic decoder representing the pixel values of the bab and bab type of Scan Interleaving (SI) method. This code may be generated by SI decoding method or inter cae depending on the enh_bab_type. When enh_bab_type==1, the first decoded value represents the bab type of SI decoding method ("0": transitional bab, "1": exceptional bab). And the other decoded values represent the pixel values of the bab. If the bab type of SI is "transitional bab", only transitional pixels in the coded-scan-lines are decoded. Otherwise, for "exceptional bab", all of the pixels in the coded-scan-lines are decoded. This binary value decoding relies on the knowledge of enh_bab_type_prob[], and enh_intra_v_prob[] and enh_intra_h_prob[]probability tables given in Annex B. When enh_bab_type==3, this binary arithmetic decoder represents the pixel values of the bab. This code is generated by inter cae. This binary value decoding relies on the knowledge of inter_prob[], probability tables given in Annex B.

6.3.5.4 Sprite coding

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

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

© ISO/IEC 2001 – All rights reserved
Licensed to UNIVERSITY OF TEXAS AT ARLINGTON [illegible]
Single user licence only. Copying and networking prohibited.

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

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

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

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

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

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

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

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

6.3.6 Macroblock related

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

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

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

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

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

Licensed to UNIVERSITY OF TEXAS AT [illegible]
Single user license only. Copying and networking prohibited.

dquant: This is a 2-bit code which specifies the change in the quantizer, quant, for I-, P-, and S(GMC)-VOPs. Table 6-27 lists the codes and the differential values they represent. The value of quant lies in range of 1 to $2^{quant\_precision}$-1; if the value of quant after adding dquant value is less than 1 or exceeds $2^{quant\_precision}$-1, it shall be correspondingly clipped to 1 and $2^{quant\_precision}$-1. If quant_pecision takes its default value of 5, the range of allowed values for quant is [1:31].

Table 6-27 -- dquant codes and corresponding values

dquant code	value
00	1
01	2
10	1
11	2

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

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

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

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

dbquant: This is a variable length code which specifies the change in quantizer for B-VOPs. Table 6-28 lists the codes and the differential values they represent. If the value of quant after adding dbquant value is less than 1 or exceeds $2^{quant\_precision}$-1, it shall be correspondingly clipped to 1 and $2^{quant\_precision}$-1. If quant_precision takes its default value of 5, the range of allowed values for the quanizer for B-VOP is [1:31].

Table 6-28 -- dbquant codes and corresponding values

dbquant code	value
10	2
0	0
11	2

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

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

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

coda_pb: This is a VLC indicating the coding status for P, S(GMC), or B apha macroblocks. The semantics are given in the table below (Table 6-29). When this VLC indicates that the alpha macroblock is all opaque, this means that all values are set to 255 (AlphaOpaqueValue).

Table 6-29 -- coda_pb codes and corresponding values

coda_pb	Meaning
1	apha residue all zero
01	apha macroblock all opaque
00	apha residue coded

6.3.6.1 MB Binary Shape Coding

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

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

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

conv_ratio: This defines the upsampling factor accordng to Table B-31 to be applied after decoding the current shape information

scan_type: This defines according to Table 6-30 whether the current bordered to be decoded bab and the eventual bordered motion compensated bab need to be transposed

Table 6-30 -- scan_type

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

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

enh_bab_type – This defines the coding type of the current bab in the enhancement layer according to Table V2 - 30 and Table V2 - 31 for P-VOP and B-VOP coding, respectively.

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

6.3.6.2 Motion vector

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

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

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

r_size = vop_fcode_forward - 1 or r_size = vop_fcode_backward - 1

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

r_size = vop_fcode_forward - 1 or r_size = vop_fcode_backward - 1

6.3.6.3 Interlaced Information

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

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

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

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

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

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

6.3.7 Block related

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

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

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

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

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

6.3.7.1 Alpha block related

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

6.3.8 Still texture object

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

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

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

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

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

number_of_tiles: This is a 16-bit of unsigned integer specifying the number of tiles encoded in this bitstream. When texture_object_layer_shape=='00', the value is derived from CEIL(texture_object_layer_width÷tile_width) * CEIL(texture_object_layer_height÷tile_height), where CEIL() rounds up to the nearest integer. When texture_object_layer_shape=='01', the value is derived from CEIL(object_width÷tile_width) * CEIL(object_height÷tile_height).

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

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

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

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

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

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

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

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

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

Table 6-31 – Wavelet type

wavelet_filter_type	Meaning
0	integer
1	Double float

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

Table 6-32 – Wavelet downloading flag

wavelet_download	meaning
0	default filters
1	specifed in bitstream

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

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

scan_direction: This field indicates the scan order of AC coefficients. In single-quant and multi-quant mode, if this flag is '0', then the coefficients are scanned in the tree-depth fashion. If it is '1', then they are scanned in the subband by subband fashion. In bilevel_quant mode, if the flag is '0', then they are scanned in bitplane by bitplane fashion. Within each bitplane, they are scanned in a subband by subband fashion. If it is "1", they are scanned from the low wavelet decomposition layer to high wavelet decomposition layer. Within each wavelet decomposition layer, they are scanned from most significant bitplane down to the least significant bitplane.

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

texture_object_layer_shape: This is a 2-bit integer defined in Table 6-33. It identifies the shape type of a texture object layer.

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA V. AGAVANE
ANSI Store order X037445 Downloaded: 7/18/2007 3:56:21 PM ET
Single user license only. Copying and networking prohibited.

Table 6-33 -- Texture Object Layer Shape type

texture_object_layer_shape	Meaning
00	rectangular
01	binary
10	reserved
11	reserved

quantisation_type: This field indicates the type of quantisation as shown in Table 6-34.

Table 6-34 -- The quantisation type

quantisation_type	Code
single quantizer	01
multi quantizer	10
bi-level quantizer	11

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

use_default_spatial_scalability: This field indicates how the spatial scalability levels are formed. If its value is one, then default spatial scalability is used, starting from (½)^(spatial_scalability_levels-1)-th of the full resolution up to the full resolution, where ^ is a power operation. f its value is zero, the spatial scalability is specified by wavelet_layer_index described below.

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

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

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

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

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

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

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

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

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

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA V. ABAWANE
Single user license only. Copying and networking prohibited.

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

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

texture_tile_type: This is a 2-bit integer defined in Table V2 - 4. It identifies the shape type of a texture object in a tile when texture_object_layer_shape is "01".

Table V2 - 4 – Texture Tile Shape

texture_tile_type	Meaning
00	Forbidden
01	opaque tile
10	Boundary tile
11	Transparent tile

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

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

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

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

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

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

6.3.8.1 Texture Layer Decoding

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

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

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

arith_decode_highbands_bb(): This is an arithmetic decoder for decoding the quantized coefficient values of the higher bands (all bands except DC band) within a single band. The bitstream is generated by an adaptive arithmetic encoder. The arithmetic decoding relies on the initialization of the uniform probability distribution models described in subclause B.2.2. This decoder uses arithmetic. It also uses an adaptive probability model based on

© ISO/IEC 2001 – All rights reserved
Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA V. AGAWANE
ANSI Store order [illegible] Downloaded: 7/18/2007 [illegible] PM ET
Single user license only. Copying and networking prohibited.

the frequency counts of the previously decoded symbols. The maximum range (or precision) specified is (2^16) - 1 (16 bits). The maximum frequency count for the magnitude and residual models is 127, and for all other models it is 127.

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

texture_snr_layer_start_code: The texture_snr_layer_start_code is a string of 32 bits. The 32 bits are '0000 0000 0000 1000 0000 0001 1100 0000' in binary. The texture_snr_layer_start_code marks the start of a new snr layer.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

where functionget_param() is defined as

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

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

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

band_max_byte: This field defines one byte of the maximum value of the DC band. The parameter band_max_value is decoded using function get_param(). The number of maximum bitplanes for DC band is derived from CEIL(log$_2$(band_max_value+1))

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

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

6.3.8.2 Shape Object decoding

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

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

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

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

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

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

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

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

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

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

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

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

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

bab_type: This is a variable length code of 1-2 bits. It indicates the coding mode used for the bab. There are three bab_types as depicted in Table 6-35. The VLC tables used depend on the decoding context i.e. the bab_types of blocks already received.

Table 6-35 -- List of bab_types and usage

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

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

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

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/VOJA V. ADAMANE
Single user license only. Copying and networking prohibited.

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

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

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

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

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

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

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

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

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

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

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

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

6.3.9 Mesh object

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

6.3.9.1 Mesh object plane

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

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

6.3.9.2 Mesh geometry

mesh_type_code: This is a 2-bit integer defined in Table 6-36. It indicates the type of initial mesh geometry to be decoded.

Table 6-36 -- Mesh type code

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

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

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

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

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

triangle_split_code: This is a 2-bit integer defined in Table 6-37. It specifies how rectangles of a uniform mesh are split to form triangles.

Table 6-37 -- Specification of the triangulation type

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

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

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

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

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

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

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

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

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

6.3.9.3 Mesh motion

motion_range_code: This is a 3-bit integer defined in Table 6-38. It specifies the dynamic range of motion vectors in half pel units.

Table 6-38 -- motion range code

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

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

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

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

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

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

6.3.10 FBA object

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

6.3.10.1 FBA object plane header

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

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

fba_object_mask: This is a 2-bit integer defined in Table 6-40. It indicates whether FBA and BAP data are present in the FBA_frame.

Table 6-40 – FBA object mask

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

6.3.10.2 FBA object plane data

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

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

fap_mask_type: This is a 2-bit integer. It indicates if the group mask will be present for the specified fap group, or if the complete faps will be present; its meaning is descibed in Table 6-42. In the case the type is '10' the '0' bit in the group mask indicates interpolate fap.

Table 6-42 – fap mask type

mask type	Meaning
00	no mask nor fap
01	group mask
10	group mask'
11	fap

fap_group_mask[group_number]: This is a variable length bit entity that indicates, for a particular group_number which fap is represented in the bitstream. The value is interpreted as a mask of 1-bit fields. A 1-bit field in the mask that is set to '1' indicates that the corresponding fap is present in the bitstream. When that 1-bit field is set to '0' it indicates that the fap is not present in the bitstream. The number of bits used for the fap_group_mask depends on the group_number, and is given in Table 6-43.

Licensed to UNIVERSITY OF TEXAS AT [illegible]
Single user license only. Copying and networking prohibited.

Table 6-43 -- fap group mask bits

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

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

Table 6-44 -- NFAP definition

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

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

fba_object_coding_type: This is a 1-bit integer indicating which coding method is used. Its meaning is described in Table 6-39.

Table 6-39 -- fba_object_coding_type

type value	Meaning
0	predictive coding
1	DCT

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

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

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

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

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

bap_pred_scale[0-31]=
{ 0, 1, 2, 3, 5, 7, 8, 11, 14, 17, 20, 23, 27, 31, 35, 39, 43, 47, 52, 57, 62, 67, 72, 77, 82, 88, 94, 100, 106, 113, 120, 127}

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

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

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

bap_mask_type	Meaning
00	No BAPs
01	BAPs transmitted in groups
10	reserved
11	BAPs transmitted individually

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

group number	group name	No. of. bits
1	Pelvis	3
2	Left leg1	4
3	Right leg1	4
4	Left leg2	6
5	Right leg2	6
6	Left arm1	5
7	Right arm1	5
8	Left arm2	7
9	Right arm2	7
10	Spine1	12
11	Spine2	15
12	Spine3	18
13	Spine4	18
14	Spine5	12
15	Left hand1	16

Licensed to UNIVERSITY OF TEXAS AT [illegible]
Single user licence only. Copying and networking prohibited.

16	Right hand1	16
17	Left hand2	13
18	Right hand2	13
19	Global positioning	6
20	Extension1	22
21	Extension2	22
22	Extension3	22
23	Extension4	22
24	Extension5	22

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

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

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

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

6.3.10.3 Temporal Header

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

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

time_code: This is a 18-bit integer containing the following: time_code_hours, time_code_minutes, marker_bit and time_code_seconds as shown in Table 6-41. The parameters correspond to those defined in the IEC standard publication 461 for "time and control codes for video tape recorders". The time code specifies the modulo part (i.e. the full second units) of the time base for the current object plane.

Table 6-41 -- Meaning of time_code

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

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

6.3.10.4 Decode frame rate and frame skip

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

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

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

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

6.3.10.5 Decode new minmax

i_new_max[j]: This is a 5-bit unsigned integer used to scale the maximum value of the arithmetic decoder used in the I frame.

i_new_min[j]: This is a 5-bit unsigned integer used to scale the minimum value of the arithmetic decoder used in the I frame.

p_new_max[j]: This is a 5-bit unsigned integer used to scale the maximum value of the arithmetic decoder used in the P frame.

p_new_min[j]: This is a 5-bit unsigned integer used to scale the minimum value of the arithmetic decoder used in the P frame.

6.3.10.6 Decode viseme and expression

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

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

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

6.3.10.7 Decode viseme_segment and expression_segment

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

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

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

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

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

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

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

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/ROOJA Y. ADAMANE
ANSI Store order X037445. Downloaded 7/18/2007 3:56:21 PM ET
Single user license only. Copying and networking prohibited.

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

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

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

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

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

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

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

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

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

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

6.3.10.8 Decode i_dc, p_dc, and ac

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

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

For DC coefficients, the quantisation stepsize is obtained as follows:

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

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

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

Licensed to UNIVERSITY OF TEXAS AT [illegible]
Single user license only. Copying and networking prohibited.

ac_q[i][nex]: This is a quantized AC coefficients of a segment of FAPi. For AC coefficients, the quantisation stepsize is three times larger than the DC quantisation sepsize and is obtained as follows:

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

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

6.3.10.9 Decode bap min max

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

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

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

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

6.3.11 3D Mesh Object

6.3.11.1 3D_Mesh_Object

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

6.3.11.2 3D_Mesh_Object_Layer

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

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

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

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

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

6.3.11.3 3DMesh_Object_Base_Layer

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

Table V2 - 5 -- Definition of partition type information

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

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

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

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

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

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

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

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

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

6.3.11.4 3DMesh_Object_Header

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

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

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

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

6.3.11.5 coord_header

coord_binding: This 2 bit unsigned integer indicates the binding of vertex coordinates to the 3D mesh. Table V2 - 6 shows the admissible values for **coord_binding**.

Table V2 - 6 -- Admissible values for coord_binding

coord_binding	binding
00	forbidden
01	bound_per_vertex
10	forbidden
11	forbidden

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

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

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

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

coord_pred_type: This 2-bit unsigned integer indicates the type of prediction used to reconstruct the vertex coordinates of the mesh. Table V2 - 7 shows the admissible values for coord_pred_type.

Table V2 - 7 -- Admissible values for coord_pred_type

coord_pred_type	prediction type
00	no_prediction
01	forbidden
10	parallelogram_prediction
11	reserved

coord_nlambda: This 2-bit unsigned integer indicates the number of ancestors used to predict geometry. The only admissible value of **coord_nlambda** is 3. Table V2 - 8 shows the admissible values as a function of **coord_pred_type.**

Table V2 - 8 -- Admissible values for coord_nlambda as a function of coord_prediction type

coord_pred_type	coord_nlambda
00	not coded
10	3

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

6.3.11.6 normal_header

normal_binding: This 2 bit unsigned integer indicates the binding of normals to the 3D mesh. The admissible values are described in Table V2 - 9.

Table V2 - 9 -- Admissible values for normal_binding

normal_binding	binding
00	not_bound
01	bound_per_vertex
10	bound_per_face
11	bound_per_corner

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

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

normal_pred_type: This 2-bit unsigned integer indicates how normal values are predicted. Table V2 - 10 shows the admissible values, and Table V2 - 11 shows admissible values as a function of normal_binding.

Table V2 - 10 -- Admissible values for normal_pred_type

normal_pred_type	prediction type
00	no_prediction
01	tree_prediction
10	parallelogram_prediction
11	reserved

Table V2 - 11 -- Admissible combinations of normal_binding and normal_pred_type

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

normal_nlambda: This 2-bit unsigned integer indicates the number of ancestors used to predict normals. Admissible values of **normal_nlambda** are 1, 2, and 3. Table V2 - 12 shows admissible values as a function of **normal_pred_type**.

Table V2 - 12 -- Admissible values for normal_nlambda as a function of normal_prediction type

normal_pred_type	*normal_nlambda*
no_prediction	not coded
tree_prediction	1, 2, 3
parallelogram_prediction	3

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

6.3.11.7 color_header

color_binding: This 2 bit unsigned integer indicates the binding of colors to the 3D mesh. Table V2 - 13 shows the admissible values.

Table V2 - 13 -- Admissible values for color_binding

color_binding	Binding
00	not_bound
01	bound_per_vertex
10	bound_per_face
11	bound_per_corner

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

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

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

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

color_pred_type: This 2-bit unsigned integer indicates how colors are predicted. Table V2 - 14 shows the admissible values, and Table V2 - 15 shows admissible values as a function of color_binding.

Table V2 - 14 -- Admissible values for color_pred_type

color_pred_type	prediction type
00	no_prediction
01	tree_prediction
10	parallelogram_prediction
11	reserved

Table V2 - 15 -- Admissible combinations of color_binding and color_pred_type

color_binding	color_pred_type
not_bound	not coded
bound_per_vertex	no_predicion, parallelogram_prediction
bound_per_face	no_predicion, tree_prediction
bound_per_corner	no_predicion, tree_prediction

color_nlambda: This 2-bit unsigned integer indicates the number of ancestos used to predict normals. Admissible values of **color_nlambda** are 1, 2, and 3. Table V2 - 16 shows admissible values as a function of **normal_pred_type**.

Table V2 - 16 -- Admissible values for color_nlambda as a function of color_prediction type

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

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

6.3.11.8 texCoord_header

texCoord_binding: This 2 bit unsigned integer indicates the binding of texture coordinates to the 3D mesh. Table V2 - 17 describes the admissible values.

Table V2 - 17 -- Admissible values for texCoord_binding

texCoord_binding	Binding
00	not_bound
01	bound_per_vertex
10	forbidden
11	bound_per_corner

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

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

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

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

texCoord_pred_type: This 2-bit unsigned integer indicates how colors are predicted. Table V2 - 18 shows the admissible values, and Table V2 - 19 shows admissible values as a function of texCoord_binding.

Table V2 - 18 -- Admissible values for texCoord_pred_type

texCoord_pred_type	prediction type
00	no_prediction
01	forbidden
10	parallelogram_prediction
11	reserved

Table V2 - 19 – Admissible combinations of texCoord_binding and texCoord_pred_type

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

texCoord_nlambda: This 2-bit unsigned integer indicates the number of ancestors used to predict normals. Admissible values of **texCoord_nlambda** are 1, 2, and 3. Table V2 - 20 shows admissible values as a function of **texCoord_pred_type**.

Table V2 - 20 – Admissible values for texCoord_nlambda as a function of texCoord_prediction type

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

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

6.3.11.9 ce_SNHC_header

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

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

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

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

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

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

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

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

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

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA V. [illegible]
Single user license only. Copying and networking prohibited.

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

6.3.11.10 connected component

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

6.3.11.11 vertex_graph

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

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

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

vg_loop_index: This unsigned integer indicates the index of run to which the current loop connects. Its unary representation (see Table V2 - 21) is arithmetic coded. If the variable openloops is equal to **vg_loop_index**, the trailing '1' in the unary representation is omitted.

Table V2 - 21 -- Unary representation of the vg_loop_index field

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

vg_run_length: This unsigned integer indicates the length of the current vertex run. Its unary representation (see Table V2 - 22) is arithmetic coded.

Table V2 - 22 -- Unary representation of the vg_run_length field

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

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

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

6.3.11.12 stitches

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

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

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

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

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

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

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

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

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

6.3.11.13 triangle_tree

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

tt_run_length: This unsigned integer indicates the length of the current triangle run. Its unary representation (see Table V2 - 23) is arithmetic coded.

Table V2 - 23 -- Unary representation of the tt_run_length field

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

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

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

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

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

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

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

6.3.11.14 triangle

td_orientation: This boolean value informs the decoder the traversal order of tt/td pair at a branch. This field is arithmetic coded. Table V2 - 24 shows the admissible values.

Table V2 - 24 -- Admissible values for td_orientation

td_orientation	traversal order
0	right branch first
1	left branch first

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

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

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

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

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

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

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

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

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

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

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

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

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA V. AGARWAL
Single user license only. Copying and networking prohibited.

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

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

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

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

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

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

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

6.3.11.15 3DMeshObject_Refinement_Layer

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

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

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

Table V2 - 25 -- Admissible values for connectivity_update

connectivity_update	**meaning**
00	not_updated
01	fs_update
10	reserved
11	reserved

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

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

stuffing_bit: This boolean value is always true.

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

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

6.3.11.15.1 pre_smoothing_parameters

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

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

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

6.3.11.15.2 post_smoothing_parameters

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

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

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

6.3.11.15.3 fs_pre_update

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

6.3.11.15.4 smoothing_constraints

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

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

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

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

6.3.12 Upstream message

6.3.12.1 upstream_message

upstream_message_type: This 3-bit value indicates the type of the upstream informaton as shown in Table V2 - 26. The meaning of each upstream type is as follows:

> video_newpred: This upstream message conveys the decoding status of the receiver (decoder) for the NEWPRED mode. The NEWPRED is the error resilience tool by selecting the reference picture of the inter-frame coding according to the error condition of the network. This upstream message shows whether the decoder decode the forward video data correctly or not. This message returns corresponding to the VOP or Video Packet of the forward video data. Which type of message is required for the encoder is indicated in requested_upstream_message_type in the VOL header of the downstream data. The decoder definitions of NEWPRED are described in subclauses 7.14 and E.1.6.
>
> SNHC_QoS: This upstream message conveys some information that reveals to the encoder (server) the performances of the decoder w.r.t. decoding and rendering 3D objects with different parameter settings, i.e. with a varying number of triangles, different screen coverages of the rendered objects and different rendering modes. Using this information, the encoder can restrict the 3D content to the capabilities of the decoder, using for instance mesh simplificaton and/or rendering mode selection.

Table V2 - 26 – Meaning of upstream_message_type

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

© ISO/IEC 2001 – All rights reserved
Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/DOUG R. ADAMS
ANSI Store order #X27445 Downloaded: 7/18/2007 2:38:21 PM ET
Single user license only. Copying and networking prohibited.

6.3.12.2 upstream_video_newpred

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

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

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

0: reliable
1: unreliable

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

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

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

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

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

6.3.12.3 upstream_SNHC_QoS

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

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

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

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

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/[illegible] V. ADAMANE
ANSI Store order #X37445_ Downloaded: 7/18/2007 3:58:21 PM ET
Single user license only. Copying and networking prohibited.

Table V2 - 27 -- Meaning of rendering_mode_type

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

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

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

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

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

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

7 The visual decoding process

This clause specifies the decoding process that the decoder shall perform to recover visual data from the coded bitstream. As shown in Figure 7-1, the visual decoding process includes several decoding processes such as shape-motion-texture decoding, still texture decoding, mesh decoding, and face decoding processes. After decoding the coded bitstream, it is then sent to the compositor to integrate various visual objects.

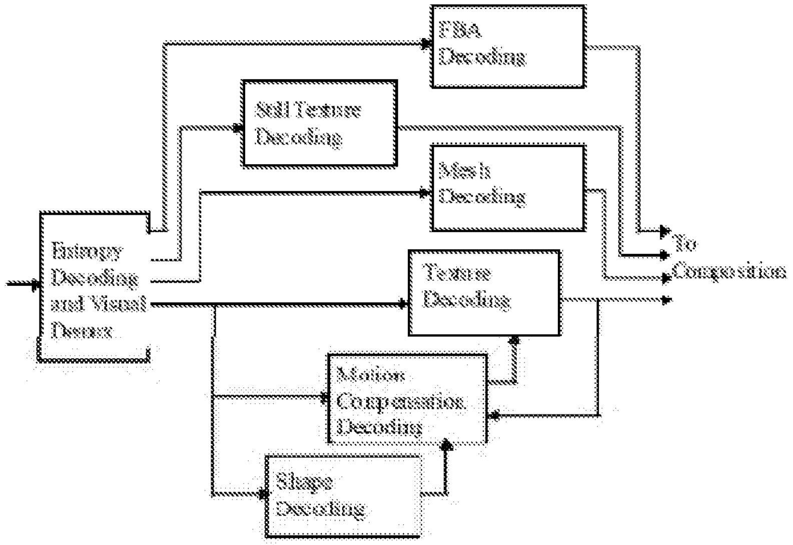


Figure 7-1 – A high level view of basic visual decoding;
specialized decoding such as scalable, sprite and error resilient decoding are not shown

© ISO/IEC 2001 – All rights reserved
Licensed to UNIVERSITY OF TEXAS AT ARLINGTON [illegible]
Single user licence only. Copying and networking prohibited.

In subclauses 7.1 through 7.9 the VOP decoding process is specified in which shape, motion, texture decoding processes are the major contents. The still texture object decoding is described in subclauses 7.10. Subclause 7.11 includes the mesh decoding process, and subclause 7.12 features the face object decoding process. The output of the decoding process is explained in subclause 7.13.

7.1 Video decoding process

This subclause specifies the decoding process that a decoder shall perform to recover VOP data from the coded video bitstream.

With the exception of the Inverse Discrete Cosine Transform (IDCT) the decoding process is defined such that all decoders shall produce numerically identical results. Any decoding process that produces identical results to the process described here, by definition, complies with this part of ISO/IEC 14496.

The IDCT is defined statistically such that different implementations for this function are allowed. The IDCT specification is given in annex A.

Figure 7-2 is a diagram of the Video Decoding Process without any scalability feature. The diagram is simplified for clarity. The same decoding scheme is applied when decoding all the VOPs of a given session.

NOTE Throughout this part of ISO/IEC 14496 two dimensional arrays are represented as *name[q][p]* where 'q' is the index in the vertical dimension and 'p' the index in the horizontal dimension.

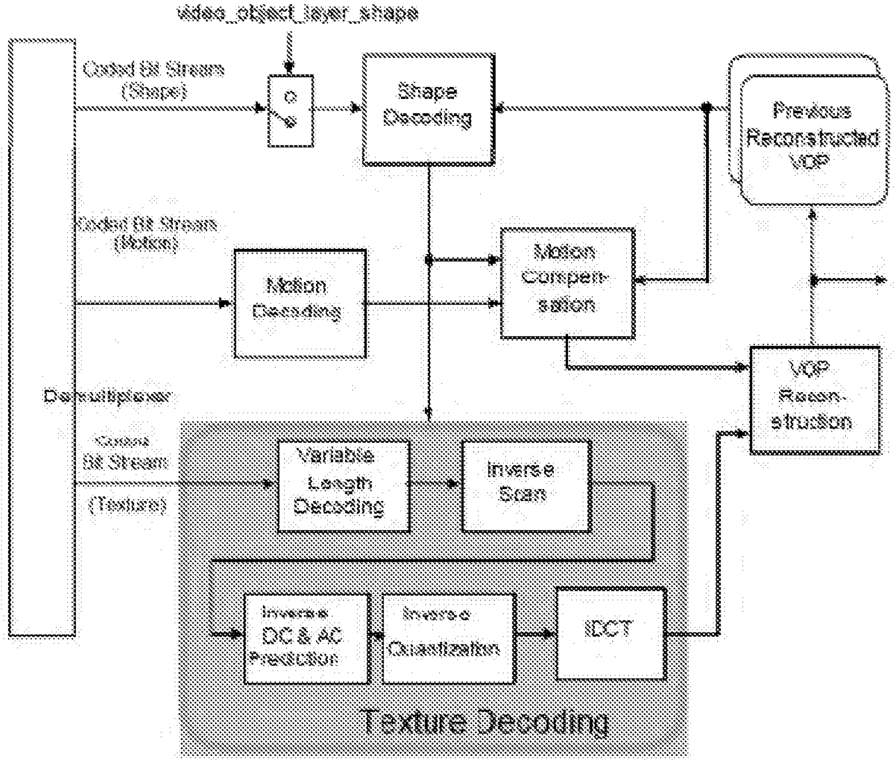


Figure 7-2 -- Simplified Video Decoding Process

The decoder is mainly composed of three parts: shape decoder, motion decoder and texture decoder. The reconstructed VOP is obtained by combining the decoded shape, texture and motion information.

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA [illegible]
Single user licence only. Copying and networking prohibited.

7.2 Higher syntactic structures

The various parameters and flags in the bitstream for VideoObjectLayer(), Group_of_VideoObjectPlane(), VideoObjectPlane(), video_plane_with_short_header(), macroblock() and block(), as well as other syntactic structures related to them shall be interpreted as 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. In case the bitstream being decoded contains B-VOPs, reordering of VOPs may be needed as discussed in subclause 6.1.3.7.

7.3 VOP reconstruction

The luminance and chrominance values of a VOP from the decoded texture and motion information are reconstructed as follows:

1. In case of INTRA macroblocks, the luminance and chrominance values f[y][x] from the decoded texture data form the luminance and chrominance values of the VOP: d[y][x] = f[y][x].
2. In case of INTER macroblocks, first the prediction values p[y][x] are calculated using the decoded motion vector information and the texture information of the respective reference VOPs. Then, the decoded texture data f[y][x] is added to the prediction values, resulting in the final luminance and chrominance values of the VOP: d[y][x] = p[y][x] + f[y][x]
3. Finally, the calculated luminance and chrominance values of the reconstructed VOP are saturated so that

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

7.4 Texture decoding

This subclause describes the process used to decode the texture information of a VOP. The process of video texture decoding is given in Figure 7-3.

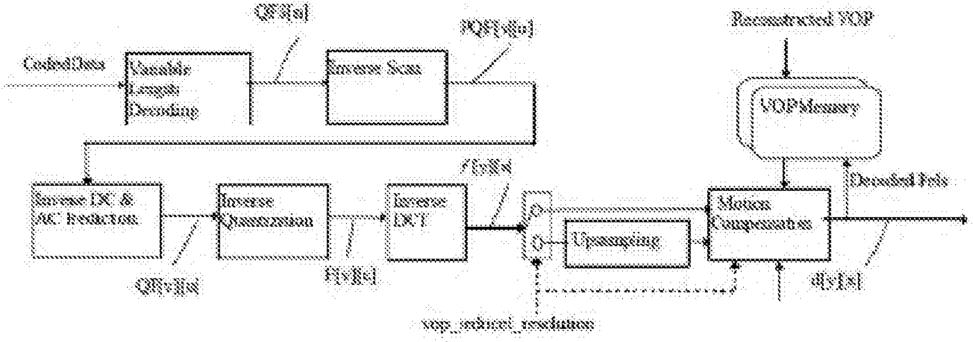


Figure 7-3 -- Video Texture Decoding Process

NOTE If flag 'sadct_disable' in the header of VideoObjectLayer() is set to '0', some standard texture decoding techniques from version 1 are modified. In 8x8-blocks with at least one transparent and opaque pel, standard inverse DCT (clause 7.4.8) is replaced by inverse shape adaptive DCT (SA-DCT). The number of SA-DCT coefficients in such a block is equal to the number of opaque pels. Only after BAB decoding, it is known to which 8x8-blocks inverse SA-DCT is applied. The usage of inverse SA-DCT in such blocks implies modifications in the processes "Inverse Scan" (clause 7.3.2), "Inverse AC Prediction" (clause 7.3.3.3) and "Inverse DCT" (clause 7.3.5) whereas the processing in the remaining blocks of Figure 7-3 keeps unmodified. The processing of inverse SA-DCT additionally depends on 'mb_type', i.e.: different types of SA-DCT processing are used in intra- and inter-coded blocks (see clause 7.3.5).

7.4.1 Variable length decoding

This subclause explains the decoding process. Subclause 7.4.1.1 specifies the process used for the DC coefficients (n=0) in an intra coded block. (n is the index of the coefficient in the appropriate zigzag scan order). Subclause 7.4.1.2 specifies the decoding process for all other coefficients; AC coefficients ($n \neq 0$) and DC coefficients in non-intra coded blocks.

7.4.1.1 DC coefficients decoding in intra blocks

Differential DC coefficients in blocks in intra macroblocks are encoded as variable length code denoting dct_dc_size as defined in Table B-13 and Table B-14 in annex B, and a fixed length code dct_dc_differential (Table B-15). The dct_dc_size categorizes the dc coefficients according to their "size". For each category additional bits are appended to the dct_dc_size code to uniquely identify which difference in that category actually occurred (Table B-15). This is done by appending a fixed length code, dct_dc_differential, of dct_dc_size bits. The final value of the decoded dc coefficient is the sum of this differential dc value and the predicted value.

When short_video_header is 1, the dc coefficient of an intra block is not coded differentially. It is instead transmitted as a fixed length unsigned integer code of size 8 bits, unless this integer has the value 255. The values 0 and 128 shall not be used – they are reserved. If the integer value is 255, this is interpreted as a signaled value of 128.

7.4.1.2 Other coefficients

The ac coefficients are obtained by decoding the variable length codes to produce EVENTs. An EVENT is a combination of a last non-zero coefficient indication (LAST; "0": there are more nonzero coefficients in this block, "1": this is the last nonzero coefficient in this block), the number of successive zeros preceding the coded coefficient (RUN), and the non-zero value of the coded coefficient (LEVEL).

When short_video_header is 1, the most commonly occurring EVENTS are coded with the variable length codes given in Table B-17 (for all coefficients other than intra DC whether in intra or inter blocks). The last bit "s" denotes the sign of level, "0" for positive and "1" for negative.

When short_video_header is 0, the variable length code table is different for intra blocks and inter blocks. The most commonly occurring EVENTs for the luminance and chrominance components of intra blocks in this case are decoded by referring to the intra columns of Table B-23 when reversible_vlc is set to '1' in I-, P-, or S(GMC)-VOPs, and by referring to Table B-16, otherwise. The most commonly occurring EVENTs for the luminance and chrominance components of inter blocks in this case are decoded by referring to the inter columns of Table B-23 when reversible_vlc is set to '1' in I-, P-, or S(GMC)-VOPs, and by referring to Table B-17, otherwise. The last bit "s" denotes the sign of level, "0" for positive and "1" for negative. The combinations of (LAST, RUN, LEVEL) not represented in these tables are decoded as described in subclause 7.4.1.3.

7.4.1.3 Escape code

Many possible EVENTS have no variable length code to represent them. In order to encode these statistically rare combinations an Escape Coding method is used. The escape codes of DCT coefficients are encoded in five modes. The first three of these modes are used when short_video_header is 0 and in the case that the reversible VLC tables are not used, and the fourth is used when short_video_header is 1. In the case that the reversible VLC tables are used, the fifth escape coding method as in Table B-23 is used. Use of escape sequence of the reversible VLC (Table B-24 and Table B-25) for encoding the combinations listed in Table B-23 is prohibited. Their decoding process is specified below.

Type 1: ESC is followed by "0", and the code following ESC + "0" is decoded as a variable length code using the standard Tcoef VLC codes given in Table B-16 and Table B-17, but the values of LEVEL are modified following decoding to give the restored value LEVEL^S, as follows:

$$\text{LEVEL}^S = \text{sign}(\text{LEVEL}^+) \times [\, \text{abs}(\text{LEVEL}^+) + \text{LMAX} \,]$$

where LEVEL^+ is the value after variable length decoding and LMAX is obtained from Table B-19 and Table B-20 as a function of the decoded values of RUN and LAST.

Type 2: ESC is followed by "10", and the code following ESC + "10" is decoded as a variable length code using the standard Tcoef VLC codes given in Table B-16 and Table B-17, but the values of RUN are modified following decoding to give the restored value RUN^S, as follows:

$$RUN^S = RUN^+ + (RMAX + 1)$$

where RUN^+ is the value after variable length decoding. RMAX is obtained from Table B-21 and Table B-22 as a function of the decoded values of LEVEL and LAST.

Type 3: ESC is followed by "11", and the code following ESC + "11" is decoded as fixec length codes. This type of escape codes are represented by 1-bit LAST, 6-bit RUN and 12-bit LEVEL. A marker bit is inserted before and after the 12-bit-LEVEL in order to avoid the resync_marker emulation. Use of this escape sequence for encoding the combinations listed in Table B-16 and Table B-17 is prohibited. The codes for RUN and LEVEL are given in Table B-18 a and b.

Type 4: The fourth type of escape code is used if and only if short_video_header is 1. In this case, the 15 bits following ESC are decoded as fixed length codes represented by 1-bit LAST, 6-bit RUN and 8-bit LEVEL. The values 0000 0000 and 1000 000 for LEVEL are not used (they are reserved). The codes for RUN and LEVEL are given in Table B-18 a and c.

7.4.1.4 Intra dc coefficient decoding for the case of switched vlc encoding

At the VOP layer, using quantizer value as the threshold, a 3 bit code (intra_dc_vlc_thr) allows switching between 2 VLCs (DC Intra VLC and AC Intra VLC) when decoding DC coefficients of Intra macroblocks, see Table 6-21.

NOTE When the intra AC VLC is turned on, Intra DC coefficients are not handled separately any more, but treated the same as all other coefficients. That means that a zero Intra DC coefficient will not be coded but will simply increase the run for the following AC coefficients. The definitions of mcbpc and cbpy in subclause 6.3.8 are changed accordingly.

7.4.2 Inverse scan

This subclause specifies the way in which the one dimensional data, QFS[n] is converted into a two-dimensional array of coefficients denoted by PQF[v][u] where u and v both lie in the range of 0 to 7. Let the data at the output of the variable length decoder be denoted by QFS[n] where n is in the range of 0 to 63. Three scan patterns are defined as shown in Figure 7-4. The scan that shall be used is determined by the following method. For intra blocks, if acpred_flag=0, zigzag scan is selected for all blocks in a macroblock. Otherwise, DC prediction direction is used to select a scan on block basis. For instance, if the DC prediction refers to the horizontally adjacent block, alternate-vertical scan is selected for the current block. Otherwise (for DC prediction referring to vertically adjacent block), alternate-horizontal scan is used for the current block. For all other blocks, the 8x8 blocks of transform coefficients are scanned in the "zigzag" scanning direction.

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

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

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

Figure 7-4 -- (a) Alternate-Horizontal scan (b) Alternate-Vertical scan (c) Zigzag scan

The inverse scan pattern from Figure 7-4 represent a conversion from the one-dimensional array 'QFS[n]' to two-dimensional array 'PQF[v][u]. This conversion can be expressed as follows where the arrays 'inv_scan_uscan_type][n]' and 'inv_scan_v[scan_type][n]' denote the u- and v-components of the selected inverse scan path from Figure 7-4:

```
for (n=0; n<64; n++)
    PQF[inv_scan_v[scan_type][n]][inv_scan_u[scan_type][n]] = QFS[n];
```

If 'sadct_disable' is set to '0', this conversion is modified for those 8x8-blocks in which the number of opaque pels is less than 64.

```
if ((video_object_layer!="rectangular") && (sadct_disable==0) &&
  (opaque_pels<64)) {
  coeff_count=0;
  for (n=0; n<64; n++){
     PQF[inv_scan_v[scan_type][n]][inv_scan_u[scan_type][n]]=0;
     if (coeff_width[inv_scan_v[scan_type][n]] > inv_scan_u[scan_type][n]) {
        PQF[inv_scan_v[scan_type][n]][inv_scan_u[scan_type][n]] =
          QFS[coeff_count];
        coeff_count++;
     }
  }
}
```

NOTES

1 After finalization of modified inverse scan the maximal number of non-zero SA-DCT coefficients (see parameter 'coeff_count') equals to 'opaque_pels'. It is absolutely necessary to keep this constraint by setting PQF[v][u] to zero at positions where SA-DCT coefficients are not defined. A non-zero value at these undefined positions of PQF[v][u] would confuse subsequent AC prediction.

2 The permitted values of elements in arrays 'inv_scan_u[scan_type][n]', 'inv_scan_v [scan_type][n]' and 'coeff_width[v]' lie in a range from 0 to 7.

3 Apart from the usage of inverse SA-DCT in clause 7.3.5 and slight modifications in clause 7.3.3.3 (inverse AC prediction) the further processing of 'PQF[v][u]' is independent of the setting of flag 'sadct_disable' and keeps unmodified for 'sadct_disable=0'.

The above definition of the modified inverse scan requires an auxiliary array 'coeff_ width[v]' which has to be derived from the decoded BAB. It indicates the number of SA-DCT coefficients which are available in each particular row of PQF[v][u]. See Annex A (subclause A.3.1) for details.

7.4.3 DC and AC prediction for intra macroblocks

This subclause specifies the prediction process for decoding of coefficients. This prediction process is only carried out for intra-macroblocks (I-MBs) and when short_video_header is '0'. When short_video_header is '1' or the macroblock is not an I-MB, this prediction process is not performed.

7.4.3.1 DC and AC Prediction Direction

This adaptive selection of the DC and AC prediction direction is based on comparison of the horizontal and vertical DC gradients around the block to be decoded. Figure 7-5 shows the three blocks surrounding the block to be decoded. Block 'X', 'A', 'B' and 'C' respectively refer to the current block, the left block, the above-left block, and the block immediately above, as shown.

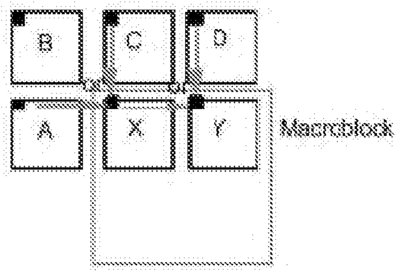


Figure 7-5 -- Previous neighboring blocks used in DC prediction

The inverse quantized DC values of the previously decoded blocks, F[0][0], are used to determine the direction of the DC and AC prediction as follows.

```
if ( $|F_A[0][0] - F_B[0][0]| < |F_B[0][0] - F_C[0][0]|$)
        predict from block C
else
        predict from block A
```

If any of the blocks A, B or C are outside of the VOP boundary, or the video packet boundary, or they do not belong to an intra coded macroblock, their F[0][0] values are assumed to take a value of $2^{bits\_per\_pixel+2}$ and are used to compute the prediction values.

7.4.3.2 Adaptive DC Coefficient Prediction

The adaptive DC prediction method involves selection of either the F[0][0] value of immediately previous block or that of the block immediately above it (in the previous row of blocks) depending on the prediction direction determined above.

```
if (predict from block C)
        $QF_X[0][0] = PQF_X[0][0] + F_C[0][0] // dc\_scaler$
else
        $QF_X[0][0] = PQF_X[0][0] + F_A[0][0] // dc\_scaler$
```

dc_scalar is defined in Table 7-1. This process is independently repeated for every block of a macroblock using the appropriate immediately horizontally adjacent block 'A' and immediately vertically adjacent block 'C'.

DC predictions are performed similarly for the luminance and each of the two chrominance components.

7.4.3.3 Adaptive ac coefficient prediction

This process is used when ac_pred_flag = '1', which indicates that AC prediction is performed when decoding the coefficients.

Either coefficients from the first row or the first column of a previous coded block are used to predict the co-sited coefficients of the current block. On a block basis, the best direction (from among horizontal and vertical directions) for DC coefficient prediction is also used to select the direction for AC coefficients prediction; thus, within a macroblock, for example, it becomes possible to predict each block independently from either the horizontally adjacent previous block or the vertically adjacent previous block. The AC coefficients prediction is illustrated in Figure 7-6.

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/ROD A. C. ADAMANE
ANSI Store order X007445 Downloaded 7/18/2007 2:38:21 PM ET
Single user license only. Copying and networking prohibited.

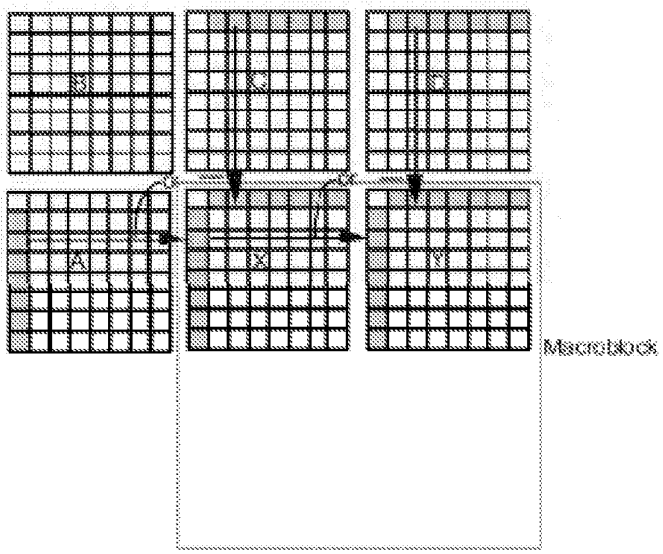


Figure 7-6 -- Previous neighboring blocks and coefficients used in AC prediction

To compensate for differences in the quantisation of previous horizontally adjacent or vertically adjacent blocks used in AC prediction of the current block, scaling of prediction coefficients becomes necessary. Thus the prediction is modified so that the predictor is scaled by the ratio of the current quantisation stepsize and the quantisation stepsize of the predictor block. The definition is given in the equations below.

If block 'A' was selected as the predictor for the block for which coefficient prediction is to be performed, calculate the first column of the quantized AC coefficients as follows.

$$QF_X[v][0] = PQF_X[v][0] + (QF_A[v][0] * QP_A) // QP_X \qquad v = 1 \text{ to } 7$$

If block 'C' was selected as the predictor for the block for which coefficient prediction is to be performed, calculate the first row of the quantized AC coefficients as follows.

$$QF_X[0][u] = PQF_X[0][u] + (QF_C[0][u] * QP_C) // QP_X \qquad u = 1 \text{ to } 7$$

If the prediction block (block 'A' or block 'C') is outside of the boundary of the VOP or video packet, then all the prediction coefficients of that block are assumed to be zero.

When AC coefficient prediction is used in conjunction with 'sadct_disable=0', the results of AC prediction has to be post-processed by shape parameters. This shape-adaptive post-processing takes care that those elements of 'QF[v][u]', which are not defined in SA-DCT and which should be zero therefore, are again set to zero after AC prediction. An auxiliary array 'coeff_ width[v]' which has to be derived from the decoded SAB (see subclause A.3.1 in Annex A for details) is used for this purpose: in column 'u=0' ('A'-predictor) or in row 'v=0' ('C'-predictor), all elements of 'QF[v][u]' with 'coeff_width[v] = 0' or 'u >= coeff_width[0]' are set to zero after AC prediction.

7.4.3.4 Saturation of QF[v][u]

The quantized coefficients resulting from the DC and AC Prediction are saturated to lie in the range [-2048, 2047]. Thus:

$$QF[v][u] = \begin{cases} 2047 & QF[v][u] > 2047 \\ QF[v][u] & -2048 \le QF[v][u] \le 2047 \\ -2048 & QF[v][u] < -2048 \end{cases}$$

7.4.4 Inverse quantisation

The two-dimensional array of coefficients, $QF[v][u]$, is inverse quantised to produce the reconstructed DCT coefficients. This process is essentially a multiplication by 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).

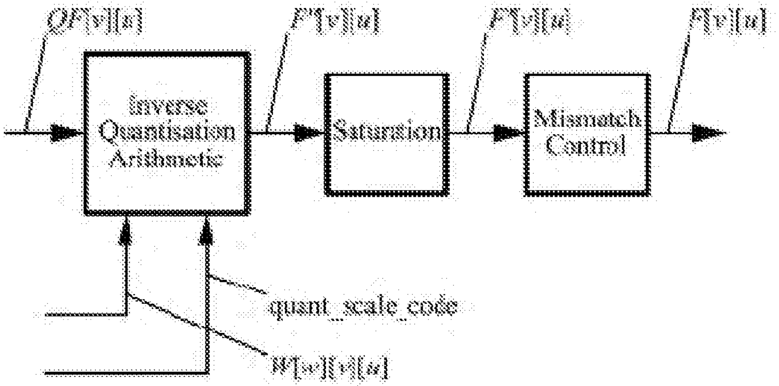


Figure 7-7 -- Inverse quantisation process

Figure 7-7 illustrates the overall inverse quantisation process. After the appropriate inverse quantisation arithmetic the resulting coefficients, $F''[v][u]$, are saturated to yield $F'[v][u]$ and then a mismatch control operation is performed to give the final reconstructed DCT coefficients, $F[v][u]$.

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.

7.4.4.1 First inverse quantisation method

This subclause specifies the first of the two inverse quantisation methods. The method described here is used when quant_type equals 1.

7.4.4.1.1 Intra dc coefficient

The DC coefficients of intra coded blocks shall be inverse quantised in a different manner to all other coefficients.

In intra blocks $F''[0][0]$ shall be obtained by multiplying $QF[0][0]$ by a constant multiplier.

The reconstructed DC values are computed as follows.

$F''[0][0] = dc\_scaler * QF[0][0]$

When short_video_header is 1, dc_scaler is 8, otherwise dc_scaler is defined in Table 7-1.

7.4.4.1.2 Other coefficients

All coefficients other than the DC coefficient of an intra block shall be inverse quantised as specified in this subclause. Two weighting matrices are used. One shall be used for intra macroblocks and the other for non-intra macroblocks. Each matrix has a default set of values which may be overwritten by down-loading a user defined matrix.

Let the weighting matrices be denoted by *W*[*w*][*v*][*u*] where *w* takes the values 0 to 1 indicating which of the matrices is being used. *W*[*0*][*v*][*u*] is for intra macroblocks, and *W*[*1*][*v*][*u*] is for non-intra macroblocks. The value of *quantiser_scale* is determined by vop_quant, dquant, dbquant, and quant_scale for luminance and chrominance, and additionally by vop_quant_alpha for grayscale alpha. For example, the value of quantiser scale for luminance and chrominance shall be an integer from 1 to 31 when not_8_bit == '0'. The following equation specifies the arithmetic to reconstruct F''[v][u] from QF[v][u] (for all coefficients except intra DC coefficients).

$$F''[v][u] = \begin{cases} 0, \text{ if } QF[v][u] = 0 \\ ((2 \times QF[v][u] + k) \times W[w][v][u] \times quantiser\_scale) / 16, \text{ if } QF[v][u] \neq 0 \end{cases}$$

$$\text{where :}$$

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

NOTE The above equation uses the "/" operator as defined in subclause 4.1.

7.4.4.2 Second inverse quantisation method

This subclause specifies the second of the two inverse quantisation methods. The method described here is used for all the coefficients other than the DC coefficient of an intra block when quant_type==0. In the second inverse quantisation method, the DC coefficient of an intra block is quantized using the same method as in the first inverse quantisation method (see subclause 7.4.4.1.1). The quantisation parameter *quantiser_scale* may take integer values from 1 to $2^{quant\_precision}$-1. The quantisation stepsize is equal to twice the *quantiser_scale*.

7.4.4.2.1 Dequantisation

$$|F''[v][u]| = \begin{cases} 0, & \text{if } QF[v][u] = 0, \\ (2 \times |QF[v][u]| + 1) \times quantiser\_scale, & \text{if } QF[v][u] \neq 0, \ quantiser\_scale \text{ is odd}, \\ (2 \times |QF[v][u]| + 1) \times quantiser\_scale - 1, & \text{if } QF[v][u] \neq 0, \ quantiser\_scale \text{ is even}. \end{cases}$$

The sign of *QF*[*v*][*u*] is then incorporated to obtain *F''*[*v*][*u*]: *F''*[*v*][*u*]= Sign(*QF*[*v*][*u*])×|*F''*[*v*][*u*]|

7.4.4.3 Nonlinear inverse DC quantisation

NOTE This subclause is valid for both quantisation methods.

Within an Intra macroblock for which short_video_header is 0, luminance blocks are called type 1 blocks, chroma blocks are classified as type 2. When short_video_header is 1, the inverse quantisation of DC intra coefficients is equivalent to using a fixed value of dc_scaler = 8, as described above in subclause 7.4.1.1.

- DC coefficients of Type 1 blocks are quantized by Nonlinear Scaler for Type 1
- DC coefficients of Type 2 blocks are quantized by Nonlinear Scaler for Type 2

Table 7-1 specifies the nonlinear dc_scaler expressed in terms of piece-wise linear characteristics.

Table 7-1 -- Non linear scaler for DC coefficients of DCT blocks, expressed in terms of relation with quantizer_scale

Component:Type	**dc_scaler for quantiser_scale range**			
	1 through 4	**5 through 8**	**9 through 24**	**>= 25**
Luminance: Type1	8	2x quantiser_scale	quantiser_scale +8	2 x quantiser_scale -16
Chrominance: Type2	8	(quantiser_scale +13)/2		quantiser_scale -6

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/FODJA Y. ADAMANE
ANSI Store order X017445. Downloaded 7/18/2007 3:56:21 PM ET
Single user license only. Copying and networking prohibited.

7.4.4.4 Saturation

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

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

7.4.4.5 Mismatch control

This mismatch control is only applicable to the first inverse quantisation method. Mismatch control shall be performed by any process equivalent to the following. Firstly all of the reconstructed, saturated coefficients, $F'[v][u]$ in the block shall be summed. This value is then tested to determine whether it is odd or even. If the sum is even then a correction shall be made to just one coefficient; $F[7][7]$. Thus:

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

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

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

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) maybe 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.

7.4.4.6 Summary of quantiser process for method 1

In summary, the method 1 inverse quantisation process is any process numerically equivalent to:

```
for (v=0; v<8;v++) {
    for (u=0; u<8;u++) {
        if (QF[v][u] == 0)
            F''[v][u] = 0;
        else if ( (u==0) && (v==0) && (macroblock_intra) ) {
            F''[v][u] = dc_scaler * QF[v][u];
        } else {
            if ( macroblock_intra ) {
                F''[v][u] = ( QF[v][u] * W[0][v][u] * quantiser_scale * 2 ) / 16;
            } else {
                F''[v][u] = ( ( ( QF[v][u] * 2 ) + Sign(QF[v][u]) ) * W[1][v][u]
                                                    * quantiser_scale ) / 16;
            }
        }
    }
}

sum = 0;
for (v=0; v<8;v++) {
    for (u=0; u<8;u++) {
        if ( F''[v][u] > $2^{bits\_per\_pixel + 3}$ - 1 ) {
            F'[v][u] = $2^{bits\_per\_pixel + 3}$ - 1;
        } else {
            if ( F''[v][u] < $-2^{bits\_per\_pixel + 3}$ ) {
                F'[v][u] = $-2^{bits\_per\_pixel + 3}$;
            } else {
                F'[v][u] = F''[v][u];
            }
        }
    sum = sum + F'[v][u];
    F[v][u] = F'[v][u];
    }
}

if ((sum & 1) == 0) {
    if ((F[7][7] & 1) != 0) {
        F[7][7] = F'[7][7] - 1;
    } else {
        F[7][7] = F'[7][7] + 1;
    }
}
```

7.4.5 Inverse DCT

Once the DCT coefficients, F[u][v] are reconstructed, the inverse DCT transform defined in annex A shall be applied to obtain the inverse transformed values, f[y][x]. These values shall be saturated so that: $-2^{bits\_per\_pixel} \leq f[y][x] \leq 2^{bits\_per\_pixel} - 1$, for all *x*, *y*.

If flag 'sadct_disable' in the header of VideoObjectLayer() is set to '0', standard inverse 8x8-DCT – as specified in version 1 – is only applied to 8x8-blocks with 64 opaque pels. In 8x8-blocks with a least one transparent pel (i.e. number of opaque pels is less than 64), standard inverse DCT is replaced by inverse shape adaptive DCT (SA-DCT). Similar to standard 8x8-DCT, SA-DCT reconvers F[v][u] to f[x][y]. The internal processing of inverse SA-DCT is controlled by special shape parameters which have to be computed from decoded BAB. An extended version of SA-DCT, called ΔDC-SA-DCT, is used in intra-coded 8x8-blocks with 'opaque_pels<64'. The decision on the transform to be used in the current 8x8-block depends on the settings of 'video_object_layer_shape', 'sadct_disable', 'vop_coding_type' and 'derived_mb_type' as well as parameter 'opaque_pels'. The decision rule is specified in Table V2 - 28. Details on internal SA-DCT and ΔDC-SA-DCT processing are given in clauses 10.3 and 10.4 of annex A.

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA V. AGAWANE
Single user license only. Copying and networking prohibited.

Table V2 - 28 – Decision rule for the using standard 8x8-DCT, SA-DCT and ΔDC-SA-DCT

decision rules	DCT tool
(video_object_layer_shape=="rectangular") || (sadct_disable==1) || (opaque_pels==64)	standard 8x8-DCT
(video_object_layer_shape !="rectangular") && (sadct_disable==0) && (opaque_pels <64) && (vop_coding_type ="B") && ((derived_mb_type==3) || (derived_mb_type==4))	ΔDC-SA-DCT
(video_object_layer_shape !="rectangular") && (sadct_disable==0) && (opaque_pels < 64) && (((vop_coding_type ="P") && (derived_mb_type!=3) && (derived_mb_type != 4)) || (vop_coding_type ="B"))	SA-DCT

7.4.8 Upsampling of the Inverse DCT output for Reduced Resolution VOP

For the VOP with vop_reduced_resolution flag equal to "1", the upsampling process described in this subclause shall be applied to the Inverse DCT output $f[y][x]$. When the flag is set to "0" or the flag is not present, this upsampling process shall be skipped.

The 16×16 reconstructed prediction error block is obtained by upsampling the 8×8 reconstructed prediction error block $f[y][x]$. In order to realize a simple implementation, filtering is closed within a block which enables to perform an individual upsampling on block basis. Figure V2 - 6 shows the positioning of samples. The upsampling procedure for the luminance and chrominance pixels which are inside the 16×16 reconstructed prediction error blocks is defined in subclause 7.4.8.1. For the creation of the luminance and chrominance pixels which are at the boundary of 16×16 reconstructed prediction error block, the procedure is defined in subclause 7.4.8.2. Chrominance blocks as well as luminance blocks are up-sampled. The upsampled 16×16 prediction error blocks are used for the motion compensation decoding process for Reduced Resolution VOP described in the subclause 7.6.10.

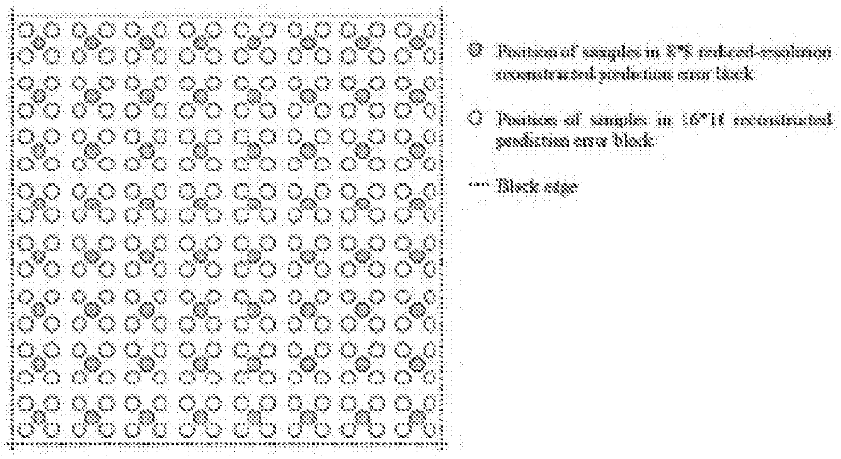

Position of samples in 8*8 reduced-resolution reconstructed prediction error block
Position of samples in 16*16 reconstructed prediction error block
Block edge

Figure V2 - 6 – Positioning of samples in 8×8 reduced resolution reconstructed prediction error block and 16×16 reconstructed prediction error block

7.4.6.1 Upsampling procedure for the pixels inside a 16x16 reconstructed prediction error block

The creation of reconstructed prediction error for pixels inside block is described in Figure V2 - 7.

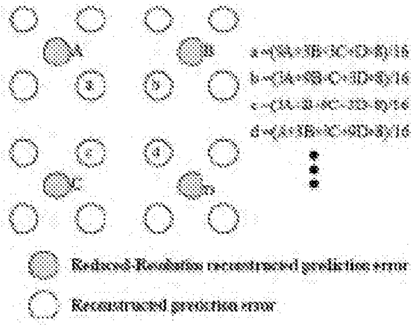


Figure V2 - 7 – Creation of reconstructed prediction error for pixels inside block

7.4.6.2 Upsampling procedure for the pixels at the boundary of 16x16 reconstructed prediction error block

The creation of reconstructed prediction error for pixels of a 16x16 block is shown in Figure V2 - 8.

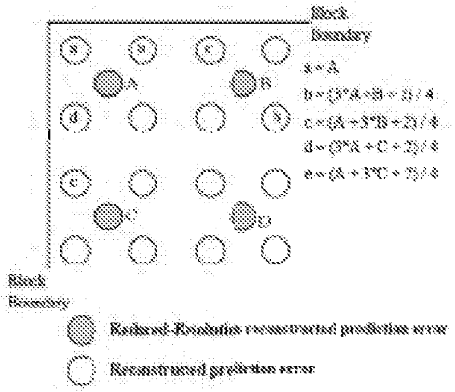


Figure V2 - 8 – Creation of reconstructed prediction error for pixels at the block boundary

7.5 Shape decoding

Binary shape decoding is based on a block-based representation. The primary coding methods are block-based context-based binary arithmetic decoding and block-based motion compensation. The primary data structure used is denoted as the binary alpha block (bab). The bab is a square block of binary valued pixels representing the opacity/transparency for the pixels in a specified block-shaped spatial region of size 16x16 pels. In fact, each bab is co-located with each texture macroblock.

7.5.1 Higher syntactic structures

7.5.1.1 VOL decoding

If video_object_layer_shape is equal to '00' then no binary shape decoding is required. Otherwise, binary shape decoding is carried out.

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA Y. ADAMANE
ANSI Store order X037445 Downloaded 7/18/2007 3:56:21 PM ET
Single user license only. Copying and networking prohibited.

7.5.1.2 VOP decoding

If video_object_layer_shape is not equal to '00' then, for each subsequent VOP, the dimensions of the bounding rectangle of the reconstructed VOP are obtained from:

- vop_width
- vop_height

If these decoded dimensions are not multiples of 16, then the values of vop_width and vop_height are rounded up to the nearest integer, which is a multiple of 16.

Additionally, in order to facilitate motion compensation, the horizontal and spatial position of the VOP are obtained from:

- vop_horizontal_mc_spatial_ref
- vop_vertical_mc_spatial_ref

These spatial references may be different for each VOP but the same coordinate system must be used for all VOPs within a vol. Additionally, the decoded spatial references must have an even value

- vop_shape_coding_type

This flag enables the use of intra shape codes in P, B, or S(GMC)-VOPs. Finally, in the VOP class, it is necessary to decode

- change_conv_ratio_disable

This specifies whether conv_ratio is encoded at the macroblock layer.

Once the above elements have been decoded, the binary shape decoder may be applied to decode the shape of each macroblock within the bounding rectangle.

7.5.2 Macroblock decoding

The shape information for each macroblock residing within the bounding rectangle of the VOP is decoded into the form of a 16x16 bab.

7.5.2.1 Mode decoding

Each bab belongs to one of seven types listed in Table 7-2. The type information is given by the bab_type field which influences decoding of further shape information. For I-VOPs only three out of the seven modes are allowed as shown in Table 7-2.

Table 7-2 -- List of bab types

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

7.5.2.1.1 I-VOPs

Suppose that $f(x,y)$ is the bab_type of the bab located at (x,y), where x is the BAB column number and y is the BAB row number. The code word for the bab_type at the position (i,j) is determined as follows. A context C is computed from previously decoded bab_type's.

$$C = 27*(f(i-1,j-1)-2) + 9*(f(i,j-1)-2) + 3*(f(i+1,j-1)-2) + (f(i-1,j)-2)$$

If $f(x,y)$ references a bab outside the current VOP, bab_type is assumed to be transparent for that bab (i.e. $f(x,y)=2$). The bab_type of babs outside the current video packet is also assumed to be transparent. The VLC used to decode bab_type for the current bab is switched according to the value of the context C. This context-switched VLC table is given in Table B-27.

7.5.2.1.2 P- and B-, and S(GMC)-VOPs

The decoding of the current bab_type is dependent on the bab_type of the co-located bab in the reference VOP. The reference VOP is either a forward reference VOP or a backward reference VOP. The forward reference VOP is defined as the most recent non-empty (i.e. vop_coded != 0) I- or P-, or S(GMC)-VOP in the past, while the backward VOP is defined as the most recently decoded I- or P-, or S(GMC)-VOP in the future. If the current VOP is a P-, or S(GMC)-VOP, the forward reference VOP is selected as the reference VOP. If the current VOP is a B-VOP the following decision rules are applied:

1. If the backwards reference VOP is empty, the non-empty one (forward) is selected as the reference VOP for the current B-VOP.

2. If both reference VOPs are non-empty, the forward reference VOP is selected if its temporal distance to the current B-VOP is not larger than that of the backward reference VOP, otherwise, the backward one is chosen.

In the special cases when closed_gov == 1 and the forward reference VOP belongs to the previous GOV, the current B-VOP takes the backward VOP as reference.

If the sizes of the current and reference VOPs are different, some babs in the current VOP may not have a co-located equivalent in the reference VOP. Therefore the bab_type matrix of the reference VOP is manipulated to match the size of the current VOP. Two rules are defined for that purpose, namely a cut rule and a copy rule:

- *cut rule*. If the number of lines (respectively columns) is smaller in the current VOP than in the reference VOP, the bottom lines (respectively rightmost columns) are eliminated from the reference VOP such that both VOP sizes match.

- *copy rule*. If the number of lines (respectively columns) is larger in the current VOP than in the reference VOP, the bottom line (respectively rightmost column) is replicated as many times as needed in the reference VOP such that both VOP sizes match.

An example is shown in Figure 7-8 where both rules are applied.

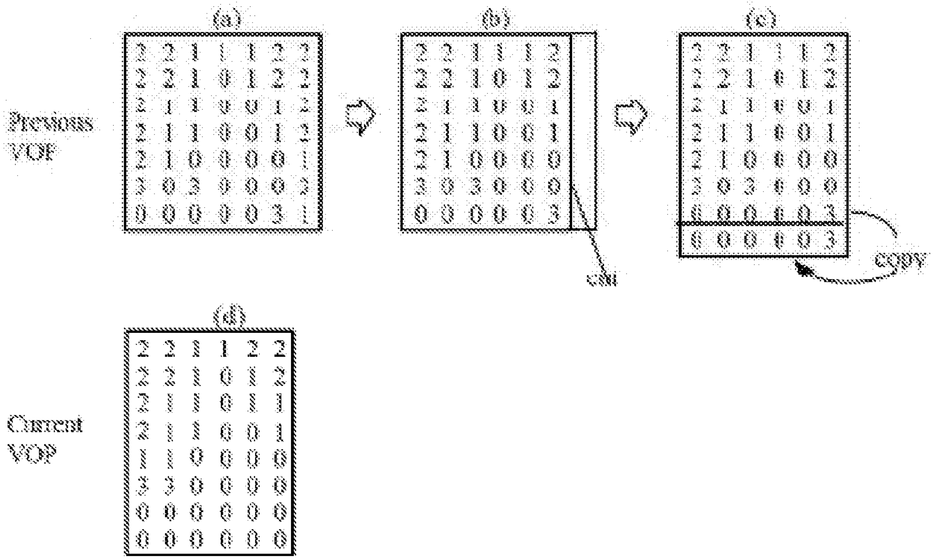


Figure 7-8 -- Example of size fitting between current VOP and reference VOP. The numbers represent the type of each bab

The VLC to decode the current bab_type is switched according to the value of bab_type of the co-located bab in the reference VOP. This context-switched VLC tables for P, B, and S(GMC)-VOPs are given in Table B-28. If the type of the bab is transparent, then the current bab is filled with zero (transparent) values. A similar procedure is carried out if the type is opaque, where the reconstructed bab is filled with values of 255 (opaque). For both transparent and opaque types, no further decoding of shape-related data is required for the current bab. Otherwise further decoding steps are necessary, as listed in Table 7-3. Decoding for motion compensation is described in subclause 7.5.2.2, and cae decoding in subclause 7.5.2.5.

Table 7-3 -- Decoder components applied for each type of bab

bab_type	Motion compensation	CAE decoding
0	yes	no
1	yes	no
2	no	no
3	no	no
4	no	yes
5	yes	yes
6	yes	yes

7.5.2.2 Binary alpha block motion compensation

Motion Vector of shape (MVs) is used for motion compensation (MC) of shape. The value of MVs is reconstructed as described in subclause 7.5.2.3. Unrestricted motion compensation is applied, however, Overlapped MC, half sample MC and 8x8 MC are not carried out. Integer pixel unrestricted motion compensation is carried out on a 16x16 block basis according to subclause 7.5.2.4.

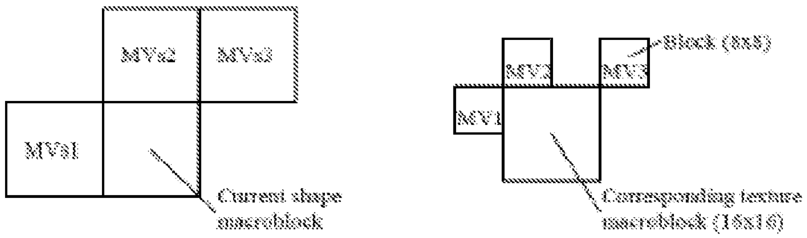
If bab_type is MVDs==0 && No Update or MVDs!=0 && No Update then the motion compensated bab is taken to be the decoded bab, and no further decoding of the bab is necessary. Otherwise, cae decoding is required.

7.5.2.3 Motion vector decoding

If bab_type indicates that MVDs!=0, then mvds_x and mvds_y are VLC decoded. For decoding mvds_x, the VLC given in Table B-29 is used. The same table is used for decoding mvds_y, unless the decoded value of mvds_x is zero. If mvds_x == 0, the VLC given in Table B-30 is used for decoding mvds_y. If bab_type indicates that MVDs==0, then both mvds_x and mvds_y are set to zero.

The integer valued shape motion vector MVs=(mvs_x,mvs_y) is determined as the sum of a predicted motion vector MVPs and MVDs = (mvds_x,mvds_y), where MVPs is determined as follows.

MVPs is determined by analysing certain candidate motion vectors of shape (MVs) and motion vectors of selected texture blocks (MV) around the MB corresponding to the current bab. They are located and denoted as shown in Figure 7-9 where MV1, MV2 and MV3 are rounded up to integer values towards 0. If the selected texture block is a field predicted macroblock, then MV1, MV2 or MV3 are generated by averaging the two field motion vectors and rounding toward zero. Regarding the texture MV's, the convention is that a MB possessing only 1 MV is considered the same as a MB possessing 4 MV's, where the 4 MV's are equal. By traversing MVs1, MVs2, MVs3, MV1, MV2 and MV3 in this order, MVPs is determined by taking the first encountered MV that is defined. If no candidate motion vectors is defined, MVPs = (0,0).



(1) MV for shape (2) MV for texture

Figure 7-9 -- Candidates for MVPs

In the case that video_object_layer_shape is "binary_only" or vop_coding_type indicates B-VOP, MVPs is determined by considering the motion vectors of shape (MVs1, MVs2 and MVs3) only. The following subclauses explain the definition of MVs1, MVs2, MVs3, MV1, MV2 and MV3 in more detail.

Defining candidate predictors from texture motion vectors:

One shape motion vector predictor MV_i (i =1,2,3) is defined for each block located around the current bab according to *Figure 7-9 (2)*. The definition only depends on the transparency of the reference MB. MV_i is set to the corresponding block vector as long as it is in a non-transparent reference MB, otherwise, it is not defined. Note that if a reference MB is outside the current VOP or video packet, it is treated as a transparent MB.

Defining candidate predictors from shape motion vectors:

The candidate motion vector predictors MVs_i are defined by the shape motion vectors of neighbouring bab located according to Figure 7-9 (1). The MVs_i are defined according to Table 7-4.

Table 7-4 – Definition of candidate shape motion vector predictors MVs1, MVs2, and MVs3 from shape motion vectors for P, B, and S(GMC)-VOPs. Note that interlaced modes are not included

Shape mode of reference MB	MVs for each reference shape block-i (a shape block is 16x16)
MVDs == 0 or MVDs !=0 bab_type 0, 1, 5,6	The retrieved shape motion vector of the said reference MB is defined as MVs_i. Note that MVs_i is defined, and hence valid, even if the reconstructed shape block is transparent.
all_0, bab_type 2	MVs_i is undefined
all=255, bab_type 3	MVs_i is undefined
Intra, bab_type 4	MVs_i is undefined

If the reference MB is outside of the current video packet, MV_i and MVs_i are undefined.

7.5.2.4 Motion compensation

For inter mode babs (bab_type = 0,1,5 or 6), motion compensation is carried out by simple MV displacement according to the MVs.

Specifically, when bab_type is equal to 0 or 1 i.e. for the no-update modes, a displaced block of 16x16 pixels is copied from the binary alpha map of the previously decoded I or P-, or S(GMC)- VOP for which vop_coded is not equal to '0'. When the bab_type is equal to 5 or 6 i.e. when interCAE decoding is required, then the pixels immediately bordering the displaced block (to the left, right, top and bottom) are also copied from the most recent valid reference VOP's (as defined in subclause 6.3.5) binary alpha map into a temporary shape block of 18x18 pixels size (see Figure 7-12). If the displaced position is outside the bounding rectangle, then these pixels are assumed to be "transparent".

If the current VOP is a B-VOP the following decision rules are applied:

- If the backwards reference VOP is empty, the non-empty one (forward) is selected as the reference VOP for the current B-VOP.
- If both reference VOPs are non-empty, the forward reference VOP is selected if its temporal distance to the current B-VOP is not larger than that of the backward reference VOP, otherwise, the backward one is chosen.

In the special cases when closed_gov == 1 and the forward reference VOP belongs to the previous GOV, the current B-VOP takes the backward VOP as reference.

7.5.2.5 Context based arithmetic decoding

Before decoding the binary_arithmetic_code field, border formation (see subclause 7.5.2.5.2) needs to be carried out. Then, if the scan_type field is equal to 0, the bordered to-be decoded bab and the eventual bordered motion compensated bab need to be transposed (as for matrix transposition). *If change_conv_rate_disable is equal to 0,* then conv_ratio is decoded to determine the size of the sub-sampled BAB, which is 16/conv_ratio by 16/conv_ratio pixels large. If change_conv_rate_disable is equal to 1, then the decoder assumes that the bab is not subsampled and thus the size is simply 16x16 pixels. Binary_arithmetic_code is then decoded by a context-based arithmetic decoder as follows. The arithmetic decoder is firstly initialised (see subclause 7.5.3.3). The pixels of the sub-sampled bab are decoded in raster order. At each pixel,

1. A context number is computed based on a template, as described in subclause 7.5.2.5.1.
2. The context number is used to access the probability table (Table B-32).
3. Using the accessed probability value, the next bits of binary_arithmetic_code are decoded by the arithmetic decoder to give the decoded pixel value.

When all pixels in sub-sampled BAB have been decoded, the arithmetic decoder is terminated (see subclause 7.5.3.6).

If the scan_type field is equal to 0, the decoded bab is transposed. Then up-sampling is carried out if conv_ratio is different from 1, as described in subclause 7.5.2.5.3. Then the decoded bab is copied into the decoded shape map.

7.5.2.5.1 Context computation

For INTRA coded BABs, a 10 bit context $C = \sum_k c_k \cdot 2^k$ is built for each pixel as illustrated in Figure 7-10 (a), where c_k=0 for transparent pixels and c_k=1 for opaque pixels.

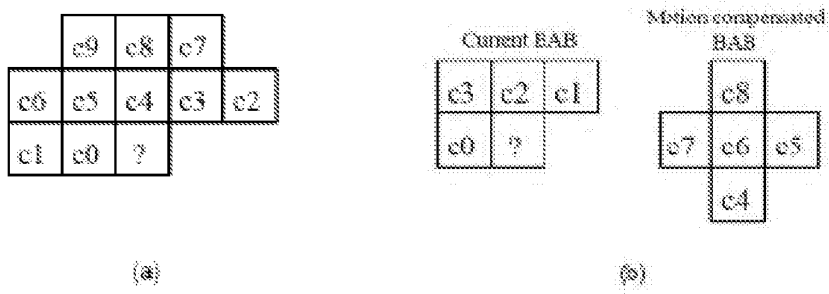


Figure 7-10 -- (a) The INTRA template (b) The INTER template where c6 is aligned with the pixel to be decoded. The pixel to be decoded is marked with '?'

For INTER coded BABs, temporal redundancy is exploited by using pixels from the bordered motion compensated BAB (depicted in Figure 7-12) to make up part of the context. Specifically, a 9 bit context $C = \sum_k c_k \cdot 2^k$ is built as illustrated in Figure 7-10 (b).

There are some special cases to note.

- When building contexts, any pixels outside the bounding rectangle of the current VOP to the left and above are assumed to be zero (transparent).
- When building contexts, any pixels outside the space of the current video packet to the left and above are assumed to be zero (transparent).
- The template may cover pixels from BABs which are unknown at decoding time. Unknown pixels are defined as area U in Figure 7-11.
- The values of these unknown pixels are defined by the following procedure:
    - When constructing the INTRA context, the following steps are taken in the sequence
        1. if (c7 is unknown) c7=c8,
        2. if (c3 is unknown) c3=c4,
        3. if (c2 is unknown) c2=c3.
    - When constructing the INTER context, the following conditional assignment is performed.

if (c1 is unknown) c1=c2

7.5.2.5.2 Border formation

When decoding a BAB, pixels from neighbouring BABs shall be used to make up the context. For both the INTRA and INTER cases, a 2 pixel wide border about the current BAB is used where pixels values are known, as depicted in Figure 7-11.

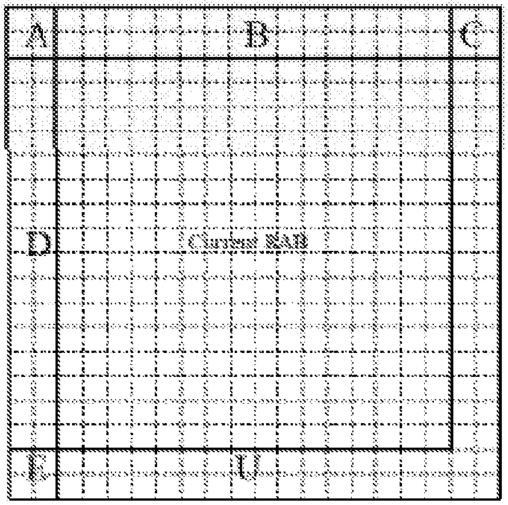


Figure 7-11 – Bordered BAB. A: TOP_LEFT_BORDER. B: TOP_BORDER. C: TOP_RIGHT_BORDER. D: LEFT_BORDER. E: BOTTOM_LEFT_BORDER. U: pixels which are unknown when decoding the current BAB

If the value of conv_ratio is not equal to 1, a sub-sampling procedure is further applied to the BAB borders for both the current BAB and the motion compensated BAB.

The border of the current BAB is partitioned into 5 regions:

- TOP_LEFT_BORDER, which contains pixels from the BAB located to the upper-left of the current BAB and which consists of 2 lines of 2 pixels
- TOP_BORDER, which contains pixels from the BAB located above the current BAB and which consists of 2 lines of 16 pixels
- TOP_RIGHT_BORDER, which contains pixels from the BAB located to the upper-right of the current BAB and which consists of 2 lines of 2 pixels
- LEFT_BORDER, which contains pixels from the BAB located to the left of the current BAB and which consists of 2 columns of 16 pixels
- BOTTOM_LEFT_BORDER, which contains pixels from the BAB located to the bottom-left of the current BAB and which consists of 2 lines of 2 pixels

The TOP_LEFT_BORDER and TOP_RIGHT_BORDER are not sub-sampled, and kept as they are. The TOP_BORDER and LEFT_BORDER are sub-sampled such as to obtain 2 lines of 16/conv_ratio pixels and 2 columns of 16/conv_ratio pixels, respectively.

The sub-sampling procedure is performed on a line-basis for TOP_BORDER, and a column-basis for LEFT_BORDER. For each line (respectively column), the following algorithm is applied: the line (respectively column) is split into groups of conv_ratio pixels. For each group of pixels, one pixel is associated in the sub-sampled border. The value of the pixel in the sub-sampled border is OPAQUE if half or more pixels are OPAQUE in the corresponding group. Otherwise the pixel is TRANSPARENT.

The 2x2 BOTTOM_LEFT_BORDER is filled by replicating downwards the 2 botton border samples of the LEFT_BORDER after the down-sampling (if any).

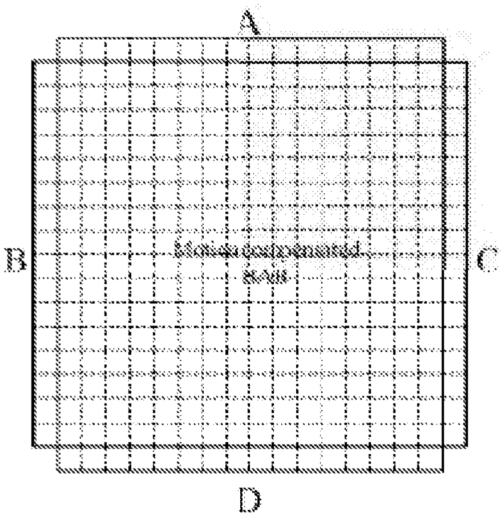


Figure 7-12 – Bordered motion compensated BAB. A: TOP_BORDER. B: LEFT_BORDER. C: RIGHT_BORDER. D: BOTTOM_BORDER

In the case of a motion compensated BAB, the border is also partitioned into 4, as shown Figure 7-12:

- TOP_BORDER, which consists of a line of 16 pixels
- LEFT_BORDER, which consists of a column of 16 pixels
- RIGHT_BORDER, which consists of a column of 16 pixels
- BOTTOM_BORDER, which consists of a line of 16 pixels

The very same sub-sampling process as described above is applied to each of these borders.

7.5.2.5.3 Upsampling

When conv_ratio is different from 1, up-sampling is carred out for the BAB. This is illustrated in Figure 7-13 where "O" in this figure is the coded pixel and "X" is the interpolated pixel. To compute the value of the interpolated pixel, a filter context from the neighboring pixels is first calculated. For the pixel value calculation, the value of "0" is used for a transparent pixel, and "1" for an opaque pixel. The values of the interpolated pixels (Pi, i=1,2,3,4, as shown in Figure 7-14) can then be determined by the following equation:

P1 : if(4*A + 2*(B+C+D) + (E+F+G+H+I+J+K+L) > Th[Cf]) then "1" else "0"

P2 : if(4*B + 2*(A+C+D) + (E+F+G+H+I+J+K+L) > Th[Cf]) then "1" else "0"

P3 : if(4*C + 2*(B+A+D) + (E+F+G+H+I+J+K+L) > Th[Cf]) then "1" else "0"

P4 : if(4*D + 2*(B+C+A) + (E+F+G+H+I+J+K+L) > Th[Cf]) then "1" else "0"

The 8-bit filter context, Cf, is calculated as follows:

$$C_f = \sum_k c_k \cdot 2^k$$

Based on the calculated Cf, the threshold value (Th[Cf]) can be obtained from the look-up table as follows:

```
TH[256] = {
 3, 6, 6, 7, 4, 7, 7, 8, 6, 7, 5, 8, 7, 8, 8, 9,
 6, 5, 5, 8, 5, 6, 8, 9, 7, 6, 8, 9, 8, 7, 9, 10,
 6, 7, 7, 8, 7, 8, 8, 9, 7, 10, 8, 9, 8, 9, 9, 10,
 7, 8, 6, 9, 6, 8, 9, 10, 8, 9, 9, 10, 11, 10, 10, 11,
 6, 9, 5, 8, 5, 6, 8, 9, 7, 10, 10, 9, 8, 7, 9, 10,
 7, 6, 8, 9, 8, 7, 7, 10, 8, 9, 9, 10, 9, 8, 10, 9,
 7, 8, 8, 9, 6, 6, 9, 10, 8, 9, 9, 10, 9, 10, 10, 9,
 8, 9, 11, 10, 7, 10, 10, 11, 9, 12, 10, 11, 10, 11, 11, 12,
 6, 7, 5, 8, 5, 6, 8, 9, 5, 6, 6, 9, 8, 9, 9, 10,
 5, 8, 8, 9, 6, 7, 9, 10, 6, 7, 8, 10, 8, 10, 10, 11,
 7, 8, 6, 9, 8, 6, 9, 10, 8, 7, 9, 10, 9, 10, 10, 11,
 8, 9, 7, 10, 9, 10, 8, 11, 9, 10, 10, 11, 10, 11, 9, 12,
 7, 8, 6, 9, 8, 6, 9, 10, 10, 9, 7, 10, 8, 10, 10, 11,
 8, 7, 7, 10, 7, 8, 8, 9, 9, 10, 10, 11, 10, 11, 11, 12,
 8, 9, 9, 10, 9, 10, 10, 9, 9, 10, 10, 11, 10, 11, 11, 12,
 9, 10, 10, 11, 10, 11, 11, 12, 10, 11, 11, 12, 11, 12, 12, 13};
```

TOP_LEFT_BORDER, TOP_RIGHT_BORDER, sub-sampled TOP_BORDER and sub-sampled LEFT_BORDER described in the previous subclause are used. The other pixels outside the BAB are extended from the outermost pixels inside the BAB as shown in Figure 7-13.

In the case that conv_ratio is 4, the interpolation is processed twice. The above mentioned borders of 4x4 BAB are used for the interpolation from 4x4 to 8x8, and top-border (respectively left-border) for the interpolation from 8x8 to 16x16 are up-sampled from the 4x4 BAB top-border (respectively left-border) by simple repetition.

When the BAB is on the left (and/or top) border of VOP, the borders outside VOP are set to zero value. The upsampling filter shall not use pixel values outside of the current video packet.

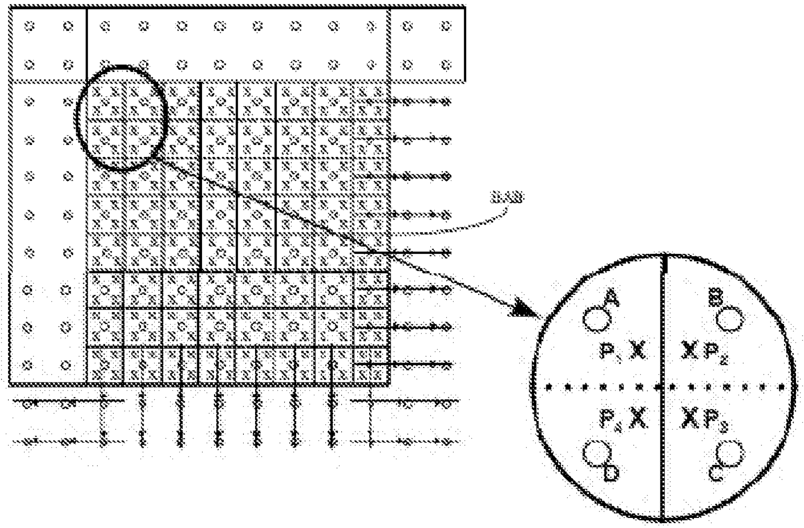
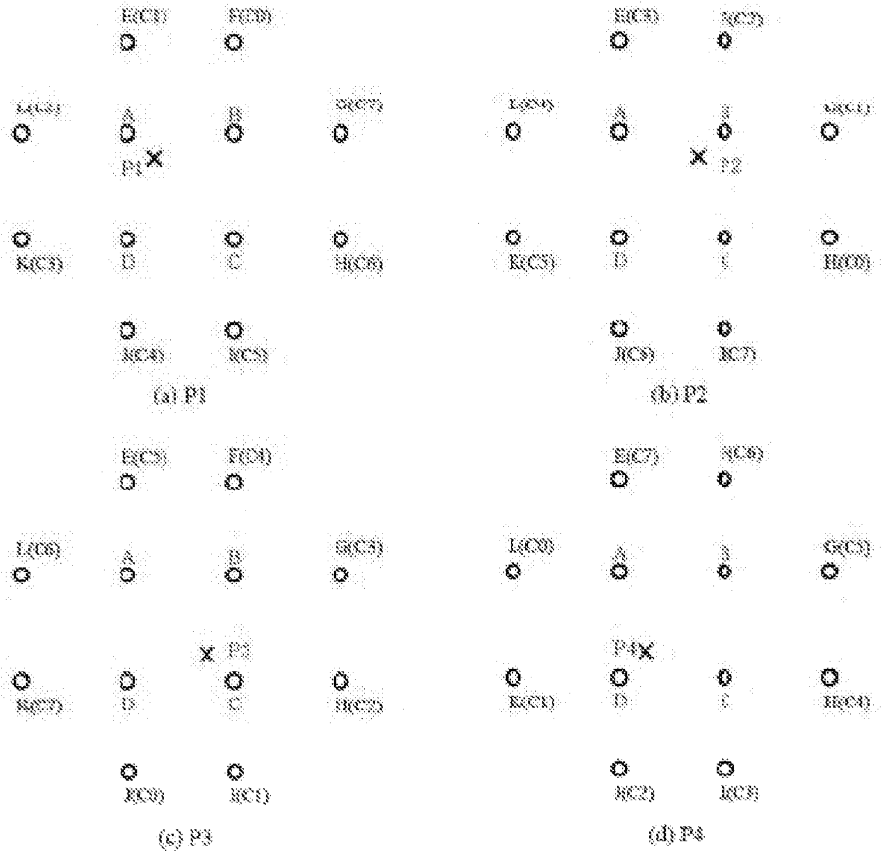


Figure 7-13 – Upsampling

© ISO/IEC 2001 – All rights reserved
Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/OSCAR Y. ADAMANE
ANSI Store order #X37445 Downloaded 7/18/2007 3:08:21 PM ET
Single user license only. Copying and networking prohibited.



(a) P1 (b) P2 (c) P3 (d) P4

Figure 7-14 – Interpolation filter and interpolation construction

7.5.2.5.4 Down-sampling process in inter case

If bab_type is '5' or '6' (see Table 7-3), downsampling of the motion compensated bab is needed for calculating the 9 bit context in the case that conv_ratio is not 1. The motion compensated bab of size 16x16 pixels is down sampled to bab of size 16/conv_ratio by 16/conv_ratio pixels by the following rules:

- conv_ratio==2

If the average of pixel values in 2 by 2 pixel block is equal to or greater than 127.5 the pixel value of the downsampled bab is set to 255 otherwise it is set to 0.

- conv_rato==4

If the average of pixel values in 4 by 4 pixel block s equal to or greater than 127.5 the pixel value of the downsampled bab is set to 255 otherwise it is set to 0.

7.5.3 Arithmetic decoding

Arithmetic decoding consists of four main steps:

- Removalof stuffed bits
- Initialzaton which is performed prior to the decodingof the first symbol

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON [illegible]
[illegible]
Single user license only. Copying and networking prohibited.

- Decoding of the symbol themselves. The decoding of each symbol may be followed by a re-normalization step.
- Termination which is performed after the decoding of the last symbol

7.5.3.1 Registers, symbols and constants

Several registers, symbols and constants are defined to describe the arithmetic decoder.

- HALF: 32-bit fixed point constant equal to ½ (0x80000000)
- QUARTER: 32-bit fixed point constant equal to ¼ (0x40000000)
- L: 32-bit fixed point register. Contains the lower bound of the interval
- R: 32-bit fixed point register. Contains the range of the interval.
- V: 32-bit fixed point register. Contains the value of the arithmetic code. V is always larger than or equal to L and smaller than L+R.
- p0: 16-bit fixed point register. Probability of the '0' symbol.
- p1: 16-bit fixed point register. Probability of the '1' symbol.
- LPS: boolean. Value of the least probable symbol ('0' or '1').
- bit: boolean. Value of the decoded symbol.
- pLPS: 16-bit fixed point register. Probability of the LPS.
- rLPS: 32-bit fixed point register. Range corresponding to the LPS.

7.5.3.2 Bit stuffing

In order to avoid start code emulation, 1's are stuffed into the bitstream whenever there are too many successive 0's. If the first MAX_HEADING bits are 0's, then a 1 is transmitted after the MAX_HEADING-th 0. If MAX_MIDDLE or more 0's are sent successively a 1 is inserted after the MAX_MIDDLE-th 0. If the number of trailing 0's is larger than MAX_TRAILING, then a 1 is appended to the stream.The decoder shall properly skip these inserted 1's when reading data into the V register (see subclauses 7.5.3.3 and 7.5.3.5).

MAX_HEADING equals 3, MAX_MIDDLE equals 10, and MAX_TRAILING equals 2

7.5.3.3 Initialization

The lower bound L is set to 0, the range R to HALF-0x1 (0x7fffffff) and the first 31 bits are read in register V.

7.5.3.4 Decoding a symbol

When decoding a symbol, the probability p0 of the '0' symbol is provided according to the context computed in subclause 7.5.2.5.1 and using Table B-32. p0 uses a 16-bit fixed-point number representation. Since the decoder is binary, the probability of the '1' symbol is defined to be 1 minus the probability of the '0' symbol, i.e. p1 = 1-p0.

The least probable symbol LPS is defined as the symbol with the lowest probability. If both probabilities are equal to ½ (0x8000), the '0' symbol is considered to be the least probable.

The range rLPS associated with the LPS may simply be computed as R*pLPS: The 16 most significant bits of register R are multiplied by the 16 bits of pLPS to obtain the 32 bit rLPS number.

The interval [L,L+R) is split into two intervals [L,L+R-rLPS) and [L+R-rLPS,L+R). If V is in the latter interval then the decoded symbol is equal to LPS. Otherwise the decoded symbol is the opposite of LPS. The interval [L,L+R) is then reduced to the sub-interval in which V lies.

After the new interval has been computed, the new range R might be smaller than QUARTER. If so, re-normalization is carried out, as described below.

7.5.3.5 Re-normalization

As long as R is smaller than QUARTER, re-normalization is performed.

- If the interval [L,L+R) is within [0,HALF), the interval is scaled to [2L,2L+2R). V is scaled to 2V.
- If the interval [L,L+R) is within [HALF,1) the interval is scaled to [2(L-HALF),2(L-HALF)+2R). V is scaled to 2(V-HALF).
- Otherwise the interval is scaled to [2(L-QUARTER),2(L-QUARTER)+2R). V is scaled to 2(V-QUARTER).

After each scaling, a bit is read and copied into the least significant bit of register V.

7.5.3.6 Termination

After the last symbol has been decoded, additional bits need to be "consumed". They were introduced by the encoder to guarantee decodability.

In general 3 further bits need to be read. However, in some cases, only two bits need to be read. These cases are defined by:

- if the current interval covers entirely [QUARTER-0x1,HALF)
- if the current interval covers entirely [HALF-0x1, 3QUARTER)

After these additional bits have been read, 32 bits shall be "unread", i.e. put the content of register V back into the bit buffer.

7.5.3.7 Software

The example software for arithmetic decoding for binary shape decoding is included in annex B.

7.5.4 Spatial scalable binary shape decoding

7.5.4.1 Decoding of base layer

The decoding process of the base layer is the same as non-scalable binary shape decoding process.

7.5.4.2 Decoding of enhancement layer

When spatial scalability is enabled (scalability is set to 1 and hierarchy_type is set to 0) with enhancement_type == 0 or when spatial scalability is enabled with enhancement_type == 1 and use_ref_shape == 0, scalable shape coding process is used for decoding of binary shape.

If spatial scalability is enabled, use_ref_shape is set to 1 and enhancement_type is set to 1, the same non-scalable decoding process is applied for binary shape of enhancement layer. In this case, the following rules are applied for enhancement layer.

1. In PVOP, Inter shape coding should be done as bab_type of co-located MB in the reference VOP (lower layer) is "Opaque".

2. In BVOP, forward reference VOP, most recently decoded non-empty VOP in the same layer, is always selected as reference VOP of shape coding.

If spatial scalability is enabled, use_ref_shape is set to 1 and enhancement_type is set to 0, then the up-sampled binary shape fom reference layer is used for the binary shape of enhancement layer. The up sampling and down sampling process of this purpose also follows up-down sampling method described in the subclause 7.5.4.4.

When spatial scalability is enabled and enhancement_type is set to 0, in the enhancement layer, the forward prediction in P-VOP and the backward prediction in B-VOP are used as the spatial prediction. In that case the shape informaion is coded by scan interleaving (SI) based method. For the forward prediction in B-VOP a motion compensated temporal prediction is made from the reference VOP in the enhancement layer. In that case the shape informaion is coded by the CAE method as like in base layer except that the shape motion vectors are obtained from those of the collocated bab in the lower layer. Motion vector and shape coding mode(bab_type) of collocated bab in the lower layer are used for decoding the enhancement layer bab.

The location of collocated bab in the lower layer can be found by following method.

collocated_MBX
= min (max (0, current_MBX*shape_hor_sampling_factor_m/shape_hor_sampling_factor_n), NumMBXLower-1);

collocated_MBY
= min (max(0, current_MBY*shape_ver_sampling_factor_m/shap_ever_sampling_factor_n), NumMBYLower-1);

For the current MB location [current_MBX, current_MBY], the location of collocated bab in the reference_layer is denoted as [collocated_MBX, collocated_MBY]. current_MBX, current_MBY, collocated_MBX and collocated_MBY are the MB-unit coordinations. NumMBXLower and NumMBYLower denote the number of microblocks in the lower layer VOP on horizontal and vertical directions, respectively.

7.5.4.3 Prediction in the enhancement layer

A motion compensated temporal prediction is made from the reference VOP in the enhancement layer. In addition, a spatial prediction is formed from the lower reference layer decoded VOP. These predictions are selected individually or combined to form the actual prediction.

vop_shape_coding_type is not used in the Spatial Scalable Binary Shape Coding process.

7.5.4.4 Spatial prediction

The spatial prediction is made by resampling the lower reference layer reconstructed VOP to the same sampling grid as the enhancement layer. For the resampling, repetition is used on the lower layer.

For enhancement layer encoding/decoding, the reference_layer VOP should be up-sampled as the sampling ratio, which is included in the VOL syntax. In VOL syntax for enhancement layer, there are four fields for the up-sampling ratio, i.e., shape_hor_sampling_factor_n, shape_hor_sampling_factor_m, shape_vert_sampling_factor_n and shape_vert_sampling_factor_n.

Down sampling
For the sampling rate, below sampling process is adopted for both horizontal and vertica directions.

For the down-sampling rate m/n,

$m/n = [1/2^k][(m2^k)/n]$, where k is the largest integer that is equal or smaller than $\log_2(n/m)$.

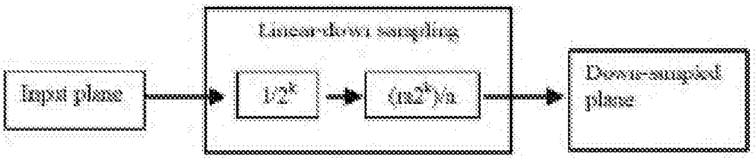


Figure V2 - 9 – Down sampling process

1. Find k = floor($\log_2(n/m)$).

2. Sampling a pixel per 2^k pixels.

    if($x\%2^k == (2^k - 1)$) Sampling F(x) pixel;

3. On the result of 2, sampling a pixel per $n/(m2^k)$ pixels.

Up sampling

For the sampling rate, below sampling process is adopted for both horizontal and vertical directions.

For the up-sampling rate n/m,

$n/m = [n/(m2^k)]\ [2^k]$, where k is the largest integer that is equal or smaller than $\log_2(n/m)$.

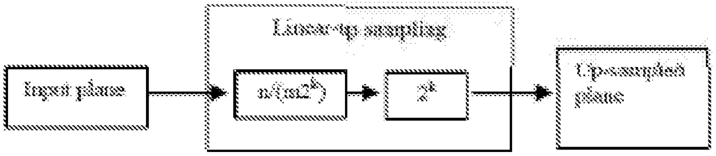


Figure V2 - 10 – Up sampling process

1. Find k = floor($\log_2(n/m)$).

2. Repeat one pixel per $(m2^k)/(n-m2^k)$ pixels.

3. On the result of 2, Repeat $2^k - 1$ pixels per pixel.

For a case that the sampling ratio is 1/2 for both horizontal and vertical direction, one pixel value in the lower layer is upsampled to 2x2 pixels in the current layer by the following methods:

The binary alpha image $f_{current}[y_c][x_c]$ in the enhancement layer is upsampled by repetition of the pixels of the image $f_{lower}[y_L][x_L]$ in the lower layer according to the following formula:

$f_{current}[2y_L+i][2x_L+j] = f_{lower}[y_L][x_L]$, (i, j=0, 1)

In Figure 7-s, the spatial prediction of one BAB is shown for example. In the enhancement layer the pixel value $f_{current}[2y_L+1][2x_L+1$ is always same with the corresponding pixel value $f_{lower}[y_L][x_L]$ in the lower layer.

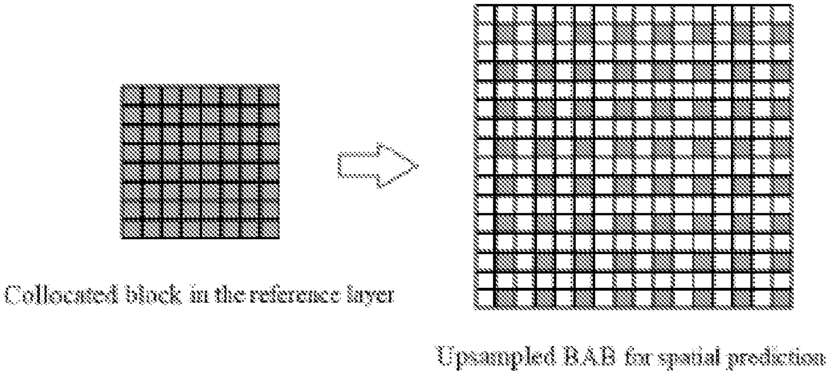

Collocated block in the reference layer
Upsampled BAB for spatial prediction

Figure V2 - 11 -- Associated pixel locations and spatial prediction of current BAB by repetition

In the case of arbitrary shape VOPs, its size and locations change from time to time. In order to ensure the formation of the spatial prediction, it is necessary to identify the location of the resampled VOPs in the absolute coordinate system. vop_horizontal_mc_spatial_ref and vop_vertical_mc_spatial_ref in the resampled reference layer should be coincident with those in the enhanoment layer. Hence after the position of the spatial reference of the reference layer is scaled by *_sampling_factor_n/*_sampling_factor_m, the resampled reference VOP should be relocated by referencing the offset values that are the differences between the positions of the spatial references of two VOPs in the horizontal and the vertical direction.

Since the shape motion vectors and the shape modes in the reference layer are also needed for the actual shape coding in the enhancement layer bab, the bab type matrix and the motion vector matrix are resampled by repetition. In case of the motion vector matrix each components are scaled by the sampling ratio on both directions.

7.5.4.5 Temporal prediction

A motion compensated temporal prediction is made from the reference VOP in the enhancement layer. For predicting a bab, the motion vectors of the collocated bab in the lower layer is used. Enhancement layer shape motion vector [mvX, mvY] can be used by scaling the lower layer shape motion vector [mvXLower, mvYLower]. The scaling method is as follows:

mvX = (mvXLower * shape_hor_sampling_factor_n) // shape_hor_sampling_factor_m

mvY = (mvYLower * shape_ver_sampling_factor_n) // shape_ver_sampling_factor_m

7.5.4.6 Types for enhancement layer bab

Each bab belongs to one of four types given in Table V2 - 29. This enh_bab_type field influences decoding of further shape information of enhancement layers. For P-VOP coding only 0 and 1 of the enh_bab_type are allowed. For B-VOP coding scheme, all of the four modes are allowed.

Table V2 - 29 -- List of enh_bab_types and usage

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

When the enh_bab_type == 1, the bab is predicted by upsampling from the collocated block in the lower reference layer and is decoded by intra-mode CAE based on Scan Interleaving(SI).

The enh_bab_type == 0 means that the error of the predicted bab from the lower reference layer is less than the threshold. The predicted bab is given by upsamping the collocated block in the lower reference layer. The spatial prediction method in 7.5.4.4 or linear repetition method can be used for the upsampling. The predicted bab is used as the current bab.

For enh_bab_type==2 and enh_bab_type==3, no motion vector is decoded for the temporal prediction of enhancement layer bab. The motion vector of the collocated block in the lower reference layer can be utilized. Each component of the vector are scaled by two for referencing. If the collocated bab in the lower reference layer is intra coding mode, the value of the motion vector is set to zero. If enh_bab_type is decode as "inter NOT_CODED", then the error of the predicted bab from the previous VOP is less than the threshold. And the predicted bab is used as the current bab. If enh_bab_type is decode as "inter CODED", the current bab is decoded by inter-CAE which is used in non-scalable shape decoding.

7.5.4.7 Decoding bab types for enhancement layer

In order to decode the bab types, the collocated bab of the lower layer is also examined. Based on the lower layer bab types, the following VLC table is used for decoding the enh_bab_type field. For B-VOP decoding in enhancement layer enh_bab_type is decoded using in Table V2 - 31. For P-VOP decoding in enhancement layer, there are only two enh_bab_type fields, i.e., 0 and 1, and they are decoded by 1~2 bit described in Table V2 - 30.

Table V2 - 30 -- Decoding of enh_bab_type for P-VOP in enhancement layer

enh_bab_type	Code
0	1
1	01

Table V2 - 31 -- Decoding of enh_bab_type for B-VOP in enhancement layer

		enh_bab_type in the enhancement layer			
		0	1	2	3
enh_bab_type in the lower reference layer or (bab_type in the base layer)	0(2,3)	1	01	001	000
	1(4)	110	0	10	111
	2(0,1)	001	01	1	000
	3(5,6)	110	0	111	10

Note that the numbers 0-3 in these tables correspond to the enh_bab_types given in 7.8.2.1.6. When the lower layer is the base layer, {"MVDs==0 && No Update", "MVDs!=0 && No Update"}, {"all_0", "all_255"}, {"intraCAE"} and {"MVDs==0 && interCAE", "MVDs!=0 && interCAE"} modes are converted to the enh_bab_type 2, 0, 1 and 3, respectively.

7.5.4.8 Intra coded enhancement layer decoding

Intra coded enhancement layer decoding uses scan interleaving algorithm before performing intra-mode CAE. The decoding order with SI scanning is as follows:

1. Copy B from reference_layer

2. Decodingorder with Vertical scanning : Vr --> Vp1 -->Vp2 --> ... --> Vpk

3. Decodingorder with Horizontal scanning : Hr --> Hp1 --> Hp2 --> ... --> Hpl

where:

B: Pixel that can be copied from collocated pixel in the reference layer.V_r: Vertical scanning Pixel of Residual term of vertical linear-up sampling

V_{p2}: Vertical scanning Pixel of 2^l term of vertical linear-up sampling

H_r: Horizonta scanning Pixel of Residual term of Horzontal linear-up sampling

H_{p2}: Horizonta scanning Pixel of 2^k term of Horizontal linear-up sampling

k: The largest integer that is equal or smaller than
log$_2$(shape_hor_sampling_factor_n / shape_hor_sampling_factor_m)

l: The largest integer that is equal or smaller than
log$_2$(shape_ver_sampling_factor_n / shape_ver_samping_factor_m)

For an example, when shape_hor_sampling_factor_n = 8, shape_hor_sampling_factor_m = 3, shape_ver_sampling_factor_n = 5, shape_hor_sampling_factor_m =1, the scan order of pixels for decoding is shown in Figure V2 - 12.

Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2
Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1
Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2
Hr	Hr	Hr	Hr	Hr	Hr	Hr	Hr	Hr	Hr	Hr	Hr	Hr	Hr	Hr	Hr
Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2
Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1
Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2
Vp1	Vr	Vp1	B	Vp1	B	Vp1	B	Vp1	Vr	Vp1	B	Vp1	B	Vp1	B
Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2
Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1
Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2
Vp1	Vr	Vp1	B	Vp1	B	Vp1	B	Vp1	Vr	Vp1	B	Vp1	B	Vp1	B
Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2
Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1
Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2
Vp1	Vr	Vp1	B	Vp1	B	Vp1	B	Vp1	Vr	Vp1	B	Vp1	B	Vp1	B

Figure V2 - 12 – An example of decoding order when enh_bab_type==1
(Decoding order: B → V_r → V_{p1} → H_r → H_{p1} → H_{p2})

7.5.4.9 Border formation for intra-mode CAE for enhancement layer decoding

The border formation for decoding of intra CODED bab (enh_bab_type==1) is the same as that of the non-scalable shape coding (see subclause 7.5.2.5.2) except E and U regions in Figure 7-11. The border of E and U region of the current bordered bab contains the pixels from the collocated regions in the up-sampled lower layer plane and which consists of 2 lines in bottom and right of the current bab.

7.5.4.10 Scan Interleaving Algorithm

The spatial (or resolution) scalability can be easily sought by incorporating the so-called Scan Interleaving (SI) method. In SI, coded-scan-lines are decoded by using the prediction from the closest upper and lower neighboring two lines (reference-scan-lines) as shown in Figure V2 - 13. In the figure, horizontal direction scanning is described. The pixels on coded-scan-lines are the pixels to be decoded and the pixels on reference-scan-lines are the pixels that are already reconstructed. The closest upper and lower neighboring pixel values in reference-scan-lines are used for reconstructing the current pixel. (The closest upper and lower neighboring pixels mean the closest left and right neighboring pixels in vertical direction scanning case.) By using those two reference pixels, the current decoded pixel has three types as follows:

Transitional sample data (TSD): The closest upper and lower neighboring pixels that are already reconstructed have different colors each other.

Exceptional sample data (ESD): The closest upper and lower neighboring pixels that are already reconstructed are same and the current pixel is different with the neighboring pixels.

Predictable sample data (PSD): The closest upper and lower neighboring pixels that are already reconstructed are the same as the current pixel.

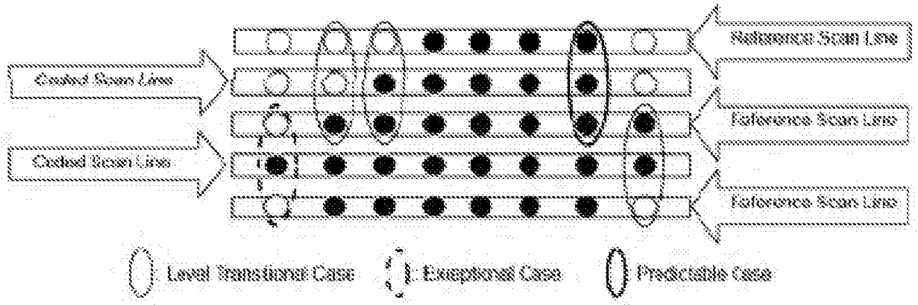


Figure V2 - 13 -- Scan Interleaving for Scalable Coding: an example of a times by 2 case

To encoding/decoding shape mask plane, SI peforms the scanning method in vertical and horizontal directions, successively. SI with vertical scanning (V-SI) and SI with horizontal scanning (H-SI) are separately applied to the input image.

For decoding a bab in the current layer, the spatially predicted bab from collocated block in the lower layer is enhanced by V-SI. And then H-SI is successively performed on the enhanced plane of the V-SI. The reconstructing order is presented in subclause 7.5.4.8.

In the first step of decoding a bab, a binary flag, which denotes the current bab type of SI, is decoded by BAC. There are two bab types of SI : transitional bab (TSD-B) and exceptional bab (ESD-B). If a bab has no ESD, the bab is set to TSD-B. Otherwise the bab is set to ESD-B. When the flag is '0', the current bab is TSD-B. Otherwise, the current bab is ESD-B.

In the encoder, for encoding TSD-B, only TSD pixel are encoded by CAE with the context shown in Figure V2 - 15. And for encoding ESD-B, all the pixels in the coded-scan-lines are encoded by CAE with the context shown in Figure V2 - 15. The encoding order is the same as shown in subclause 7.5.4.8.

[illegible]
[illegible]
Single user license only. Copying and networking prohibited.

If a bab is TSD-B, only TSD are decoded in the order of V-SI and H-SI. The specific decoding order is presented in 7.5.4.8. The decoding process of TSD-B is as follows.

Following the scanning order, if the current pixel is TSD, then the pixel is decoded by BAC with the context in Figure V2 - 15. And if the two neighbor pixels (the closest upper(left) and bottom(right)) are the same value, then the current pixel has the same value as its neighbor pixels.

If a bab is ESD-B, all the pixels in coded scan line are decoded by BAC with the context in Figure V2 - 15.

7.5.4.11 Determination of the scanning type in enhancement layer decoding

When enh_bab_type is 1 (intra CODED) and the scale ratio between the layers is square(i.e 200x300 <--> 400x600), before performing BAC, the scanning type should be determined. For the determination, the top and left borders of the collocated block of the lower layer are extended with its neighboring pixels like Figure V2 - 14. If an extended border is outside VOP, the pixel values of the extended border are set to zero. Two counts, N_L and N_U are calculated (N_L: the number of pixels which have different values from the left neighboring pixel value; N_U: the number of pixels which have different values from the upper neighboring value). If N_U is greater than N_L, the spatially predicted bab for decoding the enhancement layer bab has to be transposed and then the intra-mode CAE for enhancement layer is performed. Otherwise, the intra-mode CAE for enhancement layer is performed on the spatially predicted bab without transposing, that is default condition.

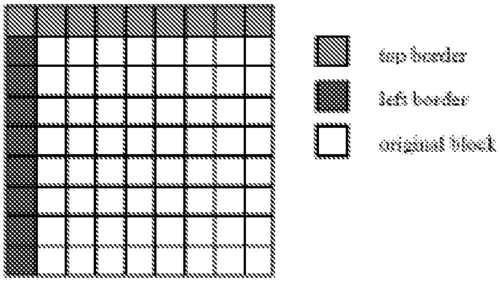


Figure V2 - 14 -- Extension of border pixels

7.5.4.12 Decoding of bab type of SI

The first decoded symbol of enh_binary_arithmetic_code(), which performs BAC, denotes the current bab type of SI. The probability of SI_bab_type_prob[] given in Annex B is used for decoding bab type of SI. When this flag is '0', the current bab is TSD-B. Otherwise, the current bab is ESD-B.

7.5.4.13 Decoding of intra-mode CAE for enhancement layer

For decoding intra mode CAE for enhancement layer, BAC using the V-SI and H-SI decoding method is performed. By using neighboring 7 bits context, a pixel is arithmetic decoded as in CAE, such that:

Compute a context using the templates shown in Figure V2 - 15 ((a) : for V-SI, (b): for H-SI)

Using the context, find probability from the probability table. i.e. enh_intra_v_prob[] for V-SI and enh_intra_h_prob[] for H-SI. These tables are given in annex B.

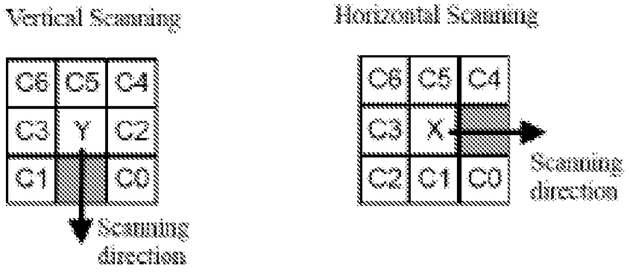
Decoding the pixel value using the probability.

The following subclause describes the computation of the contexts for intra mode CAE.

7.5.4.14 Contexts for intra-mode CAE for enhancement layer

Depending on the scanning direction, a 7 bit context $C=\sum_k c_k \cdot 2^k$ is built for each coded pixel as shown in Figure V2 - 15, where c_k=0 for transparent pixels and c_k=1 for opaque pixels. (a) of Figure V2 - 15 is for the intra-mode CAE with V-SI (SI by the vertical scanning), and (b) of Figure V2 - 15 is for the intra-mode CAE with H-SI (SI by the horizontal scanning). In (a) of the figure, C6, C5 and C4 are the closest decoded pixels in top-left, top, and top-right location, respectively. C3, and C2 are the closest decoded pixels in left and right location, respectively. And C1 and C0 are the closest decoded pixels in bottom-left and bottom-right location, respectively.

In (b) of the figure, C6, C5 and C4 are the closest decoded pixels in top-left, top, and top-right location, respectively. C3 is the closest decoded pixel in left location. And C2, C1 and C0 are the closest decoded pixels in bottom-left, bottom, and bottom-right location, respectively.



(a) (b)

Figure V2 - 15 – The context information at different scanning of SI

There are some special cases to consider:

When building contexts, any pixels outside the bounding box of the current VOP to the left and above are assumed to be zero (transparent).

When building contexts and error_resilient_disable==0, any pixels outside the space of the current video packet to the left and above are assumed to be zero (transparent).

When constructing the context, any pixel outside the bordered BAB is assumed to be the same pixel value of the closest pixel in the border of the current BAB from the context location.

7.5.4.15 Decoding of Inter-mode CAE for enhancement layer

For decoding of enhancement layer with B-VOP coding, the inter-mode CAE is performed in the case of the "inter coded" mode (enh_bab_type==3). The context information in Figure V2 - 16 is utilized for this decoding. The decoding process of a pixel is done in the same way as non-scalable binary shape coding.

When constucting the Inter-mode CAE context the following conditional assignment is performed.

```
if(c1 is unknown) c1=c2
```

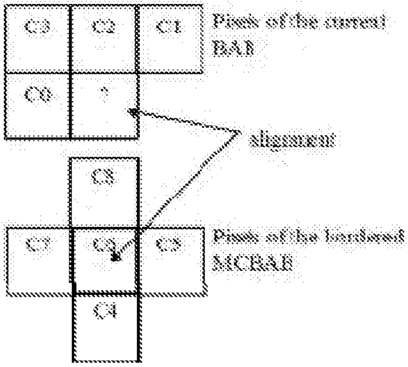


Figure V2 - 16 – The templates of Inter-mode CAE for enhancement layer. C6 is aligned with the pixel to be decoded. The pixel to be decoded is marked with '?'.

7.5.4.16 Quality (SNR) Scalable Coding

Quality Scalability can be easily sought by spatial scalability. Using the spatial scalability and spatial prediction, binary shape can achieve the quality (SNR) scalability. In the decoding process, each spatial scalable layer can be spatially predicted to the full resolution. For the binary shape, this scalable structure is equivalent to quality scalable structure.

7.5.5 Grayscale Shape Decoding

Grayscale alpha plane decoding is achieved by the separate decoding of a support region and the values of the alpha channel. The support region is transmitted by using the binary shape as described above. The alpha values are transmitted as texture data with arbitrary shape, using almost the same coding method as is used for the luminance texture channel.

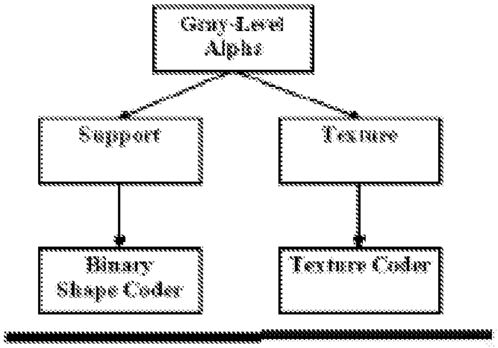


Figure 7-15 – Grayscale shape coding

All samples which are indicated to be transparent by the binary shape data, must be set to zero in the decoded grayscale alpha plane. Within the VOP, alpha samples have the values produced by the grayscale alpha decoding process. Decoding of binary shape information is not dependent on the decoding of grayscale alpha. The alpha values are decoded into 16x16 macroblocks in the same way as the luminance channel (see subclauses 7.4 and 7.6, and 7.8.7). The 16x16 blocks of alpha values are referred to as alpha macroblocks hereafter. The data for each alpha macroblock is present in the bitstream immediately following the texture data for the corresponding texture macroblock. Any aspect of alpha decoding that is not covered in this document should be assumed to be the same as for the decoding of luminance.

7.5.5.1 Grayscale Alpha COD Modes

When decoding grayscale alpha macroblocks, CODA is first encountered and indicates the coding status for alpha. It is important to understand that the macroblock syntax elements for alpha are still present in the bitstream for P, B, or S(GMC) macroblocks even if the texture syntax elements indicate "not-coded" (not_coded='1'). In this respect, the decoding of the alpha and texture data are independent. The only exception is for B-VOPs when the colocated P-VOP texture macroblock is skipped. In this case, no syntax is transmitted for texture or grayscale alpha, as both types of macroblock are skipped.

For macroblocks which are completely transparent (indicated by the binary shape coding), no alpha syntax elements are present and the grayscale alpha samples must all be set to zero (transparent). If CODA="all opaque" [I, P, B, or S(GMC) macroblocks] or CODA="not coded" [P, B, or S(GMC) macroblocks] then no more alpha data is present. Otherwise, other alpha syntax elements follow, including the coded block pattern (cbpa), followed by alpha texture data for those 8x8 blocks which are coded and non-transparent, as is the case for regular luminance macroblock texture data.

When CODA="all opaque", the corresponding decoded alpha macroblock is filled with a constant value of 255. This value will be called AlphaOpaqueValue.

7.5.5.2 Alpha Plane Scale Factor

For both binary and grayscale shape, the VOP header syntax element "vop_constant_alpha" can be used to scale the alpha plane. If this bit is equal to '1', then each pixel in the decoded VOP is scaled before output, using vop_constant_alpha_value. The scaling formula is:

scaled_pixel = (original_pixel * (vop_constant_alpha_value + 1)) / 256

Scaling is applied at the output of the decoder, such that the decoded original values, not the scaled values are used as the source for motion compensation.

7.5.5.3 Gray Scale Quantiser

When no_gray_quant_update is equal to "1", the grayscale alpha quantiser is fixed for all macroblocks to the value indicated by vop_alpha_quant. Otherwise, the grayscale quantiser is reset at each new macroblock to a value that depends on the current texture quantiser (after any update by dquant). The relation is:

current_alpha_quant = (current_texture_quant * vop_alpha_quant) / vop_quant

The resulting value of current_alpha_quant must then be clipped so that it never becomes less than 1.

7.5.5.4 Intra Macroblocks

When the texture mb_type indicates an intra macroblock in an I-, P-, or S(GMC)-VOP, the grayscale alpha data is also decoded using intra mode.

The intra dc value is decoded in the same way as for luminance, using the same non-linear transform to convert from alpha_quant to DCScalarA. However, intra_dc_vlc_thr is not used for alpha, and therefore AC coefficient VLCs are never used to code the differential intra dc coefficient.

DC prediction is used in the same way as for luminance. However, when coda_i indicates that a macroblock is all opaque, a synthetic intra dc value is created for each block in the current macroblock so that adjacent macroblocks can correctly obtain intra dc prediction values. The synthetic intra dc value is given as:

BlockIntraDC = (((AlphaOpaqueValue * 8) + (DcScalerA>>1)) / DcScalerA) * DcScalerA

AlphaOpaqueValue is described in subclause 7.5.4.1.

The intra cbpa VLC makes use of the inter cbpy VLC table, but the intra alpha block DCT coefficients are decoded in the same manner as with luminance intra macroblocks.

When interlaced is equal to "1", alternate scan should not be applied to coding of grey-level alpha.

When both interlaced is equal to "1" and video_object_layer_shape is equal to "11" (grayscale), only the frame DCT should be appied to coding of gray-level alpha.

7.5.5.5 Inter Macroblocks and Motion Compensation

Motion compensaton is carried out for P-, B-, and S(GMC)-VOPs, using the 8x8 or 16x16 luminance motion vectors, or the warping of the previous decoded VOP, in the same way as for luminance data, except that regular motion compensation is used instead of OBMC. Forward, backward, bidirectional and direct mode motion compensation are used for B-VOPs. Where the luminance motion vectors are not present because the texture macroblock is skipped, the exact same style of non-coded motion compensation used for luminance is applied to the alpha data (but without OBMC). Note that this does not imply that the alpha macroblock is skipped, because an error signal to update the resulting motion compensated alpha macroblock may still be present if indicated by coda_pb. When the colocated P-VOP texture macroblock is skipped for B-VOPs, then the alpha macroblock is assumed to be skipped with no syntax transmitted.

cbpa and the alpha inter DCT coefficients are decoded in the same way as with luminance cbpy and inter DCT cofficients.

When both interlaced is equal to "1" and video_object_layer_shape is equal to "11" (grayscale), the field padding should be applied to coding of gray-level alpha.

When both interlaced is equal to "1" and video_object_layer_shape is equal to "11" (grayscale), both frame and field motion compensation are applicable in coding of texture and gray-level alpha, but only the frame DCT should be applied to coding of gray-level alpha.

7.5.5.6 Method to be used when blending with greyscale alpha signal

The following explains the blending method to be applied to the video object in the compositor, which is controlled by the composition_method flag and the linear_composition flag. The linear_composition flag is informative only, and the decoder may ignore it and proceed as if it had the value 0. However, it is normative that the composition_method flag be acted upon.

The descriptions below show the processing taking place in YUV space; note that the processing can of course be implemented in RGB space to obtain equivalent results.

composition_method=0 (cross-fading)

If layer N, with an n-bit alpha signal, is overlaid over layer M to generate a new layer P, the composited Y, U, V and alpha values are:

$$\mathrm{Pyuv} = ((2^n-1 - \mathrm{Nalpha}) * \mathrm{Myuv} + (\mathrm{Nalpha} * \mathrm{Nyuv})) / (2^n-1)$$

$$\mathrm{Palpha} = (2^n-1)$$

composition_method=1 *(Additive mixing)*

If layer N, with an n-bit alpha signal, is overlaid over layer M to generate a new layer P, the composited Y, U, V and alpha values are:

$$\mathrm{Pyuv} = \begin{cases} \mathrm{Myuv} & \text{.....} \ \mathrm{Nalpha} = 0 \\ (\mathrm{Myuv} - \mathrm{BLACK}) - ((\mathrm{Myuv} - \mathrm{BLACK}) * \mathrm{Nalpha} \ / (2^n-1)) + \mathrm{Nyuv} & \text{....} \ \mathrm{Nalpha} > 0 \end{cases}$$

(this is equivaent to Pyuv = Myuv*(1-alpha) + Nyuv, taking account of black level and the fact that the video decoder does not produce an output in areas where alpha=0)

$Palpha = Nalpha + Malpha - (Nalpha*Malpha) / (2^n-1)$

where

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

NOTE The compositor must convert foreground and backgound 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 14496-1:1999).

- linear_composition = 0: The compositing process is carried out using the video signal in the format from which it is produced by the video decoder, that is, without converting to linear signals. Note that because video signals are usually non-linear ("gamma-corrected"), the composition will be approximate.
- linear_composition = 1: The compositing process is carried out using linear signals, so the output of the video decoder is converted to linear if it was originally in a non-linear form, as specified by the video_signal_type field. Note that the alpha signal is always linear, and therefore requires no conversion.

7.5.6 Multiple Auxiliary Component Decoding

Auxiliary components are defined for the VOP on a pixel-by-pixel basis, and contain data related to the video object, like disparity, depth, additional texture. Up to 3 auxiliary components (including the grayscale shape) are possible. The number and type of these components is indicated by the video_object_layer_shape_extension given in Table V2 - 1. For example, a value '0000' indicates the grayscale (alpha) shape. The same support region as described in 7.5.4 is used for all auxiliary components, and the decoding procedure is the same as described in subclauses 7.5.4.1 - 7.5.4.6.

7.6 Motion compensation decoding

In order to perform motion compensated prediction on a per VOP basis, a special padding technique, i.e. the macroblock-based repetitive padding, is applied for the reference VOP. The details of these techniques are described in the following subclauses.

Since a VOP may have arbitrary shape, and this shape can change from one instance to another, conventions are necessary to ensure the consistency of the motion compensation process.

The absolute (frame) coordinate system is used for referencing every VOP. At every given instance, a bounding rectangle that includes the shape of that VOP, as described in subclause 7.5, is defined. The left and top corner, in the absolute coordinates, of the bounding rectangle is decoded from VOP spatial reference. Thus, the motion vector for a particular feature inside a VOP, e.g. a macroblock, refers to the displacement of the feature in absolute coordinates. No alignment of VOP bounding rectangles at different time instances is performed.

For motion compensation, the amplitude resolution of the motion vectors is signaled by the quarter_sample flag in the VOL header, see subclause 6.2.3. Either ½ pel resolution ("half sample mode", quarter_sample==0) or ¼ pel ("quarter sample mode", quarter_sample==1) is used. For ½ pel resolution, the necessary ½ pel samples are calculated by means of half sample interpolation as described in subclause 7.6.2.1. For ¼ pel resolution, the necessary ½ and ¼ pel samples are calculated by means of half and quarter sample interpolation as described in subclause 7.6.2.2.

In addition to the above motion compensation processing, three additional processes are supported, namely, unrestricted motion compensation, four MV motion compensation,and overlapped motion compensation. Note that in all three modes, macroblock-based padding of the arbitrarily shaped reference VOP is performed for motion compensation.

7.6.1 Padding process

The padding process defines the values of luminance and chrominance samples outside the VOP for prediction of arbitrarily shaped objects. Figure 7-16 shows a simplified diagram of this process.

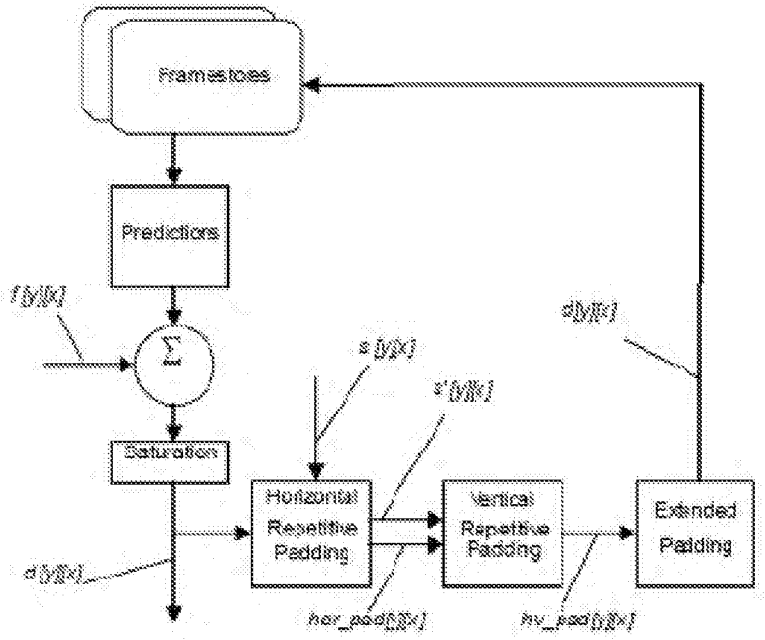


Figure 7-16 – Simplified padding process

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. A macroblock that lies on the VOP boundary (hereafter referred to as a boundary macroblock) is padded by replicating the boundary samples of the VOP towards the exterior. This process is divided into horizontal repetitive padding and vertical repetitive padding. The remaining macroblocks that are completely outside the VOP (hereafter referred to as exterior macroblocks) are filled by extended padding.

NOTE The padding process is applied to all macroblocks inside the bounding rectangle of a VOP. The bounding rectangle of the luminance component is defined by vop_width and vop_height extended to multiple of 16, while that of the chrominance components is defined by (vop_width>>1) and (vop_height>>1) extended to multiple of 8.

7.6.1.1 Horizontal repetitive padding

Each sample at the boundary of a VOP is replicated horizontally to the left and/or right direction in order to fill the transparent region outside the VOP of a boundary macroblock. If there are two boundary sample values for filling a sample outside of a VOP, the two boundary samples are averaged (//2).

hor_pad[y][x] is generated by any process equivalent to the following example. For every line with at least one shape sample *s[y][x]* == 1(inside the VOP):

```
for (x=0; x<N; x++) {
    if (s[y][x] == 1) { hor_pad[y][x] = d[y][x]; s'[y][x] = 1; }
    else {
        if ( s[y][x'] == 1 && s[y][x''] == 1) {
            hor_pad[y][x] = (d[y][x']+ d[y][x''])//2;
            s'[y][x] = 1;
        } else if ( s[y][x'] == 1 ) {
            hor_pad[y][x] = d[y][x']; s'[y][x] = 1;
        } else if ( s[y][x''] == 1 ) {
            hor_pad[y][x] = d[y][x'']; s'[y][x] = 1;
        }
    }
}
```

where *x'* is the location of the nearest valid sample (*s[y][x']* == 1) at the VOP boundary to the left of the current location *x*, *x''* is the location of the nearest boundary sample to the right, and *N* is the number of samples of a line in a macroblock. *s'[y][x]* is initialized to 0.

7.6.1.2 Vertical repetitive padding

The remaining unfilled transparent horizontal samples (where *s'[y][x]* == 0) from subclause 7.6.1.1 are padded by a similar process as the horizontal repetitive padding but in the *vertical* direction. The samples already filled in subclause 7.6.1.1 are treated as if they were inside the VOP for the purpose of this vertical pass.

hv_pad[y][x] is generated by any process equivalent to the following example. For every column of *hor_pad[y][x]* :

```
for (y=0; y<M; y++) {
    if (s'[y][x] == 1)
        hv_pad[y][x] =hor_pad[y][x];
    else {
        if ( s'[y'][x] == 1 && s'[y''][x]== 1 )
            hv_pad[y][x] = (hor_pad[y'][x] + hor_pad[y''][x])//2;
        else if ( s'[y'][x] == 1 )
            hv_pad[y][x] = hor_pad[y'][x];
        else if (s'[y''][x] == 1 )
            hv_pad[y][x] = hor_pad[y''][x];
    }
}
```

where *y'* is the location of the nearest valid sample (*s'[y'][x]* == 1) above the current location *y* at the boundary of *hv_pad*, *y''* is the location of the nearest boundary sample below *y*, and *M* is the number of samples of a column in a macroblock.

7.6.1.3 Extended padding

Exterior macroblocks immediately next to boundary macroblocks are filled by replicating the samples at the border of the boundary macroblocks. Note that the boundary macroblocks have been completely padded in subclause 7.6.1.1 and subclause 7.6.1.2. If an exterior macroblock is next to more than one boundary macroblocks, one of the macroblocks is chosen, according to the following convention, for reference.

The boundary macroblocks surrounding an exterior macroblock are numbered in priority according to Figure 7-17. The exterior macroblock is then padded by replicating upwards, downwards, leftwards, or rightwards the row of samples from the horizontal or vertical border of the boundary macroblock having the largest priority number.

The remaining exterior macroblocks (not located next to any boundary macroblocks) are filled with $2^{bits\_per\_pixel-1}$. For 8-bit luminance component and associated chrominance this implies filling with 128.

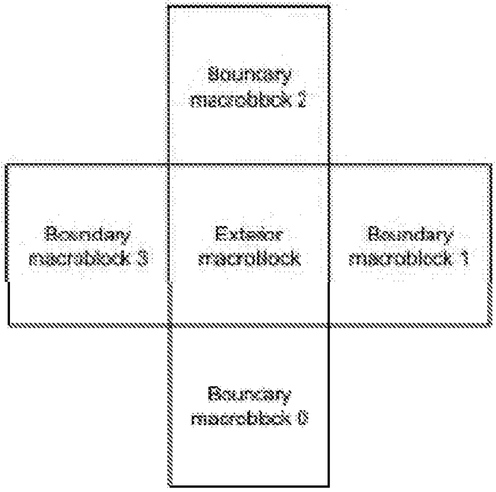


Figure 7-17 -- Priority of boundary macroblocks surrounding an exterior macroblock

7.6.1.4 Padding for chrominance components

Chrominance components are padded according to subclauses 7.6.1.1 through 7.6.1.3 for each 8x8 block. The padding is performed by referring to a shape block generated by decimating the shape block of the corresponding luminance component. This decimating of the shape block is performed by the subsampling process described in subclause 6.1.3.6.

7.6.1.5 Padding of interlaced macroblocks

Macroblocks of interlaced VOP (interlaced = 1) are padded according to subclauses 7.6.1.1 through 7.6.1.3. The vertical padding of the luminance component, however, is performed for each field independently. A sample outside of a VOP is therefore filled with the value of the nearest boundary sample of the same field. Completely transparent blocks are padded with $2^{bits\_per\_pixel-1}$. Chrominance components of interlaced VOP are padded according to subclause 7.6.1.4, however, based on fields to enhance subjective quality of display in 4:2:0 format. The padding method described in this subclause is not used outside the bounding rectangle of the VOP.

7.6.1.6 Vector padding technique

The vector padding technique is applied to generate the vectors for the transparent blocks within a non-transparent macroblock, for an INTRA-coded macroblock and for a skipped macroblock included in a P-VOP. It works in a similar way as the horizontal followed by the vertical repetitive padding, and can be simply regarded as the repetitive padding performed on a 2x2 block except that the padded values are two dimensional vectors. A macroblock has four 8x8 luminance blocks, let {MVx[i], MVy[i], i=0,1,2,3} and {Transp[i], i=0,1,2,3} be the vectors and the transparencies of the four 8x8 blocks, respectively, the vector padding is any process numerically equivalent to:

```
if (the macroblock is INTRA-coded, or skipped and included in a P-VOP) {
    MVx[0] = MVx[1] = MVx[2] = MVx[3] = 0
    MVy[0] = MVy[1] = MVy[2] = MVy[3] = 0
} else {
    if(Transp[0] == TRANSPARENT) {
        MVx[0]=(Transp[1] != TRANSPARENT) ? MVx[1] :((Transp[2]!=TRANSPARENT) ?
        MVx[2]:MVx[3]));
        MVy[0]=(Transp[1] != TRANSPARENT) ? MVy[1]:((Transp[2]!=TRANSPARENT) ?
        MVy[2]:MVy[3]));
    }
if(Transp[1] == TRANSPARENT) {
    MVx[1]=(Transp[0] != TRANSPARENT) ? MVx[0] :((Transp[3]!=TRANSPARENT) ?      MVx[3]:MVx[2]));
    MVy[1]=(Transp[0] != TRANSPARENT) ? MVy[0]:((Transp[3]!=TRANSPARENT) ? MVy[3]:MVy[2]));
}
if(Transp[2] == TRANSPARENT) {
    MVx[2]=(Transp[3] != TRANSPARENT) ? MVx[3] :((Transp[0]!=TRANSPARENT) ?      MVx[0]:MVx[1]));
    MVy[2]=(Transp[3] != TRANSPARENT) ? MVy[3]:((Transp[0]!=TRANSPARENT) ? MVy[0]:MVy[1]));
}
if(Transp[3] == TRANSPARENT) {
    MVx[3]=(Transp[2] != TRANSPARENT) ? MVx[2] :((Transp[1]!=TRANSPARENT) ?      MVx[1]:MVx[0]));
    MVy[3]=(Transp[2] !=TRANSPARENT) ? MVy[2]:((Transp[1]!=TRANSPARENT) ?  MVy[1]:MVy[0]));
    }
}
```

Vector padding is only used in I-, P-, and S(GMC)-VOPs, it is applied on a macroblock directly after it is decoded. The block vectors after padding are used in the P-, and S(GMC)-VOP vector decoding and binary shape decoding, and in the B-VOP direct mode decoding. Note that the averaged motion vector described in 7.8.7.3 is used as the motion vectors of a GMC macroblock (i.e. a macroblock included in an S (GMC)-VOP and mcsel == '1') for this padding process.

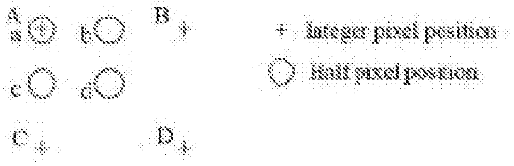
7.6.2 Sample interpolation for non-integer motion vectors

The sample interpolation in case of non-integer motion vectors is performed for 16x16 or 8x8 frame blocks in case of progressive and for 16x8 field blocks in case of interlaced macroblocks. It is done using the luminance and chrominance components of the respectively reconstructed and padded reference VOP.

The value for the quarter_sample flag is defined in he VOL header (see subclause 6.2.4), the value of rounding_control is defined using the vop_rounding_type bit in the vop header (see subclause 6.3.5). Note that the samples outside the padded region cannot be used for interpolation. For interlaced video, the half and quarter sample values are vertically interpolated between two successive lines of the same field. The field motion vectors are given in frame coordinates; that is the vertical coordinates of the integer samples differ by 2.

7.6.2.1 Half sample mode interpolation

The process for interpolation of half sample values defined in this subclause is carried out only in half sample mode, i.e. when quarter_sample==0. Here, the half sample values are calculated by bilinear interpolation as depicted in Figure 7-18. The value of rounding_control is defined using the vop_rounding_type bit in the VOP header (seesubclause 6.3.5). Note that the samples outside the padded region cannot be used for interpolation.



a = A,
b = (A + B + 1 - rounding_control) / 2
c = (A + C + 1 - rounding_control) / 2,
d = (A + B + C + D + 2 - rounding_control) / 4

Figure 7-18 -- Interpolation scheme for half sample search

7.6.2.2 Quarter sample mode interpolation

The process for interpolation of half and quarter sample values defined in this subclause is carried out only in quarter sample mode, i.e. when quarter_sample==1. Further, the process is only carried out for the luminance component of a VOP; for motion compensation of the chrominance components, the respective motion vectors are rounded to the next half pel position (see Table 7-9) and then processed as described in half pel sample interpolation (subclause 7.6.2.1).

In quarter sample mode interpolation, for each block of size MxN in the reference VOP which position is defined by the decoded motion vector for the block to be predicted, a reference block of size (M+1)x(N+1) biased in the direction of the half or quarter sample position is read from the reconstructed and padded reference VOP. Then, this reference block is symmetrically extended at the block boundaries by three samples using block boundary mirroring according to Figure V2 - 17.

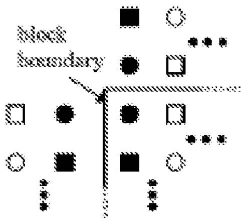


Figure V2 - 17 -- block boundary mirroring

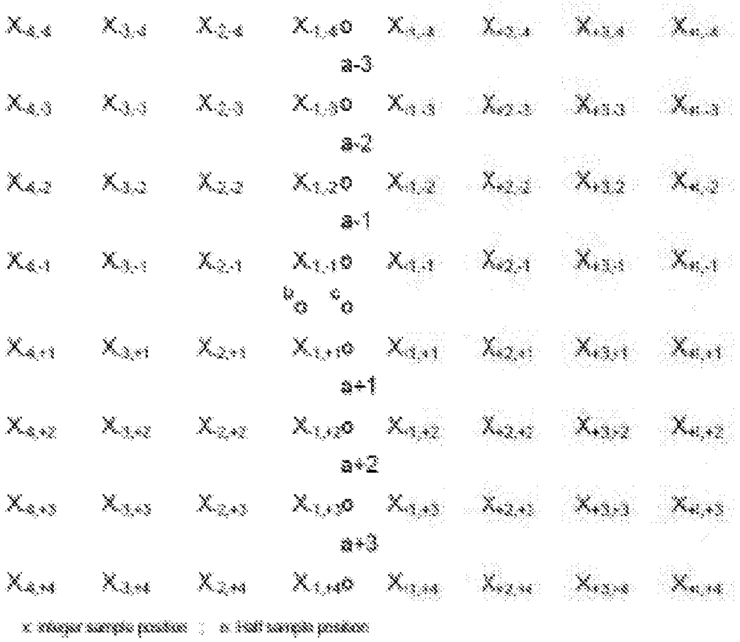
The half sample values and (if applicable) the quarter sample values are then calculated from this extended reference block in two steps described in the following.

7.6.2.2.1 Calculation of the half sample values

The half sample values are calculated by horizontal filtering (a in Figure V2 - 18) and subsequent vertical filtering (b and c in Figure V2 - 18) according to Figure V2 - 18. For the filtering in both directions an 8-tap FIR interpolation filter describedby

(C01[4] ,C01[3],C01[2],C01[1],C01[1],C01[2] ,C01[3] ,C01[4])/256

with the filter coefficients C01[1..4]= [160, -48, 24, -8] is used.


X-4,-4 X-3,-4 X-2,-4 X-1,-4 o X+1,-4 X+2,-4 X+3,-4 X+4,-4
a-3
X-4,-3 X-3,-3 X-2,-3 X-1,-3 o X+1,-3 X+2,-3 X+3,-3 X+4,-3
a-2
X-4,-2 X-3,-2 X-2,-2 X-1,-2 o X+1,-2 X+2,-2 X+3,-2 X+4,-2
a-1
X-4,-1 X-3,-1 X-2,-1 X-1,-1 o X+1,-1 X+2,-1 X+3,-1 X+4,-1
b o c o
X-4,+1 X-3,+1 X-2,+1 X-1,+1 o X+1,+1 X+2,+1 X+3,+1 X+4,+1
a+1
X-4,+2 X-3,+2 X-2,+2 X-1,+2 o X+1,+2 X+2,+2 X+3,+2 X+4,+2
a+2
X-4,+3 X-3,+3 X-2,+3 X-1,+3 o X+1,+3 X+2,+3 X+3,+3 X+4,+3
a+3
X-4,+4 X-3,+4 X-2,+4 X-1,+4 o X+1,+4 X+2,+4 X+3,+4 X+4,+4
x: integer sample position ; o: Half sample position

$$a_i = \left(\left(\sum_{j=1}^{4} COI[j] \cdot (X_{-j,i} + X_{+j,i}) \right) + 128 - rounding\_control \right) / 256$$

$$b = \left(\left(\sum_{j=1}^{4} COI[j] \cdot (X_{-1,-j} + X_{-1,+j}) \right) + 128 - rounding\_control \right) / 256$$

$$c = \left(\left(\sum_{j=1}^{4} COI[j] \cdot (a_{-j} + a_{+j}) \right) + 128 - rounding\_control \right) / 256$$

Figure V2 - 18 – Half sample interpolation in quarter sample mode (FIR filter)

The values generated by both horizontal and vertical filtering pass are clipped to the range of $[0, 2^{N\_bit}-1]$. The value of rounding_control is defined using the vop_rounding_type bit in the vop header (see subclause 6.3.5).

7.6.2.2.2 Calculation of the quarter sample values

If applicable, the quarter sample values are calculated in a second step following the half sample interpolation described above. Here, bilinear interpolation between the corresponding half and integer sample values is carried out as described in Figure 7-18.

7.6.3 General motion vector decoding process

To decode a motion vector (MVx, MVy), the differential motion vector (MVDx, MVDy) is extracted from the bitstream by using the variable length decoding as described in subclause 6.3.6.2. Then it is added to a motion vector predictor (Px, Py) component wise to form the final motion vector. The general motion vector decoding process is any process that is equivalent to the following one. All calculations are carried out in halfpel units in the following. Please note that if the quarter sample mode is enabled (quarter_sample==1), the resulting motion vectors are still given in half pel units which absolute values need then however no longer be integer. This process

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA V. ABAWANE
ANSI Store order #X374451 Downloaded: 7/18/2007 3:56:21 PM ET
Single user license only. Copying and networking prohibited.

is generic in the sense that it is valid for the motion vector decoding in interlaced/progressive P-, S(GMC)- and B-VOPs except that the generation of the predictor (Px, Py) may be different.

```
r_size = vop_fcode - 1
f = 1 << r_size
high = ( 32 * f ) - 1;
low = ( (-32) * f );
range = ( 64 * f );

if ( (f == 1) || (horizontal_mv_data == 0) )
    MVDx = horizontal_mv_data;
else {
    MVDx = ( ( Abs(horizontal_mv_data) - 1 ) * f ) + horizontal_mv_residual + 1;
    if (horizontal_mv_data < 0)
        MVDx = - MVDx;
}

if ( (f == 1) || (vertical_mv_data == 0) )
    MVDy = vertical_mv_data;
else {
    MVDy = ( ( Abs(vertical_mv_data) - 1 ) * f ) + vertical_mv_residual + 1;
    if (vertical_mv_data < 0)
        MVDy = - MVDy;
}

if(quarter_sample==1) {
    MVDx = MVDx / 2.0;
    MVDy = MVDy / 2.0;
}

MVx = Px + MVDx;
if ( MVx < low )
    MVx = MVx + range;
if (MVx > high)
    MVx = MVx - range;

MVy = Py + MVDy;
if ( MVy < low )
    MVy = MVy + range;
if (MVy > high)
    MVy = MVy - range;
```

The value of mv_data (i.e., horizontal_mv_data and vertical_mv_data) is equal to two times the value found in the 'vector differences' column of Table B-12 associated with the received codeword.

The parameters in the bitstream shall be such that the components of the reconstructed differential motion vector, MVDx and MVDy, shall lie in the range [low:high], at the time of their use in calculating the values of MVx and MVy (i.e., intermediate values of MVDx and MVDy may occur that are outside the range [low:high]). In addition the components of the reconstructed motion vector, *MVx* and *MVy*, shall also lie in the range [*low* : *high*]. The allowed range [low : high] for the motion vectors depends on the parameter vop_fcode; it is shown in Table 7-5.

The variables *r_size*, *f*, *MVDx*, *MVDy*, *high* , *low* and *range* are temporary variables that are not used in the remainder of this part of ISO/IEC 14496. The parameters horizontal_mv_data, vertical_mv_data, horizontal_mv_residual and vertical_mv_residual are parameters recovered from the bitstream.

The variable *vop_fcode* refers either to the parameter vop_fcode_forward or to the parameter vop_fcode_backward which have been recovered from the bitstream, depending on the respective prediction mode. In the case of P- or S(GMC)-VOP prediction only forward prediction applies. In the case of B-VOP prediction, forward as well as backward prediction may apply.

Table 7-5 -- Range for motion vectors

vop_fcode_forward or vop_fcode_backward	motion vector range in halfsample units [low:high]	motion vecter range in halfsample units (quarter_sanple==1) [low:high]
1	[-32,31]	[-16,15.5]
2	[-64,63]	[-32,31.5]
3	[-128,127]	[-64,63.5]
4	[-256,255]	[-128,127.5]
5	[-512,511]	[-256,255.5]
6	[-1024,1023]	[-512,511.5]
7	[-2048,2047]	[-1024,1023.5]

If the current macroblock is a field motion compensated macroblock, then the same prediction motion vector (Px, Py) is used for both field motion vectors. Because the vertical component of a field motion vector is integral, the vertical differential motion vector encoded in the bitstream is

$$MVy = MVDy_{field} + PY / 2$$

7.6.4 Unrestricted motion compensation

Motion vectors are allowed to point outside the decoded area of a reference VOP when (and only when) the short video header format is not in use (i.e., when short_video_header is 0). For an arbitrary shape VOP, the decoded area refers to the area within the bounding rectangle, padded as described in subclause 7.6.1. A bounding rectangle is defined by vop_width and vop_height extended to multiple of 16. In case of half sample mode, when a sample referenced by a motion vector stays outside the decoded VOP area, an edge sample is used. This edge sample is retrieved by limiting the motion vector to the last full pel position inside the decoded VOP area. Limitation of a motion vector is performed on a sample basis and separately for each component of the motion vector, as depicted in Figure 7-19. In case of quarter sample mode, when a sample needed for interpolation (see subclause 7.6.2.2) stays outside the decoded VOP area, an edge sample is used prior to block boundary mirroring.

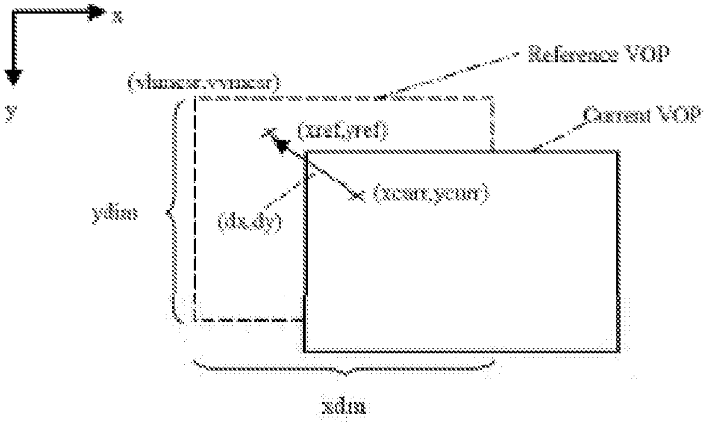


Figure 7-19 -- Unrestricted motion compensation

The coordinates of a reference sample in the reference VOP, (yref, xref) is determined as follows :

xref = MIN (MAX (xcurr+dx, vhmcsr), xdim+vhmcsr-1)

yref = MIN (MAX (ycurr+dy, vvmcsr), ydim+vvmcsr-1)

where vhmcsr = vop_horizontal_mc_spatial_ref, vvmcsr = vop_vertical_mc_spatial_ref, (ycurr, xcurr) are the coordinates of a sample in the current VOP, (yref, xref) are the coordinates of a sample in the reference VOP, (dy, dx) is the motion vector, and (ydim, xdim) are the dimensions of the bounding rectangle of the reference VOP. All coordinates are related to the absolute coordinate system shown in Figure 7-10. Note that for rectangular VOP, a reference VOP is defined by video_object_layer_width and video_object_layer_height. For an arbitrary shape VOP, a reference VOP of luminance is defined by vop_width and vop_height extended to multiple of 16, while that of chrominance is defined by (vop_width>>1) and (vop_height>>1) extended to multiple of 8.

7.6.5 Vector decoding processing and motion-compensation in progressive P- and S(GMC)-VOP

An inter-coded macroblock comprises either one motion vector for the complete macroblock or K (1< K<=4) motion vectors, one for each non-transparent 8x8 pel blocks forming the 16x16 pel macroblock, as is indicated by the mcbpc code.

For decoding a motion vector, the horizontal and vertical motion vector components are decoded differentially by using a prediction, which is formed by a median filtering of three vector candidate predictors (MV1, MV2, MV3) from the spatial neighbourhood macroblocks or blocks already decoded. Note that if any spatial neighbourhood macroblock is a GMC macroblock (i.e. included in an S (GMC)-VOP and mcsel == '1') and it is in the current VOP and video packet, the averaged motion vector described in subclause 7.8.7.3 is used as candidate motion vectors. The spatial position of candidate predictors for each block vector is depicted in Figure 7-20. In the case of only one motion vector present for the complete macroblock, the top-left case in Figure 7-20 is applied. When the short video header format is in use (i.e., when short_video_header is "1"), only one motion vector shall be present for a macroblock.

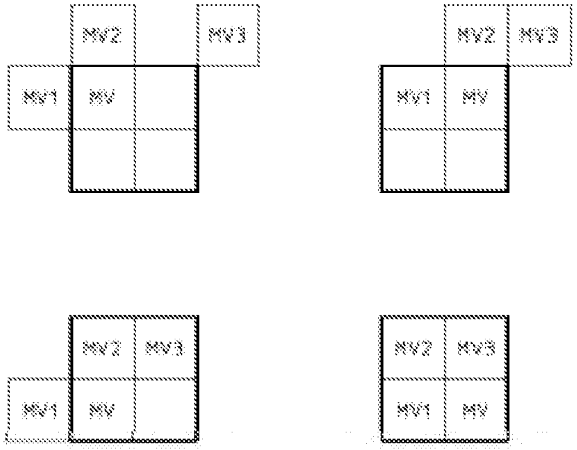


Figure 7-20 -- Definition of the candidate predictors MV1, MV2 and MV3 for each of the luminance blocks in a macroblock

The following four decision rules are applied to obtain the value of the three candidate predictors:

If a candidate predictor MVi is in a transparent spatial neighbourhood macroblock or in a transparent block of the current macroblock it is not valid, otherwise, it is set to the corresponding block vector.

© ISO/IEC 2001 – All rights reserved
Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/[illegible]
Single user licence only. Copying and networking prohibited.

If one and only one candidate predictor is not valid, it is set to zero.

If two and only two candidate predictors are not valid, they are set to the third candidate predictor.

If all three candidate predictors are not valid, they are set to zero.

Note that any neighbourhood macroblock outside the current VOP or video packet or outside the current GOB (when short_video_header is "1") for which gob_header_empty is "0" is treated as transparent in the above sense. The median value of the three candidates for the same component is computed as predictor, denoted by Px and Py:

$$Px = Median(MV1x, MV2x, MV3x)$$

$$Py = Median(MV1y, MV2y, MV3y)$$

For instance, if MV1=(-2,3), MV2=(1,5) and MV3=(-1,7), then Px = -1 and Py = 5. The final motion vector is then obtained by using the general decoding process defined in the subclause 7.6.3.

If four vectors are used, each of the motion vectors is used for all pixels in one of the four luminance blocks in the macroblock. The numbering of the motion vectors is equivalent to the numbering of the four luminance blocks as given in Figure 6-5. Motion vector MVD_{CHR} for both chrominance blocks is derived by calculating the sum of the *K* luminance vectors, that corresponds to *K* 8x8 blocks that do not lie outside the VOP shape and dividing this sum by 2**K*; in quarter sample mode the vectors are divided by 2 before summation. The component values of the resulting sixteenth/twelfth/eighth/fourth sample resolution vectors are modified towards the nearest half sample position as indicated below.

Table 7-6 -- Modification of sixteenth sample resolution chrominance vector components

sixteenth pixel position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	//16
resulting position	0	0	0	1	1	1	1	1	1	1	1	1	1	1	2	2	//2

Table 7-7 -- Modification of twelfth sample resolution chrominance vector components

twelfth pixel position	0	1	2	3	4	5	6	7	8	9	10	11	//12
resulting position	0	0	0	1	1	1	1	1	1	1	2	2	//2

Table 7-8 -- Modification of eighth sample resolution chrominance vector components

eighth pixel position	0	1	2	3	4	5	6	7	//8
resulting position	0	0	1	1	1	1	1	2	//2

Table 7-9 -- Modification of fourth sample resolution chrominance vector components

fourth pixel position	0	1	2	3	//4
resulting position	0	1	1	1	//2

Half sample values are found using bilinear interpolation as described in subclause 7.6.2. The prediction for luminance is obtained by overlapped motion compensation as described in subclause 7.6.6 if indicated by obmc_disable==0. The prediction for chrominance is obtained by applying the motion vector MVD_{CHR} to all pixels in the two chrominance blocks.

7.6.6 Overlapped motion compensation

This subclause specifies the overlapped motion compensation process. This process is performed when the flag obmc_disable=0. Note that overlapped motion compensaton is disabled over the boundary between macroblocks with different values for mcsel in an S (GMC)-VOP.

Each pixel in an 8*8 luminance prediction block is a weighted sum of three prediction values, divided by 8 (with rounding). In order to obtain the three prediction values, three motion vectors are used: the motion vector of the current luminance block, and two out of four "remote" vectors:

- the motion vector of the block at the left or right side of the current luminance block,
- the motion vector of the block above or below the current luminance block.

For each pixel, the remote motion vectors of the blocks at the two nearest block borders are used. This means that for the upper half of the block the motion vector corresponding to the block above the current block is used, while for the lower half of the block the motion vector corresponding to the block below the current block is used. Similarly, for the left half of the block the motion vector corresponding to the block at the left side of the current block is used, while for the right half of the block the motion vector corresponding to the block at the right side of the current block is used.

The creation of each pixel, $\tilde{p}(i,j)$, in an 8*8 luminance prediction block is governed by the following equation:

$$\bar{p}(i,j) = (q(i,j) \times H_0(i,j) + r(i,j) \times H_1(i,j) + s(i,j) \times H_2(i,j) + 4) // 8,$$

where $q(i,j)$, $r(i,j)$, and $s(i,j)$ are the pixels from the referenced picture as defined by

$$q(i,j) = p(i + MV_x^0, j + MV_y^0),$$
$$r(i,j) = p(i + MV_x^1, j + MV_y^1),$$
$$s(i,j) = p(i + MV_x^2, j + MV_y^2).$$

Here, (MV_x^0, MV_y^0) denotes the motion vector for the current block, (MV_x^1, MV_y^1) denotes the motion vector of the block either above or below, and (MV_x^2, MV_y^2) denotes the motion vector either to the left or right of the current block as defined above.

The matrices $H_0(i,j)$, $H_1(i,j)$ and $H_2(i,j)$ are defined in Figure 7-21, Figure 7-22, and Figure 7-23, where (i,j) denotes the column and row, respectively, of the matrix.

If one of the surrounding blocks was not coded, the corresponding remote motion vector is set to zero. If one of the surrounding blocks was coded in intra mode, the corresponding remote motion vector is replaced by the motion vector for the current block. If the current block is at the border of the VOP and therefore a surrounding block is not present, the corresponding remote motion vector is replaced by the current motion vector. In addition, if the current block is at the bottom of the macroblock, the remote motion vector corresponding with an 8*8 luminance block in the macroblock below the current macroblock is replaced by the motion vector for the current block.

4	5	5	5	5	5	5	4
5	5	5	5	5	5	5	5
5	5	6	6	6	6	5	5
5	5	6	6	6	6	5	5
5	5	6	6	6	6	5	5
5	5	6	6	6	6	5	5
5	5	5	5	5	5	5	5
4	5	5	5	5	5	5	4

Figure 7-21 -- Weighting values, H_0, for prediction with motion vector of current luminance block

2	2	2	2	2	2	2	2
1	1	2	2	2	2	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	2	2	2	2	1	1
2	2	2	2	2	2	2	2

Figure 7-22 -- Weighting values, H1, for prediction with motion vectors of the luminance blocks on top or bottom of current luminance block

2	1	1	1	1	1	1	2
2	2	1	1	1	1	2	2
2	2	1	1	1	1	2	2
2	2	1	1	1	1	2	2
2	2	1	1	1	1	2	2
2	2	1	1	1	1	2	2
2	2	1	1	1	1	2	2
2	1	1	1	1	1	1	2

Figure 7-23 -- Weighting values, H_2, for prediction with motion vectors of the luminance blocks to the left or right of current luminance block

7.6.7 Temporal prediction structure

A forward reference VOP is defined as a most recently decoded I- or P- or S(GMC)-VOP in the past for which "vop_coded==1". A backward reference VOP is defined as the most recently decoded I- or P- or S(GMC)-VOP in the future, regardless of its value for "vop_coded".

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA V. ADAMANE
Single user license only. Copying and networking prohibited.

A target P- or S(GMC)-VOP shall make reference to the forward reference VOP

A target B-VOP can make reference

- to the forward and/or the backward reference VOP, if for the backward reference VOP "vop_coded==1"
- only to the forward reference VOP, if for the backward reference VOP "vop_coded==0"

Note that for the reference VOP selection of binary shape coding the rules stated in subclause 7.5.2.4 shall be applied

The temporal prediction structure is depicted in Figure 7-24.

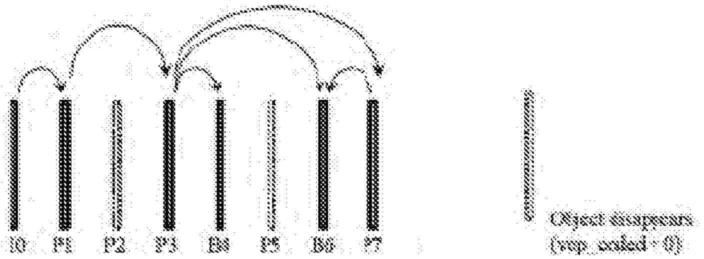


Figure 7-24 -- Temporal Prediction Structure

7.6.8 Vector decoding process of non-scalable progressive B-VOPs

In B-VOPs there are three kinds of vectors, namely, 16x16 forward vector, 16x16 backward vector and the delta vector for the direct mode. The vectors are decoded with respect to the corresponding vector predictors. The basic decoding process of a differential vector is the exactly same as defined in P-VOPs except that for the delta vector of the direct mode the f_code is always one. The vector is then reconstructed by adding the decoded differential vector to the corresponding vector predictor. The vector predictor for the delta vector is always set to zero, while the forward and backward vectors have their own vector predictors, which are reset to zero only at the beginning of each macroblock row. The vector predictors are updated in the following three cases:

- after decoding a macroblock of forward mode only the forward predictor is set to the decoded forward vector
- after decoding a macroblock of backward mode only the backward predictor is set to the decoded backward vector.
- after decoding a macroblock of bi-directional mode both the forward and backward predictors are updated separately with the decoded vectors of the same type (forward/backward).

7.6.9 Motion compensation in non-scalable progressive B-VOPs

In B-VOPs the overlapped motion compensation (OBMC) is not employed. The motion-compensated prediction of B-macroblock is generated by using the decoded vectors and taking reference to the padded forward/backward reference VOPs as defined below. Arbitrarily shaped reference VOPs shall be padded accordingly.

7.6.9.1 Basic motion compensation procedure

All of the ISO/IEC 14496-2 motion compensation techniques are based on the formation of a prediction block, pred[i][j] of dimension (width, height), from a reference image, ref[x][y]. The coordinates of the current block (or macroblock) in the reference VOP is (x,y), the motion half-pel resolution motion vecbr is (dx_halfpel, dy_halfpel). The pseudo-code for this procedure is given below. For quarter pel mode the formation of the prediction block is carried out as described in subclause 7.6.2.2.

The component_width() and component_height() function give the coded VOP dimensions for the current component. For luminance, component_width() is video_object_layer_width for a rectangular VOP or vop_width otherwise rounded up to the next multiple of 16. The luminance component_height() is defined similarly. The chrominance dimensions are one half of the corresponding luminance dimension.

```
clip_ref(ref, x, y)
{
    return(ref[MIN(MAX(x, 0), component_width(ref) - 1)]
              [MIN(MAX(y, 0), component_height(ref) - 1)]);
}
mc(pred,                          /* prediction block */
   ref,                           /* reference component */
   x, y,                          /* ref block coords for MV=(0, 0) */
   width, height,                 /* reference block dimensions */
   dx_halfpel, dy_halfpel,        /* half-pel resolution motion vector */
   rounding,                      /* rounding control (0 or 1 ) */
   pred_y0,                       /* field offset in pred blk (0 or 1) */
   ref_y0,                        /* field offset in ref blk (0 or 1) */
   y_incr)                        /* vertical increment (1 or 2) */
{
    dx = dx_halfpel >> 1;
    dy = y_incr * (dy_halfpel >> y_incr);
    if (dy_halfpel & y_incr) {
        if (dx_halfpel & 1) {
            for (iy = 0; iy < height; iy += y_incr) {
                for (ix = 0; ix < width; ix++) {
                    x_ref = x + dx + ix;
                    y_ref = y + dy + iy + ref_y0;
                    pred[ix][iy + pred_y0] =
                        (clip_ref(ref, x_ref + 0, y_ref + 0) +
                         clip_ref(ref, x_ref + 1, y_ref + 0) +
                         clip_ref(ref, x_ref + 0, y_ref + y_incr) +
                         clip_ref(ref, x_ref + 1, y_ref + y_incr) +
                         2 - rounding) >> 2;
                }
            }
        } else {
            for (iy = 0; iy < height; iy += y_incr) {
                for (ix = 0; ix < width; ix++) {
                    x_ref = x + dx + ix;
                    y_ref = y + dy + iy + ref_y0;
                    pred[ix][iy + pred_y0] =
                        (clip_ref(ref, x_ref, y_ref + 0) +
                         clip_ref(ref, x_ref, y_ref + y_incr) +
                         1 - rounding) >> 1;
                }
            }
        }
    } else {
        if (dx_halfpel & 1) {
            for (iy = 0; iy < height; iy += y_incr) {
                for (ix = 0; ix < width; ix++) {
                    x_ref = x + dx + ix;
                    y_ref = y + dy + iy + ref_y0;
                    pred[ix][iy + pred_y0] =
                        (clip_ref(ref, x_ref + 0, y_ref) +
                         clip_ref(ref, x_ref + 1, y_ref) +
                         1 - rounding) >> 1;
                }
            }
        } else {
            for (iy = 0; iy < height; iy += y_incr) {
                for (ix = 0; ix < width; ix++) {
                    x_ref = x + dx + ix;
                    y_ref = y + dy + iy + ref_y0;
                    pred[ix][iy + pred_y0] =
                       clip_ref(ref, x_ref, y_ref);
                }
            }
        }
    }
}
```

7.6.9.2 Forward mode

Only the forward vector (MVFx,MVFy) is applied in this mode. The prediction blocks Pf_Y, Pf_U, and Pf_V are generated fom the forward reference VOP, ref_Y_for for luminance component and ref_U_for and ref_V_for for chrominance components, as follows:

```
mc(Pf_Y, ref_Y_for, x, y, 16, 16, MVFx, MVFy, 0, 0, 0, 1);

mc(Pf_U, ref_U_for, x/2, y/2, 8, 8, MVFx_chro, MVFy_chro, 0, 0, 0,1);

mc(Pf_V, ref_V_for, x/2, y/2, 8, 8, MVFx_chro, MVFy_chro, 0, 0, 0,1);
```

where (MVFx_chro, MVFy_chro) is motion vector derived from the luminance motion vector by dividing each component by 2 then rounding on a basis of Table 7-9. Here (and hereafter) the function MC is defined in subclause 7.6.9.

7.6.9.3 Backward mode

Only the backward vector (MVBx,MVBy) is applied in this mode. The prediction blocks Pb_Y, Pb_U, and Pb_V are generated from the backward reference VOP, ref_Y_back for luminance component and ref_U_back and ref_V_back for chrominance components, as follows:

```
mc(Pb_Y, ref_Y_back, x, y, 16, 16, MVBx, MVBy, 0, 0, 0, 1);

mc(Pb_U, ref_U_back, x/2, y/2, 8, 8, MVBx_chro, MVBy_chro, 0, 0, 0,1);

mc(Pb_V, ref_V_back, x/2, y/2, 8, 8, MVBx_chro, MVBy_chro, 0, 0, 0,1);
```

where (MVBx_chro, MVBy_chro) is motion vector derived from the luminance motion vector by dividing each component by 2 then rounding on a basis of Table 7-9.

7.6.9.4 Bi-directional mode

Both the forward vector (MVFx,MVFy) and the backward vector (MVBx,MVBy) are applied in this mode. The prediction blocks Pi_Y, Pi_U, and Pi_V are generated from the forward and backward reference VOPs by doing the forward prediction, the backward prediction and then averaging both predictions pixel by pixel as follows.

```
mc(Pf_Y, ref_Y_for, x, y, 16, 16, MVFx, MVFy, 0, 0, 0, 1);

mc(Pf_U, ref_U_for, x/2, y/2, 8, 8, MVFx_chro, MVFy_chro, 0, 0, 0,1);

mc(Pf_V, ref_V_for, x/2, y/2, 8, 8, MVFx_chro, MVFy_chro, 0, 0, 0,1);

mc(Pb_Y, ref_Y_back, x, y, 16, 16, MVBx, MVBy, 0, 0, 0, 1);

mc(Pb_U, ref_U_back, x/2, y/2, 8, 8, MVBx_chro, MVBy_chro, 0, 0, 0,1);

mc(Pb_V, ref_V_back, x/2, y/2, 8, 8, MVBx_chro, MVBy_chro, 0, 0, 0,1);

Pi_Y[i][j] = (Pf_Y[i][j] + Pb_Y[i][j] + 1)>>1;      i,j=0,1,2…15;

Pi_U[i][j] = (Pf_U[i][j] + Pb_U[i][j] + 1)>>1;      i,j=0,1,2…8;

Pi_V[i][j] = (Pf_V[i][j] + Pb_V[i][j] + 1)>>1;      i,j=0,1,2…8;
```

where (MVFx_chro, MVFy_chro) and (MVBx_chro, MVBy_chro) are motion vectors derived from the forward and backward luminance motion vectors by dividing each component by 2 then rounding on a basis of Table 7-9, respectively

7.6.9.5 Direct mode

This mode uses direct bi-directional motion compensation derived by employing I-, P-, or S(GMC)-VOP macroblock motion vectors and scaling them to derive forward and backward motion vectors for macroblocks in B-VOP. This is the only mode which makes it possible to use motion vectors on 8x8 blocks. Only one delta motion vector is allowed per macroblock.

7.6.9.5.1 Formation of motion vectors for the direct mode

The direct mode utilises the motion vectors (MVs) of the co-located macroblock in the most recently I-, P-, or S(GMC)- VOP. The co-located macroblock is defined as the macroblock which has the same horizontal and vertical index with the current macroblock in the B-VOP. The averaged motion vector defined in subclause 7.8.7.3 is used as the MV of the co-located macroblock when this co-located macroblock is included in an S(GMC)-VOP and mcsel == '1' (i.e. when the co-located macroblock is a GMC macroblock). The MV vectors are the block vectors of the co-located macroblock after applying the vector padding defined in subclause 7.6.1.6. If the co-located macroblock is transparent and thus the MVs are not available, the direct mode is still enabled by setting MV vectors to zero vectors.

7.6.9.5.2 Calculation of vectors

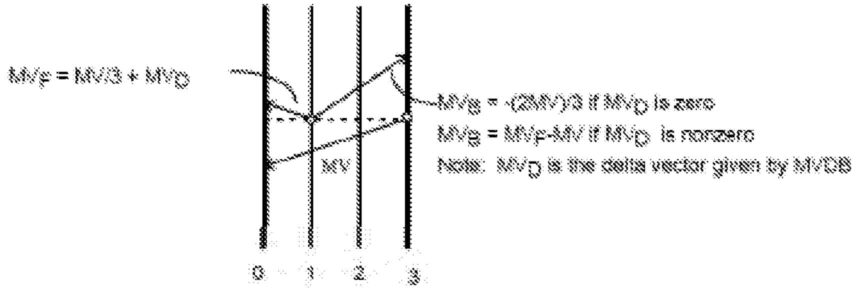

$MV_F = MV/3 + MV_D$
$MV_B = -(2MV)/3$ if MV_D is zero
$MV_B = MV_F - MV$ if MV_D is nonzero
Note: MV_D is the delta vector given by MVDB
MV
0
1
2
3

Figure 7-25 -- Direct Bi-directional Prediction

Figure 7-25 shows scaling of motion vectors. The calculation of forward and backward motion vectors involves linear scaling of the collocated block in temporally next I- P-, or S(GMC)-VOP, followed by correction by a delta vector (MVDx,MVDy). The forward and the backward motion vectors are {(MVFx[i],MVFy[i]), (MVBx[i],MVBy[i]), i = 0,1,2,3} and are given in half or quarter sample units as follows.

$$MVFx[i] = (TRB \times MVx[i]) / TRD + MVDx$$

$$MVBx[i] = (MVDx{=}{=}0)?\ ((TRB - TRD) \times MVx[i]) / TRD : MVFx[i] - MVx[i]$$

$$MvFy[i] = (TRB \times MVy[i]) / TRD + MVDy$$

$$MVBy[i] = (MVDy{=}{=}0)?\ ((TRB - TRD) \times MVy[i]) / TRD : MVFy[i] - MVy[i]$$

$$i = 0,1,2,3$$

where {(MVx[i],MVy[i]), i = 0,1,2,3} are the MV vectors of the co-located macroblock, TRB is the difference in temporal reference of the B-VOP and the previous reference VOP. TRD is the difference in temporal reference of the temporally next reference VOP with temporally previous reference VOP, assuming B-VOPs or skipped VOPs in between.

7.6.9.5.3 Generation of prediction blocks

Motion compensation for luminance is performed individually on 8x8 blocks to generate a macroblock. The process of generating a prediction block simply consists of using computed forward and backward motion vectors {(MVFx[i],MVFy[i]), (MVBx[i],MVBy[i]), i = 0,1,2,3} to obtain appropriate blocks from reference VOPs and averaging

© ISO/IEC 2001 – All rights reserved
[illegible]
[illegible]
Single user licence only. Copying and networking prohibited.

these blocks, same as the case of bi-directional mode except that motion compensaton is performed on 8x8 blocks. Note that in the case of quarter sample mode, the motion compensation of four 8x8 blocks, using the same motion vector for each block, does not lead to the sameresult as the motion compensation of a single 16x16 block using the same vector. This is due to the block boundary mirroring described in subclause 7.6.2.2.

For the motion compensation of both chrominance blocks, the forward motion vector (MVFx_chro, MVFy_chro) is calculated by the sum of K forward luminance motion vectors dividing by 2K and then rounding toward the nearest half sample position as defined in Table 7-6 to Table 7-9. In quarter sample mode the vectors are divided by 2 before summation. The backward motion vector (MVBx_chro, MVBy_chro) is derived in the same way. The rest process is he same as the chrominance motion compensation of the bi-directional mode described in subclause 7.6.9.4.

7.6.9.6 Motion compensation in skipped macroblocks

If the co-located macroblock in the most recently decoded I- or P-VOP is skipped, the current B-macroblock is treated as the forward mode with the zero motion vector (MVFx,MVFy). If the modb equals to '1' the current B-macroblock is reconstructed by using the direct mode with zero delta vector. If the co-located macroblock in the most recently decoded S(GMC)-VOP is skipped, this co-located macroblock is treated as a non-skipped macroblock with the averaged motion vector defined in subclause 7.8.7.3 for the current B-macroblock.

7.6.10 Motion Compensation Decoding of Reduced Resolution VOP

This subclause describes motion compensation decoding process for VOP with vop_reduced_resolution equal to "1". In this case, motion compensation is done with Macroblock size of 32 x 32 pixels or Block size of 16 x 16 pixels instead of Macroblock size of 16 x 16 pixels or Block size of 8 x 8 pixels for the case of normal resolution VOP(vop_reduced_resolution is set to "0" or vop_redcuced_resoluton flag is not present). Figure V2 -19 shows block diagram of motion compensation decoding process of Reduced Resolution VOP.

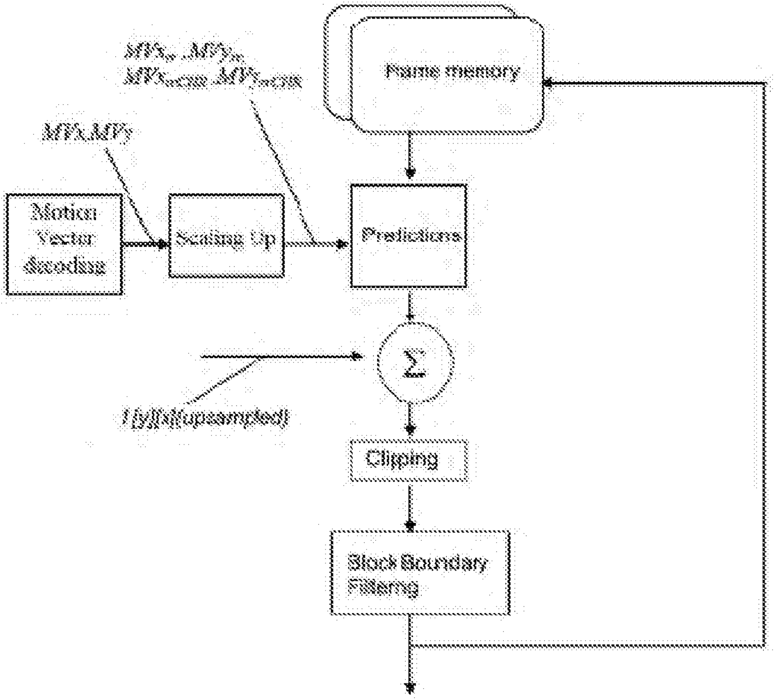


Figure V2 -19 -- Motion Compensation Decoding of Reduced Resolution VOP

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA V. ADAWANE
ANSI Store order X0274451 Downloaded: 7/18/2007 3:58:21 PM ET
Single user license only. Copying and networking prohibited.

7.6.10.1 Decoding Process of Reduced Resolution P-VOP

7.6.10.1.1 Motion Vector Decoding

Motion vector for luminance component *MVx*, *MVy* is decoded as in the same way as described in subclause 7.6.3. Four vector case, motion vector for each block is also decoded as described in 7.6.5. These decoced vectors for luminance are used for the candidate of the motion vector predictor in the decoding process of the following Macroblocks.

7.6.10.1.2 Scaling up of Motion Vectors

Decoded luminance motion vector *MVx*, *MVy* shall be scaled up to MVx_u, MVy_u by the following formula:

$$MVx_u = 0 \qquad \text{if } MVx = 0$$

$$MVx_u = \mathrm{Sign}(MVx) * (2.0 * \mathrm{Abs}(MVx) - 0.5) \qquad \text{if } MVx \neq 0$$

$$MVy_u = 0 \qquad \text{if } MVy = 0$$

$$MVy_u = \mathrm{Sign}(MVy) * (2.0 * \mathrm{Abs}(MVy) - 0.5) \qquad \text{if } MVy \neq 0$$

As a result, each vector component is restricted to have a half-integer or zero value, and the range of each motion vector component MVx_u, MVy_u is enlarged to approximately twice the range of decoded motion vector component *MVx*, *MVy*.

Motion vector for chrominace blocks MVx_{uCHR}, MVy_{uCHR} is derived by calculating the sum of the K luminance vectors MVx_u, MVy_u that corresponds to K 16x16 blocks that do not lie outside the VOP shape and dividing this sum by 2*K; the component values of the resulting sixteenth/twelfth/eighth/fourth sample resolution vectors are modified towards the nearest half sample position as indicated in Table 7-6, 7-7, 7-8 and 7-9.

7.6.10.1.3 Prediction

By using the scaled up luminance motion vector MVx_u, MVy_u and derived chrominance vector MVx_{uCHR}, MVy_{uCHR} prediction data for each 32x32 Macroblock or 16x16 block is generated from the refrence VOP data stored in frame memory. Half sample interpolation shall be done as described in subclause 7.6.2.

If Overlapped Motion Compensation is also used for the Reduced Resolution VOP, enlarged matrices of weighting values are used to perform the overlapped motion compensation. Except that the size of each block and weighting matrices is 16×16, the procedure of the creation of each prediction block is identical to the description of subclause 7.6.6.

The enlarged matrices of weighting values for the 16×16 uminance prediction are given in Figure V2 -20, Figure V2 -21, and Figure V2 -22.

4	4	5	5	5	5	5	5	5	5	5	5	5	5	4	4
4	4	5	5	5	5	5	5	5	5	5	5	5	5	4	4
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	6	6	6	6	6	6	6	6	5	5	5	5
5	5	5	5	6	6	6	6	6	6	6	6	5	5	5	5
5	5	5	5	6	6	6	6	6	6	6	6	5	5	5	5
5	5	5	5	6	6	6	6	6	6	6	6	5	5	5	5
5	5	5	5	6	6	6	6	6	6	6	6	5	5	5	5
5	5	5	5	6	6	6	6	6	6	6	6	5	5	5	5
5	5	5	5	6	6	6	6	6	6	6	6	5	5	5	5
5	5	5	5	6	6	6	6	6	6	6	6	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
4	4	5	5	5	5	5	5	5	5	5	5	5	5	4	4
4	4	5	5	5	5	5	5	5	5	5	5	5	5	4	4

Figure V2 -20 -- Weighting values, H_c, for prediction with motion vector of current 16×16 luminance block

2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
1	1	1	1	2	2	2	2	2	2	2	2	1	1	1	1
1	1	1	1	2	2	2	2	2	2	2	2	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	2	2	2	2	2	2	2	2	1	1	1	1
1	1	1	1	2	2	2	2	2	2	2	2	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

Figure V2 -21 -- Weighting values, H_1, for prediction with motion vector of 16×16 luminance blocks on top or bottom of current 16×16 luminance block

2	2	1	1	1	1	1	1	1	1	1	1	1	1	2	2
2	2	1	1	1	1	1	1	1	1	1	1	1	1	2	2
2	2	2	2	1	1	1	1	1	1	1	1	2	2	2	2
2	2	2	2	1	1	1	1	1	1	1	1	2	2	2	2
2	2	2	2	1	1	1	1	1	1	1	1	2	2	2	2
2	2	2	2	1	1	1	1	1	1	1	1	2	2	2	2
2	2	2	2	1	1	1	1	1	1	1	1	2	2	2	2
2	2	2	2	1	1	1	1	1	1	1	1	2	2	2	2
2	2	2	2	1	1	1	1	1	1	1	1	2	2	2	2
2	2	2	2	1	1	1	1	1	1	1	1	2	2	2	2
2	2	2	2	1	1	1	1	1	1	1	1	2	2	2	2
2	2	2	2	1	1	1	1	1	1	1	1	2	2	2	2
2	2	2	2	1	1	1	1	1	1	1	1	2	2	2	2
2	2	2	2	1	1	1	1	1	1	1	1	2	2	2	2
2	2	1	1	1	1	1	1	1	1	1	1	1	1	2	2
2	2	1	1	1	1	1	1	1	1	1	1	1	1	2	2

Figure V2 -22 – Weighting values, H_b, for prediction with motion vector of 16×16 luminance blocks to the left or right of current 16×16 luminance block

7.6.10.1.4 Addition and Clipping

The 16×16 reconstructed prediction error block data derived by upsampling process described in subclause 7.4.6 is added to the 16×16 prediction block data generated by prediction process defined in subclause 7.6.10.1.3. All the resulted 16×16 reconstructed block data shall be clipped to the range of $[0, 2^{\mathrm{bits\_per\_pixel}} - 1]$.

7.6.10.1.5 Block Boundary Filtering

The filter operation described in this clause shall be performed along the edges of the 16×16 reconstructed blocks data.

The filtering is performed on the complete reconstructed VOP data before storing the data in the frame memory for future prediction. No filtering is performed across VOP edges and NEWPRED segment (defined in subclause 7.14.1) edges. Chrominance as well as luminance data is filtered.

If A and B are two pixel values on a line — horizontal or vertical — of the reconstructed VOP, and A belongs to one 16×16 block called block1 whereas B belongs to a neighboring 16×16 block called block2 which is to the right or below of block1. Figure V2 -23 shows examples for the position of these pixels.

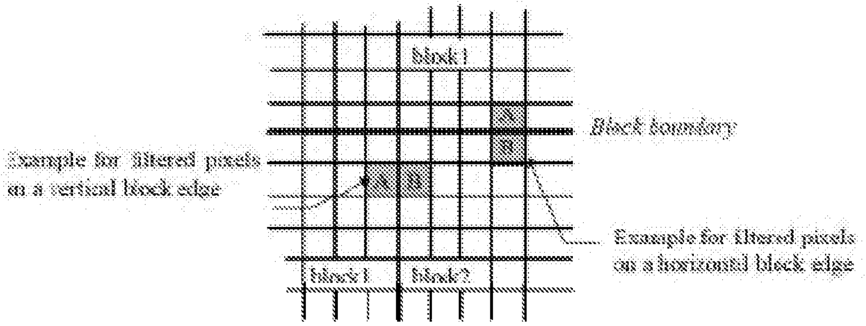

block1
Block boundary
Example for filtered pixels
on a vertical block edge
Example for filtered pixels
on a horizontal block edge
block1
block2

Figure V2 -23 – Block boundary filter

One of the following conditions must be fulfilled in order to turn the filter on for a particular edge:

block1 belongs to a coded macroblock (not_coded == 0 || Macroblock type == INTRA) or

block2 belongs to a coded macroblock (not_coded == 0 || Macroblock type == INTRA)

A shall be replaced by A1 and B shall be replaced by B1.

A1 = (3*A + B + 2) / 4

B1 = (A + 3*B + 2) / 4

Filtering across horizontal edges: Basically this process is assumed to take place first. More precisely, the pixels $\begin{pmatrix} A \\ B \end{pmatrix}$ that are used in filtering across a horizontal edge shall not have been influenced by previous filtering across a vertical edge.

Filtering across vertical edges:

Before filtering across a vertical edge using pixels (A,B), all modifications of pixels (A,B) resulting from filtering across a horizontal edge shall have taken place.

7.6.10.2 Decoding Process of Reduced Resolution I-VOP

For I-VOP with the vop_reduced_resolution equal to "1", Addition and Clipping process described in subclause 7.6.10.1.4 shall be performed with the fixed prediction data value of "0". Block Boundary Filtering operation described in subclause 7.6.10.1.5 also shall be applied. The reconstructed I-VOP data is stored in the frame memory for future prediction.

7.7 Interlaced video decoding

This subclause specifies the additional decoding process that a decoder shall perform to recover VOP data from the coded bitstream when the interlaced flag in the VOP header is set to "1". Interlaced information (subclause 6.3.6.3) specifies the method to decode bitstream of interlaced VOP.

7.7.1 Field DCT and DC and AC Prediction

When dct_type flag is set to '1' (field DCT coding), DCT coefficients of luminance data are formed such that each 8x8 block consists of data from one field as being shown in Figure 6-7. DC and optional AC (see "ac_pred_flag") prediction will be performed for a intra-coded macroblock. For the intra macroblocks which have dct_type flag being set to "1", DC/AC prediction are performed to field blocks shown in Figure 7-20. After taking inverse DCT, all luminance blocks will be inverse permuted back to (frame) macroblock. Chrominance (block) data are not effected by dct_type flag.

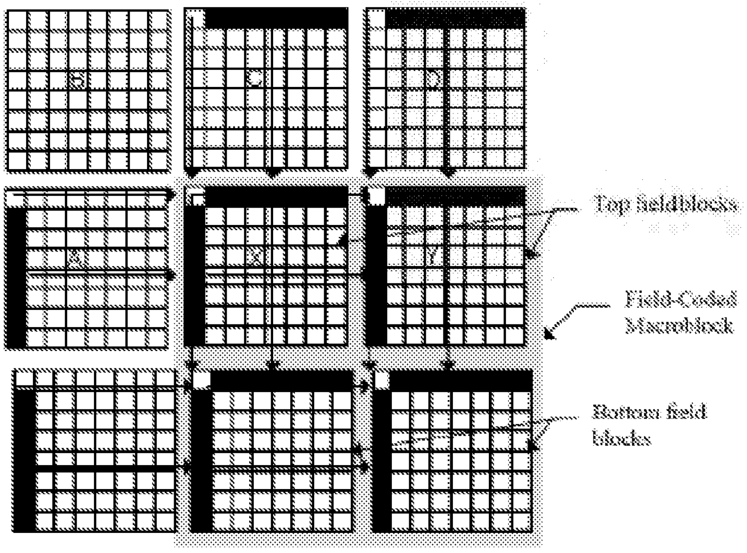


Figure 7-26 – Previous neighboring blocks used in DC/AC prediction for interlaced intra blocks.

7.7.2 Motion compensation

For non-intra macroblocks in P- and B-, and S(GMC)-VOPs, motion vectors are extracted syntactically following subclause 6.2.6 "Macroblock". The motion vector decoding is performed separately on the horizontal and vertical components.

7.7.2.1 Motion vector decoding in P- and S(GMC)-VOP

For each component of motion vector in P- and S(GMC)-VOPs, the median value of the candidate predictor vectors for the same component is computed and add to corresponding component of the motion vector difference obtained from the bitstream. To decode the motion vectors in a P- and S(GMC)-VOP, the decoder shall first extract the differential motion vectors ($(MVDx_{f1}, MVDy_{f1})$ and $(MVDx_{f2}, MVDy_{f2})$ for top and bottom fields of a field predicted macroblock, respectively) by a use of variable length decoding and then determine the predictor vector from three candidate vectors. These candidate predictor vectors are generated from the three motion vectors of three spatial neighborhood decoded macroblocks or blocks as follows. Note that the averaged motion vector as defined in subclause 7.8.7.3 is used as the candidate predictor vector for macroblocks with mcsel == '1' when the current VOP is an S(GMC)-VOP. Such macroblock is regarded as a frame predicted macroblock in the following discussion.

CASE 1 :

If the current macroblock is a field predicted macroblock and none of the coded spatial neighborhood macroblocks is a field predicted macroblock, then candidate predictor vectors MV1, MV2, and MV3 are defined by Figure 7-27. If the candidate block *i* is not in four MV motion (8x8) mode, MV*i* represents the motion vector for the macroblock. If the candidate block *i* is in four MV motion (8x8) mode, the 8x8 block motion vector closest to the upper left block of the current MB is used. The predictors for the horizontal and vertical components are then computed by

$$P_x = Median(MV1x, MV2x, MV3x)$$

$$P_y = Median(MV1y, MV2y, MV3y).$$

For differential motion vectors both fields use the same predictor and motion vectors are recovered by

$$MVx_{f1} = MVDx_{f1} + P_x$$
$$MVy_{f1} = 2 * (MVDy_{f1} + (P_y / 2))$$
$$MVx_{f2} = MVDx_{f2} + P_x$$
$$MVy_{f2} = 2 * (MVDy_{f2} + (P_y / 2))$$

where "/" is integer division with truncation toward 0. Note hat all motion vectors described above are specified as integers with one LSB representing a half- or quarter pel displacement. The vertical component of field motion vectors always even (in half- or quarter pel frame coordinates). Vertical half- or quarte pel interpolation between adjacent lines of the same field is denoted by MVy_{fi} be an odd multiple of 2 (e.g. -2,2,6,..). No vertical interpolation is needed when MVy_{fi} is an multiple of 4 for half or 8 for quarter pel (it is a full pel value).

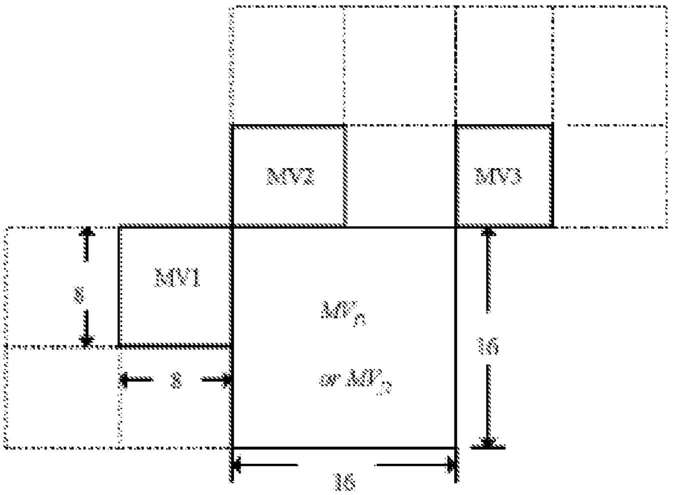


Figure 7-27 -- Example of motion vector prediction for field predicted macroblocks (Case1)

CASE 2 :

If the current macroblock or block is frame predicted macroblock or block and if at least one of the coded spatial neighborhood macroblocks is a field predicted macroblock, then the candidate predictor vector for each field predicted macroblock will be generated by averaging two field motion vectors such that all fractional pel offsets are mapped into the half or quarter -pel displacement. Each component (P_x or P_y) of the final predictor vector is the median value of the candidate predictor vectors for the same component. The motion vector is recovered by

$$MVx = MVDx + P_x$$
$$MVy = MVDy + P_y$$

where

$$P_x = Median\{MV1x, Div2Round(MVx_{f1} + MVx_{f2}), MV3x\},$$
$$P_y = Median\{MV1y, Div2Round(MVy_{f1} + MVy_{f2}), MV3y\},$$

Div2Round(x) is defined as follows: *Div2Round(x)* = (*x* >> 1) | (*x* & 1).

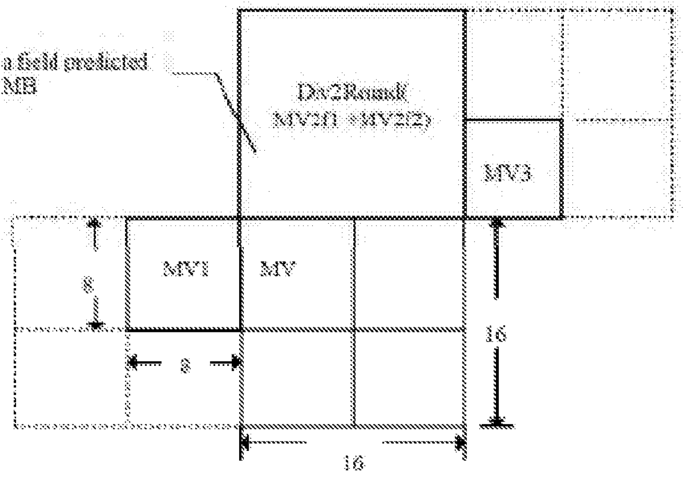


Figure 7-28 – Example of motion vector prediction for field predicted macroblocks (Case 2)

CASE 3 :

Assume that the current macroblock is a field predicted macroblock and at least one of the coded spatial neighborhood macroblocks is a field predicted macroblock. If the candidate block *i* is field predicted, the candidate predictor vector MV*i* will be generated by averaging two field motion vectors such that all fractional pel offsets are mapped into the half- or quarter pel displacement as discribed in CASE 2. If the candidate block *i* is neither in four MV motion (8x8) mode nor in field prediction mode, MV*i* represents the frame motion vector for the macroblock. If the candidate block *i* is in four MV motion (8x8) mode, the 8x8 block motion vector closest to the upper left block of the current MB is used. The predictors for the horizontal and vertical components are then computed by

$$P_x = Median(MV1x, MV2x, MV3x)$$
$$P_y = Median(MV1y, MV2y, MV3y)$$

where

$$MVi.x = Div2Round(MVx_{f1} + MVx_{f2}),$$
$$MVi.y = Div2Round(MVy_{f1} + MVy_{f2}),$$

for some *i* in {1,2,3}.

For differential motion vectors both fields use the same predictor and motion vectors are recovered by (see both Figure 7-27 and Figure 7-28)

$$MVx_{f1} = MVDx_{f1} + P_x$$
$$MVy_{f1} = 2 * (MVDy_{f1} + (P_y / 2))$$
$$MVx_{f2} = MVDx_{f2} + P_x$$
$$MVy_{f2} = 2 * (MVDy_{f2} + (P_y / 2))$$

The motion compensated prediction macroblock is calculated calling the "field_compensate_one_reference" using the motion vectors calculated above. In quarter_sample mode the macroblock is calculated as described in subclause 7.6.2.2, accordingly. The top_field_ref, botom_field_ref, and rounding type come directly from the syntax as forward_top_field_reference, forward_bottom_field_reference and vop_rounding_type respectively. The reference VOP is defined such the the even lines (0, 2, 4, ...) are the top field and the odd lines (1, 3, 5, ...) are the bottom field.

```
field_motion_compensate_one_reference(
    luma_pred, cb_pred, cr_pred, /* Prediction component pel array */
    luma_ref, cb_ref, cr_ref,    /* Reference VOP pel arrays */
    mv_top_x, mv_top_y,           /* top field motion vector */
    mv_bot_x, mv_bot_y,          /* bottom field motion vector */
    top_field_ref,               /* top field reference */
    bottom_field_ref,            /* bottom field reference */
    x, y,                        /* current luma macroblock coords */
    rounding_type)               /* rounding type */
{
    mc(luma_pred, luma_ref, x, y, 16, 16, mv_top_x, mv_top_y,
        rounding_type, 0, top_field_ref, 2);
    mc(luma_pred, luma_ref, x, y, 16, 16, mv_bot_x, mv_bot_y,
        rounding_type, 1, bottom_field_ref, 2);
    mc(cb_pred, cb_ref, x/2, y/2, 8, 8,
        Div2Round(mv_top_x), Div2Round(mv_top_y),
        rounding_type, 0, top_field_ref, 2);
    mc(cr_pred, cr_ref, x/2, y/2, 8, 8,
        Div2Round(mv_top_x), Div2Round(mv_top_y),
        rounding_type, 0, top_field_ref, 2);
    mc(cb_pred, cb_ref, x/2, y/2, 8, 8,
        Div2Round(mv_bot_x), Div2Round(mv_bot_y),
        rounding_type, 1, bottom_field_ref, 2);
    mc(cr_pred, cr_ref, x/2, y/2, 8, 8,
        Div2Round(mv_bot_x), Div2Round(mv_bot_y),
        rounding_type, 1, bottom_field_ref, 2);
}
```

In the case that obmc_disable is "0", the OBMC is not applied if the current MB is field-predicted. If the current MB is frame-predicted (including 8x8 mode) and some adjacent MBs are field-predicted, the motion vectors of those field-predicted MBs for OBMC are computed in the same manner as the candidate predictor vectors for field-predicted MBs are.

7.7.2.2 Motion vector decoding in B-VOP

For interlaced B-VOPs, a macroblock can be coded using (1) direct coding, (2) 16x16 motion compensation (includes forward, backward & bidirectional modes), or (3) field motion compensation (includes forward, backward & bidirectional modes). Motion vector in half or quarter sample accuracy will be employed for a 16x16 macroblock being coded. Chrominance vectors are derived by scaling of luminance vectors using the rounding tables described in Table 7-9 (i.e. by applying *Div2Round* to the luminance motion vectors, divided by 2 in case of quarter_sample mode). These coding modes except direct coding mode allow switching of quantizer from the one previously in use. Specification of dquant, a differential quantizer involves a 2-bit overhead as discussed earlier. In direct coding mode, the quantizer value for previous coded macroblock is used.

For interlaced B-VOP motion vector predictors, four prediction motion vectors (PMVs) are used:

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/KOPOLN ADAMANE
ANSI Store order #X07445 Downloaded 7/18/2007 2:56:21 PM ET
Single user license only. Copying and networking prohibited.

Table 7-10 – Prediction motion vector allocation for interlaced P- and S(GMC)-VOPs

Function	PMV
Top field forward	0
Bottom field forward	1
Top field backward	2
Bottom field backward	3

These PMVs are used as follows for the different macroblock prediction modes:

Table 7-11 – Prediction motion vectors for interlaced B-VOP decoding

Macroblock mode	PMVs used	PMVs updated
Direct	none	none
Frame forward	0	0,1
Frame backward	2	2,3
Frame bidirectional	0,2	0,1,2,3
Field forward	0,1	0,1
Field backward	2,3	2,3
Field bidirectional	0,1,2,3	0,1,2,3

The PMVs used by a macroblock are set to the value of current macroblock motion vectors after being used.

When a frame macroblock is decoded, the two field PMVs (top and bottom field) for each prediction direction are set to the same frame value. The PMVs are reset to zero at the beginning of each row of macroblocks. The predictors are not zeroed by skipped macroblocks or direct mode macroblocks.

The frame based motion compensation modes are described in subclause 7.6. The field motion compensation modes are calculated using the "field_motion_compensate_one_reference()" pseudo code function described above. The field forward mode is denoted by mb_type == "0001" and field_prediction == "1". The PMV update and calculation of the motion compensated prediction is shown below. The luma_fwd_ref_VOP[][], cb_fwd_ref_VOP[][], cr_fwd_ref_VOP[][] denote the entire forward (past) anchor VOP pixel arrays. The coordinates of the upper left corner of the luminance macroblock is given by (x, y) and MVD[].x and MVD[].y denote an array of the motion vector differences in the order they occur in the bitstream for the current macroblock.

```
PMV[0].x = PMV[0].x + MVD[0].x;

PMV[0].y = 2 * (PMV[0].y / 2 + MVD[0].y);

PMV[1].x = PMV[1].x + MVD[1].x;

PMV[1].y = 2 * (PMV[1].y / 2 + MVD[1].y);

field_motion_compensate_one_reference(

    luma_pred, cb_pred, cr_pred,

    luma_fwd_ref_vop, cb_fwd_ref_vop, cr_fwd_ref_vop,

    PMV[0].x, PMV[0].y, PMV[1].x, PMV[1].y,

    forward_top_field_reference,

    forward_bottom_field_reference,

    x, y, 0);
```

The field backward mode is denoted by mb_type == "001" and field_prediction == "1". The PMV update and prediction calculation is outlined the following pseudo code. The luma_bak_ref_vop[][], cb_bak_ref_vop[][], cr_bak_ref_vop[][] denote the entire backward (future) anchor VOP pixel arrays.

```
    PMV[2].x = PMV[2].x + MVD[0].x;

    PMV[2].y = 2 * (PMV[2].y / 2 + MVD[0].y);

    PMV[3].x = PMV[3].x + MVD[1].x;

    PMV[3].y = 2 * (PMV[3].y / 2 + MVD[1].y);

    field_motion_compensate_one_reference(

        luma_pred, cb_pred, cr_pred,

        luma_bak_ref_vop, cb_bak_ref_vop, cr_bak_ref_vop,

        PMV[2].x, PMV[2].y, PMV[3].x, PMV[3].y,

        backward_top_field_reference,

        backward_bottom_field_reference,

        x, y, 0);
```

The bidirectional field prediction is used when mb_type == "01" and field_prediction == "1". The prediction macroblock (in luma_pred[][], cb_pred[][], and cr_pred[][]) is calculated by:

```
for (mv = 0; mv < 4; mv++) {

        PMV[mv].x = PMV[mv].x + MVD[mv].x;

        PMV[mv].y = 2 * (PMV[mv].y / 2 + MVD[mv].y);

    }

    field_motion_compensate_one_reference(

        luma_pred_fwd, cb_pred_fwd, cr_pred_fwd,

        luma_fwd_ref_vop, cb_fwd_ref_vop, cr_fwd_ref_vop,

        PMV[0].x, PMV[0].y, PMV[1].x, PMV[1].y,

        forward_top_field_reference,

        forward_bottom_field_reference,

        x, y, 0);

    field_motion_compensate_one_reference(

        luma_pred_bak, cb_pred_bak, cr_pred_bak,
```

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/OSPOO/J.A.Y. ADAMANE
ANSI Store order #X27445' Downloaded 7/18/2007 2:58:21 PM ET
Single user license only. Copying and networking prohibited.

```
        luma_bak_ref_vop, cb_bak_ref_vop, cr_bak_ref_vop,
        PMV[2].x, PMV[2].y, PMV[3].x, PMV[3].y,
        backward_top_field_reference,
        backward_bottom_field_reference,
        x, y, 0);
    for (iy = 0; iy < 16; iy++) {
        for (ix = 0; ix < 16; ix++) {
            luma_pred[ix][iy] = (luma_pred_fwd[ix][iy] +
                                 luma_pred_bak[ix][iy] + 1) >> 1;
        }
    }
    for (iy = 0; iy < 8; iy++) {
        for (ix = 0; ix < 8; ix++) {
            cb_pred[ix][iy] = (cb_pred_fwd[ix][iy] +
                               cb_pred_bak[ix][iy] + 1) >> 1;
            cr_pred[ix][iy] = (cr_pred_fwd[ix][iy] +
                               cr_pred_bak[ix][iy] + 1) >> 1;
        }
    }
```

The direct mode prediction can be either progressive (see subclause 7.6.9.5) or interlaced as described below. Interlaced direct mode is used when ever the co-located macroblock (macroblock with the same coordinates) of the future anchor VOP has field_predition flag is "1 Note that if the future macroblock is a skipped macroblock in a P-VOP, a GMC macroblock (i.e. mcsel == '1') in an S(GMC)-VOP, or an intra macroblock, the direct mode prediction is progressive. Otherwise, interlaced direct mode prediction is used.

Interlaced direct coding mode is an extension of progressive direct coding mode. Four derived field motion vectors are calculated from the forward field motion vectors of the co-located future anchor VOP, a single differential motion vector and the temporal position of the B-VOP fields with respect to the fields of the past and future anchor VOPs. The four derived field motion vectors are denoted mvf[0] (top field forward) mvf[1], (bottom field forward), mvb[0] (top field backward), and mvb[1] (bottom field backward). MV[i] is the future anchor picture motion vector for the top (i == 0) and bottom (i == 1) fields. Only one delta motion vector (used for both field), MVD[0], occurs in the bitstream for the field direct mode predicted macroblock. MVD[0] is decoded assuming f_code == 1 regardless of the number in VOP header. The interlaced direct mode prediction (in luma_pred[][], cb_pred[][] and cr_pred[][]) is calculated as shown below.

```
for (i = 0; i < 2; i++) {
    mvf[i].x = (TRB[i] * MV[i].x) / TRD[i] + MVD[0].x;
    mvf[i].y = (TRB[i] * MV[i].y) / TRD[i] + MVD[0].y;
    mvb[i].x = (MVD[i].x == 0) ?
        (((TRB[i] - TRD[i]) * MV[i].x) / TRD[i]) :
        mvf[i].x - MV[i].x);
    mvb[i].y = (MVD[i].y == 0) ?
        (((TRB[i] - TRD[i]) * MV[i].y) / TRD[i]) :
        mvf[i].y - MV[i].y);
field_motion_compensate_one_reference(
    luma_pred_fwd, cb_pred_fwd, cr_pred_fwd,
    luma_fwd_ref_vop, cb_fwd_ref_vop, cr_fwd_ref_vop,
    mvf[0].x, mvf[0].y, mvf[1].x, mvf[1].y,
    colocated_future_mb_top_field_reference,
    colocated_future_mb_bottom_field_reference,
    x, y, 0);
field_motion_compensate_one_reference(
    luma_pred_bak, cb_pred_bak, cr_pred_bak,
    luma_bak_ref_vop, cb_bak_ref_vop, cr_bak_ref_vop,
    mvb[1].x, mvb[1].y, mvb[1].x, mvb[1].y,
    0, 1, x, y, 0);
for (iy = 0; iy < 16; iy++) {
    for (ix = 0; ix < 16; ix++) {
        luma_pred[ix][iy] = (luma_pred_fwd[ix][iy] +
                              luma_pred_bak[ix][iy] + 1) >> 1;
    }
}
for (iy = 0; iy < 8; iy++) {
    for (ix = 0; ix < 8; ix++) {
```

© ISO/IEC 2001 – All rights reserved
Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA Y. ADHAVANE
ANSI Store order X274451 Downloaded 7/18/2007 3:58:21 PM ET
Single user license only. Copying and networking prohibited.

```
            cb_pred[ix][iy] = (cb_pred_fwd[ix][iy] +
                               cb_pred_bak[ix][iy] + 1) >> 1;
            cr_pred[ix][iy] = (cr_pred_fwd[ix][iy] +
                               cr_pred_bak[ix][iy] + 1) >> 1;
        }
    }
```

The temporal references (TRB[i] and TRD[i]) are distances in time expressed in field periods. Figure 7-29 shows how they are defined for the case where i is 0 (top field of the B-VOP). The bottom field is analogously.

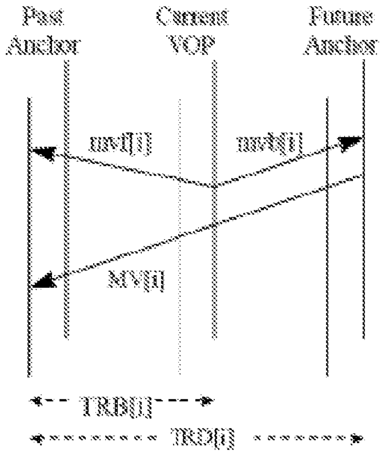


Figure 7-29 -- Interlaced direct mode

The calculation of TRD[i] and TRB[i] depends not only on the current field, reference field, and frame temporal references, but also on whether the current video is top feld first or bottom field first.

$$TRD[i] = 2*(T(future)//Tframe - T(past)//Tframe) + \delta[i]$$

$$TRB[i] = 2*(T(current)//Tframe - T(past)//Tframe) + \delta[i]$$

where T(future), T(current) and T(past) are the cumulative VOP times calculated from modulo_time_base and vop_time_increment of the future, current and past VOPs in display order. Tframe is the frame period determined by

$$Tframe = T(first\_B\_VOP) - T(past\_anchor\_of\_first\ B\_VOP)$$

where first_B_VOP denotes the first B-VOP following the Video Object Layer syntax. The important thing about Tframe is that the period of time between consecutive fields which constitute an interlaced frame is assumed to be 0.5 * Tframe for purposes of scaling the motion vectors.

The value of δ is determined from Table 7-12; it is a function of the current field parity (top or bottom), the reference field of the co-located macroblock (macroblock at the same coordinates in the furture anchor VOP), and the value of top_field_first in the B-VOP's video object plane syntax.

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA Y. ADAVANE
Single user license only. Copying and networking prohibited.

Table 7-12 – Selection of the parameter δ

future anchor VOP reference fields of the co-located macroblock		top_fied_first == 0		top_field_first == 1	
Top field reference	**Bottom field reference**	**Top field, δ[0]**	**Bottom field, δ[1]**	**Top field, δ[0]**	**Bottom field, δ[1]**
0	0	0	-1	0	1
0	1	0	0	0	0
1	0	1	-1	-1	1
1	1	1	0	-1	0

The top field prediction is based on the top field motion vector of the P-VOP macroblock of the future anchor picture. The past reference field is the reference field selected by the co-located macroblock of the future anchor picture for the top field. Analogously, the bottom field predictor is the average of pixels obtained from the future anchor's bottom field and the past anchor field referenced by the bottom field motion vector of the corresponding macroblock of the future anchor picture. When interlaced direct mode is used, vop_time_increment_resolution must be the smallest integer greater than or equal to the number of frames per second. In each VOP, vop_time_increment counts individual frames within a second.

7.8 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 7-30 is a diagram of the sprite decoding process. It is simplified for clarity.

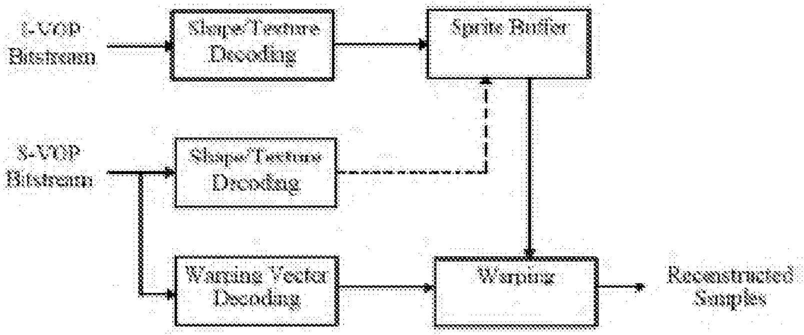


Figure 7-30 – The sprite decoding process

7.8.1 Highersyntactic structures

The various parameters in the VOL and VOP bitstreams shall be interpreted as described in clause 6. When sprite_enable == 'static', 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 with sprite_enable == 'static' in the same VOL. An S-VOP with sprite_enable == 'static' 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 with sprite_enable == 'static' can update the information stored in the sprite memory before applying warping.

Licensed to UNIVERSITY OF TEXAS AT [illegible]
Single user license only. Copying and networking prohibited.

7.8.2 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
  (i_c', j_c') = (sprite_left_coordinate / 2, sprite_top_coortinate / 2)

- Bottom right chrominance sample
  (i_c', j_c') = (sprite_left_coordinate / 2 + sprite_width// 2 – 1,
  sprite_top_coordinate / 2 + sprite_height// 2 – 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 VOPs

- Bottom right sample of chrominance
  (i_c, j_c) = (video_object_layer_width / 2 - 1, video_object_layer_height / 2 - 1) for rectangular 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 VOPs

7.8.3 Low-latency sprite reconstruction

This subclause allows a large static sprite to be reconstructed at the decoder by properly incorporating its corresponding pieces. There are two types of pieces recognized by the decoder—object and update. The decoded sprite object-piece (i.e., embedded in a S-VOP with low_latence_sprite_enable==1 and sprite_transmit_mode=="piece") is a highly quantized version of the original sprite piece while the sprite update-piece (i.e., sprite_transmit_mode=="update") is a residual designed to improve upon the quality of decoded object-piece. Sprite pieces are rectangular pieces of texture (and shape for the object-piece) and can contain "holes," corresponding to macroblocks, that do not need to be decoded. Five parameters are required by the decoder to properly incorporate the pieces: piece_quant, piece_width, piece_height, piece_xoffset, and piece_yoffset.

Macroblocks raster scanning is employed to decode each piece. However, whenever the scan encounters a macroblock which has been part of some previously sent sprite piece, then the macroblock is not decoded and its corresponding macroblock layer is empty. In that case, the decoder treats the macroblock as a hole in the current sprite piece. Since a macroblock can be refined as long as there is some available bandwidth, more than one update may be decoded per macroblock and the holes for a given refinement step have no relationship to the holes of later refinement steps. Therefore, the decoding process of a hole for an update piece is different than that for the object-piece. For the object-piece, no information is decoded at all and the decoder must "manage" where "holes"

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA V. AGAWANE
Single user license only. Copying and networking prohibited.

lie. (see subclause 7.8.3.1). For the update-piece, the not_coded bit is decoded to indicate whether or not one more refinement should be decoded for this given macroblock. (see subclause 7.8.3.2) Note that a hole could be non-transparent and have had shape information decoded previously. Multiple intermingled object-pieces and update-pieces may be decoded at the same current VOP. Part of a sequence could consist for example of rapidy showing a zooning out effect, a panning to the right, a zooming in, and finaly a panningto the left. In this case, the first decoded object-piece covers regions on all four sides of the previous VOP transmitted piece, which is now treated as a hole and not decoded again. The second decoded object-piece relates to the right panning, and the third object-piece is a smaller left-panning piece due to the zooming-in effect. Finally, the last piece is different; instead of an object, it contains the update for some previous object-piece of zooming-in (thus, the need to update to refine for higher quality). All four pieces will be decoded within the same VOP. When sprite_transmit_mode = ="pause," the decoder recognizes that all sprite object-pieces and update-pieces for the current VOP session have been sent. However, when sprite_transmit_mode = "stop," the decoder understands that all object and update-pieces have been sent for the entire video object layer, not just for the current VOP. session. In addition, once all object-pieces or update-pieces have been decoded during a VOP session (i.e., signaled by sprite_transmit_mode == "pause" or sprite_transmit_mode == "stop"), the static sprite is padded (as defined in subclause 7.6.1), then the portion to be displayed is warped, to complete the current VOP session.

For the S-VOPs (i.e., vop_coding_type == "S") with sprite_enable == 'static', the macroblock layer syntax of object-pieces is the same as those of I-VOP. Therefore, shape and texture are decoded using the macroblock layer structure in I-VOPs with the quantisation of intra macroblocks. The syntax of the update-pieces is similar to the P-VOP inter-macroblock syntax with the quantisation of non-intra macroblocks); however, the differences are indicated in Table B-1, specifically that there are no motion vectors and shape information included in this decoder syntax structure. In summary, this decoding process supports the construction of any large sprite image progressively, both spatially and in terms of quality.

7.8.3.1 Decoding of holes in sprite object-piece

Implementation of macroblock scanning must account for the possibility that a macroblock uses prediction based on some macroblock sent in a previous piece. When an object-piece with holes is decoded, the decoder in the process of reconstruction acts as if the whole original piece were decoded, but actually only the bitstream corresponding to the "new macroblock" is received. Whenever macroblocks raster scanning encounters a hole, the decoder needs to manage the retrieval of relevant information (e.g. DCT quantisation parameters, AC and DC prediction parameters, and BAB bordering values) from the corresponding macroblock decoded earlier.

7.8.3.2 Decoding of holes in sprite update-pieces

In contrast to the send_mb() used by the object-pieces, the update-pieces use the not_coded bit. When not_coded = 1 in the P-VOP syntax, the decoder recognizes that the corresponding macroblock is not refined by the current sprite update-piece. When not_coded = 0 in the P-VOP syntax, the decoder recognizes that this macroblock is refined. The prediction for the update piece is obtained by extracting the "area" of the static sprite defined by (piece_width, piece_height, piece_xoffset, piece_yoffset). This area is then padded and serves as prediction for the update pieces. Since there is no shape information included in an update-piece, the result of its transparent_mb() is retrieved from the corresponding macroblock in the object-piece decoded earlier. In addition, an update macroblock cannot be transmitted before its corresponding object macroblock. As a result, the very first sprite piece transmitted in the low-latency mode shall be an object-piece.

7.8.4 Sprite reference point decoding

The syntatic elements in sprite_trajectory () and below shal be interpreted as specified in clause 6. du[i] and dv[i] (0 =< i < no_of_sprite_warping_points) specifies the mapping between indexes of some reference points in the VOP and the corresponding reference points in the sprite or reference VOP. 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,
$(i_1, j_1) = (i_0+W,\ j_0)$,
$(i_2, j_2) = (i_0, j_0 + H)$,
$(i_3, j_3) = (i_0+W,\ j_0+H)$

Licensed to UNIVERSITY OF TEXAS AT [illegible]
Single user license only. Copying and networking prohibited.

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_of_sprite_warping_points shall be used for the rest of the decoding process.

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

$$\begin{aligned}
(i_0', j_0') &= (s / 2)\,(2\,i_0 + du[0],\ 2\,j_0 + dv[0])\\
(i_1', j_1') &= (s / 2)\,(2\,i_1 + du[1] + du[0],\ 2\,j_1 + dv[1] + dv[0])\\
(i_2', j_2') &= (s / 2)\,(2\,i_2 + du[2] + du[0],\ 2\,j_2 + dv[2] + dv[0])\\
(i_3', j_3') &= (s / 2)\,(2\,i_3 + du[3] + du[2] + du[1] + du[0],\ 2\,j_3 + dv[3] + dv[2] + dv[1] + dv[0])
\end{aligned}$$

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_of_sprite_warping_points 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:

$$\begin{aligned}
(i_1'', j_1'') = (&16\,(i_0 + W') + ((W - W')\,(r\,i_0' - 16\,i_0) + W'\,(r\,i_1' - 16\,i_1))\ //\ W,\\
&16\,j_0 + ((W - W')\,(r\,j_0' - 16\,j_0) + W'\,(r\,j_1' - 16\,j_1))\ //\ W)\\
(i_2'', j_2'') = (&16\,i_0 + ((H - H')\,(r\,i_0' - 16\,i_0) + H'\,(r\,i_2' - 16\,i_2))\ //\ H,\\
&16\,(j_0 + H') + ((H - H')\,(r\,j_0' - 16\,j_0) + H'\,(r\,j_2' - 16\,j_2))\ //\ H)
\end{aligned}$$

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:

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

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

7.8.5 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 in the following. These quantities are then used for sample reconstruction as specified in subclause 7.8.6. The following notations are used to simplify the description:

$$\begin{aligned}
I &= i - i_0,\\
J &= j - j_0,\\
I_c &= 4\,i_c - 2\,i_0 + 1,\\
J_c &= 4\,j_c - 2\,j_0 + 1,
\end{aligned}$$

When no_of_sprite_warping_point == 0,

$$\begin{aligned}
(F(i, j), G(i, j)) &= (s\,i, s\,j),\\
(F_c(i_c, j_c), G_c(i_c, j_c)) &= (s\,i_c, s\,j_c).
\end{aligned}$$

When no_of_sprite_warping_point == 1 and sprite_enable == 'static,

$$\begin{aligned}
(F(i, j), G(i, j)) &= (i_0' + sI,\ j_0' + s\,J),\\
(F_c(i_c, j_c), G_c(i_c, j_c)) &= i_0'\ ///\ 2 + s\,(i_c - i_0 / 2),\ j_0'\ ///\ 2 + s\,(j_c - j_0 / 2)).
\end{aligned}$$

When no_of_sprite_warping_point == 1 and sprite_enable == 'GMC',

$$\begin{aligned}
(F(i, j), G(i, j)) &= (i_0' + sI,\ j_0' + s\,J),\\
(F_c(i_c, j_c), G_c(i_c, j_c)) &= (((i_0' >> 1)\ |\ (i_0' \,\&\, 1)) + s\,(i_c - i_0 / 2),\ ((j_0' >> 1)\ |\ (j_0' \,\&\, 1)) + s\,(j_c - j_0 / 2))
\end{aligned}$$

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/DOGOOJA Y. ADAMANE
ANSI Store order X0?445? Downloaded: 7/18/2007 3:58:21 PM ET
Single user license only. Copying and networking prohibited.

When no_of_sprite_warping_points == 2,

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

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:

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

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

When no_of_sprite_warping_points == 3,

$$
\begin{aligned}
(F(i, j), G(i, j)) \quad & = (i_0' + ((-r\, i_0' + i_1'')\, H'\, I + (-r\, i_0' + i_2'') W'\, J) \;///\; (W'H'r), \\
& \quad j_0' + ((-r\, j_0' + j_1'')\, H'\, I + (-r\, j_0' + j_2'') W'\, J) \;///\; (W'H'r)), \\
(F_c(i_c, j_c), G_c(i_c, j_c)) \quad & = (((-r\, i_0' + i_1'')\, H'\, I_c + (-r\, i_0' + i_2'') W'\, J_c + 2\, W'H' r\, i_0' - 16W'H') \;///\; (4W'H'r), \\
& \quad ((-r\, j_0' + j_1'')\, H'\, I_c + (-r\, j_0' + j_2'') W'\, J_c + 2\, W'H' r\, j_0' - 16W'H') \;///\; (4W'H'r)).
\end{aligned}
$$

According to the definition of W' and H', the computation of these functions can be simplified by dividing the denominator and numerator of division beforehand by W' (when $W' < H'$) or H' (when $W' \geq H'$). 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 $W' \geq H'$ (i.e. $\alpha \geq \beta$) the above equations can be rewritten as:

$$
\begin{aligned}
(F(i, j), G(i, j)) \quad & = (i_0' + (((-r\, i_0' + i_1'')\, I + (-r\, i_0' + i_2'')\, 2^{\alpha-\beta} J + 2^{\alpha+\rho-1}) >> (\alpha+\rho)), \\
& \quad j_0' + (((-r\, j_0' + j_1'')\, I + (-r\, j_0' + j_2'')\, 2^{\alpha-\beta} J + 2^{\alpha+\rho-1}) >> (\alpha+\rho))), \\
(F_c(i_c, j_c), G_c(i_c, j_c)) \quad & = ((((-r\, i_0' + i_1'')\, I_c + (-r\, i_0' + i_2'')\, 2^{\alpha-\beta} J_c + 2W' r\, i_0' - 16W' + 2^{\alpha+\rho+1}) >> (\alpha+\rho+2), \\
& \quad ((-r\, j_0' + j_1'')\, I_c + (-r\, j_0' + j_2'')\, 2^{\alpha-\beta} J_c + 2W' r\, j_0' - 16W' + 2^{\alpha+\rho+1}) >> (\alpha+\rho+2)).
\end{aligned}
$$

When no_of_sprite_warping_point == 4,

$$
\begin{aligned}
(F(i, j), G(i, j)) \quad & = ((a\, I + b\, J + c) \;///\; (g\, I + h\, J + D\, W H), \\
& \quad (d\, I + e\, J + f) \;///\; (g\, I + h\, J + D\, W H)), \\
(F_c(i_c, j_c), G_c(i_c, j_c)) \quad & = ((2\, a\, I_c + 2\, b\, J_c + 4\, c - (g\, I_c + h\, J_c + 2\, D\, W H)\, s) \;///\; (4\, g\, I_c + 4\, h\, J_c + 8D\, W H), \\
& \quad (2\, d\, I_c + 2\, e\, J_c + 4\, f - (g\, I_c + h\, J_c + 2\, D\, W H)\, s) \;///\; (4\, g\, I_c + 4\, h\, J_c + 8D\, W H))
\end{aligned}
$$

where

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

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 transform. The usage of a 64 bit floating point representation should be sufficient in such case.

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/DOUGLAS ADAMS
ANSI Store order X317445; Downloaded: 7/18/2007 3:58:21 PM ET
Single user license only. Copying and networking prohibited.

7.8.5 Sample reconstruction

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

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

when sprite_enable == 'static', and

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

when sprite_enable == 'GMC', where Y_{00}, Y_{01}, Y_{10}, Y_{11} represent the luminance sample in the sprite or reference VOP 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 $r_j = G(i, j) - (G(i, j)///s)s$. See subclause 7.6.2 for the definition of rounding_control. Figure 7-31 illustrates this process.

In case any of Y_{00}, Y_{01}, Y_{10} and Y_{11} lies outside the luminance binary mask of the sprite or reference VOP, it shall be obtained by the padding process as defined in subclause 7.6.1. Additionally, in the case when the current VOP is an rectangular S(GMC)-VOP and any of Y_{00}, Y_{01}, Y_{10} and Y_{11} lies outside the luminance bounding rectangle of the reference VOP, the reconstructed sample value is obtained as described in subclause 7.6.4.

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, 255].

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

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

when sprite_enable == 'static', and

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

when sprite_enable == 'GMC', where C_{00}, C_{01}, C_{10}, C_{11} represent the chrominance sample in the sprite or reference VOP 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 chrominance binary mask of the sprite or reference VOP, it shall be obtained by the padding process as defined in subclause 7.6.1. Additionally, in the case when the current VOP is an rectangular S(GMC)-VOP and any of C_{00}, C_{01}, C_{10} and C_{11} lies outside the chrominance bounding box of the reference VOP, the reconstructed sample value is obtained as described in subclause 7.6.4.

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

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

when sprite_enable == 'static', and

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

when sprite_enable == 'GMC', where A_{00}, A_{01}, A_{10}, A_{11} represent the grayscale alpha sample in the sprite or reference VOP 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 luminance binary mask of the sprite or reference VOP, it shall be obtained by the padding process as defined in subclause 7.6.1. Additionally, in the case when the current VOP is an rectangular S(GMC)-VOP and any of A_{00}, A_{01}, A_{10} and A_{11} lies outside the grayscale alpha bounding box of the reference VOP, the reconstructed sample value is obtained as described in subclause 7.6.4.

When sprite_enable == "static", the reconstructed value of luminance binary mask sample $BY(i,j)$ shall be computed following the identical process for the luminance sample. 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 sampe transparent is ecual to 0. If the computed value is bigger or equal to 128, $BY(i, j)$ is defined as opaque. Otherwise, $BY(i, j)$ is defines as transparent. The chrominance binary mask samples shall be reconstructed by decimating of the corresponding 2 x 2 adjacent luminance binary mask samples as specified in subclause 7.6.1.4.

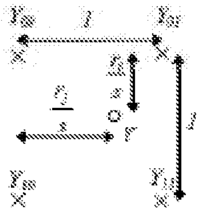


Figure 7-31 – Pixel value interpolation (it is assumed that sprite samples are located on an integer grid)

7.8.7 GMC decoding

7.8.7.1 GMC prediction

For S(GMC)-VOPs, the pixels in a macroblock (including pixels in the grayscale alpha MB) are predicted using GMC prediction when mcsel == '1'. The predicted macroblock is obtained by applying warping to the reference VOP. The prediction error is decoded by the same method as in usual interframe motion compensation. The prediction based on warping is performed using sprite reference point decoding described in clause 7.8.4, sub-pel warping described in clause 7.8.5, and sample reconstruction described in clause 7.8.6.

7.8.7.2 Texture and shape decoding in S(GMC)-VOP

Decoding of texture and shape in S(GMC)-VOPs follows the same rule as those in P-VOPs. Global warping parameters are not used for shape decoding in S(GMC)-VOPs. Macroblocks with mcsel == '1' are regarded as intra-coded macroblocks in shape motion vector decoding since they do not have their own motion vector.

Overlapped block MC is disabled over the boundary between macroblocks with different values for mcsel.

When interlaced == '1', macroblocks with mcsel == "1" are predicted using frame prediction (i.e. the warping method is identical with the case when interlaced == '0').

7.8.7.3 Motion vector decoding

Motion vectors of macroblocks with mcsel == '0' and not_coded == '0' in S-VOPs are decoded using the same rules as P-VOPs. Although macroblocks with mcsel == '1' do not have their own *block* motion vectors, they have *pel-wise* motion vectors for sprite warping obtained from global motion parameters. The candidate motion vector predictor from the reference macroblock with mcsel == '1' is obtained as the *averaged value* of the pel-wise motion vectors in the macroblock. Note that any reference macroblock outside video packet is treated as transparent. In case the frame-based direct mode in B-VOPs, the motion vector of the co-located macroblock with mcsel == '1' is also obtained as the averaged value of the pel-wise motion vectors in the macroblock when the most recenty decoded VOP is S(GMC)-VOP.

Note : Averaged value of pel-wise vectors is calculated usig the all luminance pixels in he macroblock, that is 256 pixels are used (Nb is always 256 into the following expression).

$$AMV_x = \left(\sum_y \sum_x MV_x(x,y) \right) / N_b$$

$$AMV_y = \left(\sum_y \sum_x MV_y(x,y) \right) / N_b$$

$MVx(x,y)$:	Horizontal motion vector at (x,y)
$MVy(x,y)$:	Vertical motion vectbr at (x,y)
N_b:	Number of pixels in the reference block
$AMVx$:	Averaged value of horizontal motion vectors in the block
$AMVy$:	Averaged value of vertical motion vectors in the block

Here,

Since the *AMVs* have fractional values, they are quantized to half integer values when quarter_sample == '0', and quarter integer values when quarter_sample == '1', using the "//" operator. For example, values within [0.0, 0.25) are rounded to 0, [0.25, 0.75) are rounded to 0.5, and [0.75, 1.0] are rounded to 1.0 when quarter_sample == '0'. Values within [0.0, 0.125) are rounded to 0, [0.125, 0.375) are rounded to 0.25, [0.375, 0.625) are rounded to 0.5, [0.625, 0.875) are rounded to 0.75, and [0.875, 1.0] are rounded to 1.0 when quarter_sample == '1'.

If the quantized AMV is outside the motion vector range specified by f_code, it is clipped in the range.

The operation above is performed independently for horizontal and vertical componens.

For example, if the left block is coded in GMC (Figure V2 -24), the candidate predictor is obtaned as the averaged value of the pel-wise motion vectors in the left block.

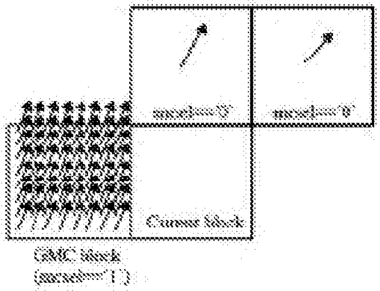


Figure V2 -24 – Pel-wise motion vectors of GMC blocks

7.9 Generalized scalable decoding

This subclause specfies the additional decoding process required for decoding scalabe coded video.

The scalability framework is referred to as generalized scalability which includes he spatial and the temporal scalabilities. The temporal scalability offers scalability of the temporal resolution, and the spatial scalability offers scalability of the spatial resolution. Each type of scalability involves more than one layer. In the case of two layers, consisting of a lower layer and a higher layer, the lower layer is referred to as the reference layer and the higher layer is called the enhancement layer.

In the case of temporal scalability, both rectangular VOPs as well as arbitrary shaped VOPs are supported. In the case of spatial scalability, only rectangular VOPs are supported. Figure 7-32 shows a high level decoder structure for generalized scalability.

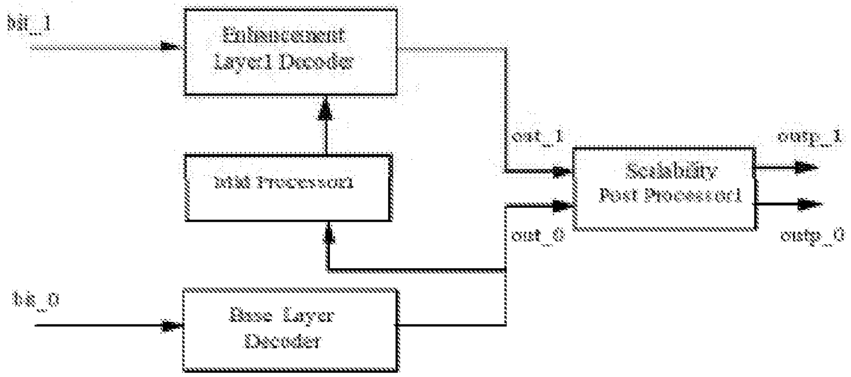


Figure 7-32 -- High level decoder structure for generalized scalability

The base layer and enhancement layer bitstreams are inpt for decoding by the corresponding base layer decoder and enhancement layer decoder.

When spatial scalability is to be performed, mid processor 1 performs spatial up or down sampling of input. The scalability post processor performs any necessary operations such as spatial up or down sampling of the decoded reference layer for display resulting at outp_0 while the enhancement layer without resolution conversion may be output as outp_1.

When temporal scalability is to be performed, the decoding of base and enhancement layer bitstreams occurs in the corresponding base and enhancement layer decoders as shown. In this case, mid processor 1 does not perform any spatial resolution conversion. The post processor simply outputs the base layer VOPs without any conversion, but temporally multiplexes the base and enhancement layer VOPs to produce higher temporal resolution enhancement layer.

The reference VOPs for prediction are selected by ref_seect_code as specified in Table 7-13 and Table 7-14. In coding of P-VOPs belonging to an enhancement layer, the forward reference is one of the following four: the most recently decoded VOP of enhancement layer, the most recent VOP of the reference layer in display order, the next VOP of the reference layer in display order, or the temporaly coincident VOP in the reference layer.

In B-VOPs, the forward reference is one of the following two: the most recently decoded enhancement VOP or the most recent reference layer VOP in display order. The backward reference is one of the following three: the temporally coincident VOP in the reference layer, the most recent reference layer VOP in display order, or the next reference layer VOP in display order.

In case of hierarchy_type == 1(temporal scalability), the default value of ref_select_code is "00". The value of vop_coded should be "1" for the first VOP in an enhancement layer if the value of video_object_layer_shape is '00'.

In case of hierarchy_type == 0(spatial scalability), the default value of ref_select_code s "11" for P-VOP and "00" for B-VOP. The value of vop_coded should be "1" for the first VOP in an enhancement layer if the value of video_object_layer_shape is '00'.

Table 7-13 -- Prediction reference choices in enhancement layer P-VOPs for scalability

ref_select_code	forward prediction reference
00	Most recently decoded enhancement VOP belonging to the same layer for which "vop_coded == 1".
01	Most recently VOP in display order belonging to the reference layer for which "vop_coded == 1".
10	Next VOP in display order belonging to the reference layer.
11	Temporally coincident VOP in the reference layer (no motion vectors)

NOTE 1 — The value "10" is prohibited if "vop_coded = 0" for the next VOP in display order belonging to the reference layer.

NOTE 2 — The value "11" is prohibited if "vop_coded = 0" for the temporally coincident VOP in the reference layer.

Table 7-14 -- Prediction reference choices in enhancement layer B-VOPs for scalability

ref_select_code	forward temporal reference	backward temporal reference
00	Most recently decoded enhancement VOP of the same layer for which "vop_coded == 1".	Temporally coincident VOP in the reference layer (no motion vectors)
01	Most recently decoded enhancement VOP of the same layer for which "vop_coded == 1".	Most recent VOP in display order belonging to the reference layer.
10	Most recently decoded enhancement VOP of the same layer for which "vop_coded == 1".	Next VOP in display order belonging to the reference layer.
11	Most recent VOP in display order belonging to the reference layer for which "vop_coded == 1".	Next VOP in display order belonging to the reference layer.

The following combinations of video_object_layer_shape in the base and enhancement layer and enhancement_type in the enhancement_layer are supported.

	Base Layer	Enhancement Layer	Enhancement _type	status
hierarchy_type=1	rectangular	rectangular	0	supported
	rectangular	binary shape	1	supported
	binary Shape	binary shape	0/1	supported
	binary Shape	rectangular	-	forbidden
	binary only	binary only	-	forbidden
hierarchy_type=0 && video_object_layer_ver_id = 0001	rectangular	rectangular	0	supported
	rectangular	binary shape	-	forbidden
	binary shape	binary shape	-	forbidden
	binary shape	rectangular	-	forbidden
	binary only	binary only	-	forbidden
hierarchy_type=0 && video_object_layer_ver_id = 0010	rectangular	rectangular	0	supported
	rectangular	binary shape	1	supported
	binary shape	binary shape	0/1	supported
	binary shape	rectangular	-	forbidden
	binary only	binary only	-	supported

7.9.1 Temporal scalability

Temporal scalability involves two layers, a lower layer and an enhancement layer. Both the lower and the enhancement layers process the same spatial resolution. The enhancement layer enhances the temporal resolution of the lower layer and if temporally remultiplexed with the lower layer provides full temporal rate.

7.9.1.1 Base layer and enhancement layer

In the case of temporal scalability, the decoded VOPs of the enhancement layer are used to increase the frame rate of the base layer. Figure 7-33 shows a simplified diagram of the motion compensation process for the enhancement layer using temporal scalability.

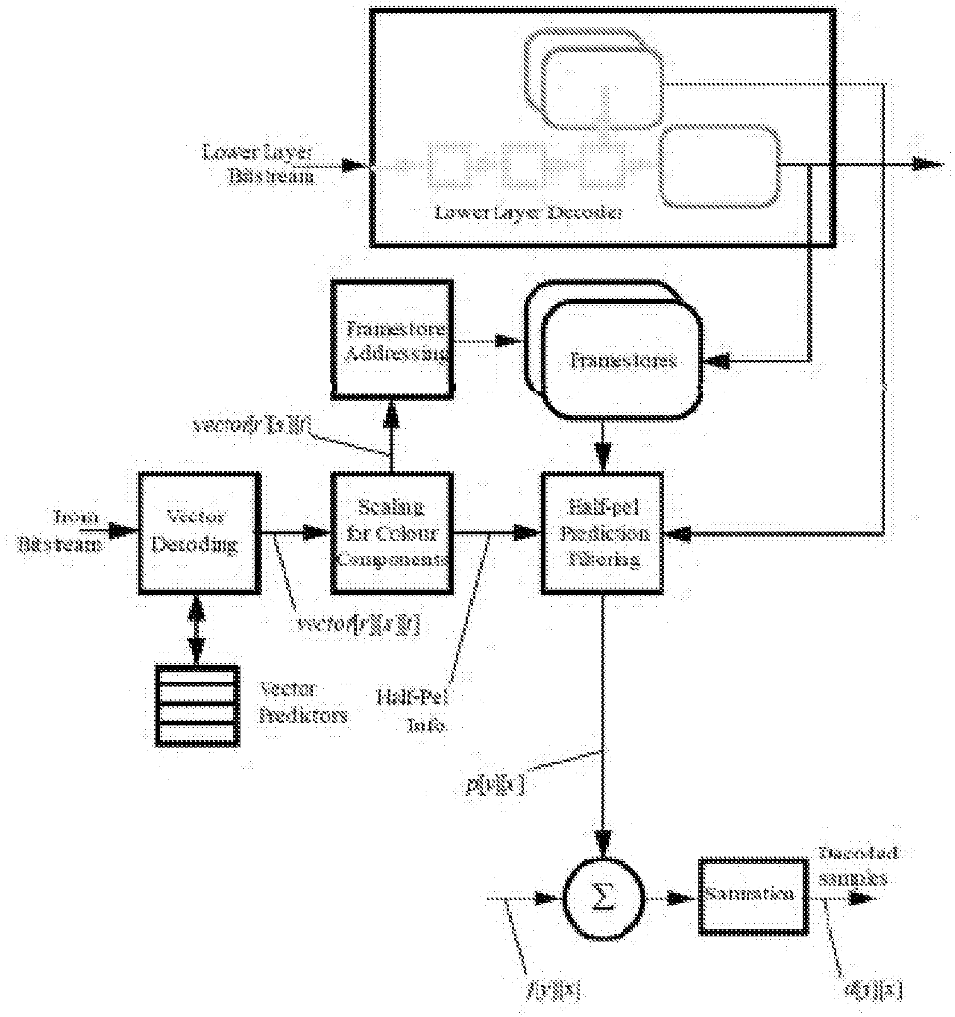


Figure 7-33 -- Simplified motion compensation process for temporal scalability

© ISO/IEC 2001 – All rights reserved
Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/[illegible]
Single user licence only. Copying and networking prohibited.

Predicted samples p[y][x] are formed either from frame stores of the reference layer or from frame stores of enhancement layer. The difference data samples f[y][x] are added to p[y][x] to form the decoded samples d[y][x].

There are two types of enhancement structures indicated by the 'enhancement_type' flag. When the value of enhancement_type is "1", the enhancement layer increases the temporal resolution of a partial region of the reference layer. When the value of enhancement_type is "0", the enhancement layer increases the temporal resolution of an entire region of the reference layer.

The value of video_object_layer_width in base layer and enhancement layer shall be identical if both layers are rectangular. The value of video_object_layer_height in base layer and enhancement layer shall be identical if both layers are rectangular.

7.9.1.2 Base layer

The decoding process of the base layer is the same as non-scalable decoding process.

7.9.1.3 Enhancement layer

The VOP of the enhancement layer is decoded as either I-VOP, P-VOP or B-VOP. The shape of the VOP is either rectangular (video_object_layer_shape is "00") or arbitrary (video_object_layer_shape is "01"). B-VOP in base layer shall not be used as a reference for enhancement layer VOP although B-VOP in enhancement layer can be a reference for enhancement layer VOP.

7.9.1.3.1 Decoding of I-VOPs

The decoding process of the I-VOPs in the enhancement layer is the same as the non-scalable decoding process. ref_layer_id, ref_layer_sampling_direc, hor_sampling_factor_n, horsampling_factor_m, vertical_sampling_factor_n, vertical_sampling_factor_m and ref_select_code are ignored in the temporal scalability I-VOPs.

7.9.1.3.2 Decoding of P-VOPs

The reference layer is indicated by ref_layer_id in Video Object Layer class. Other decoding process is the same as non-scalable P-VOPs except the process specified in subclauses 7.9.1.3.4 and 7.91.3.5.

For P-VOPs, the ref_select_code is either "00", "01" or "10".

When the value of ref_select_code is "00", the prediction reference is set by the most recently decoded VOP belonging to the same layer.

When the value of ref_select_code is "01", the prediction reference is set by the previous VOP in display order belonging to the reference layer.

When the value of ref_select_code is "10", the prediction reference is set by the next VOP in display order belonging to the reference layer

In the case of hierarchy_type=1(temporal scalability), hor_sampling_factor_n, hor_sampling_factor_m, vert_sampling_factor_n and vert_sampling_factor_m are ignored.

7.9.1.3.3 Decoding of B-VOPs

The reference layer is indicated by ref_layer_id in Video Object Layer class. Other decoding process is the same as non-scalable B-VOPs except the process specified in subclauses 7.9.1.3.4 and 7.91.3.5.

For B-VOPs, the ref_select_code is either "01", "10" or "11".

When the value of ref_select_code is "01", the forward prediction reference is set by the most recently decoded VOP belonging to the same layer and the backward prediction reference is set by the previous VOP in display order belonging to the reference layer.

When the value of ref_select_code is "10", the forward prediction reference is set by the most recently decoded VOP belonging to the same layer, and the backward prediction reference is set by the next VOP in display order belonging to the reference layer.

When the value of ref_select_code is "11", the forward prediction reference is set by the previous VOP in display order belonging to the reference layer and the backward prediction reference is set by the next VOP in display order belonging to the reference layer. The picture type of the reference VOP shall be either I or P (vop_coding_type = "00" or "01").

When the value of ref_select_code is "01" or "10", direct mode is not allowed. modb shall always exist in each macroblock, i.e. the macroblock is not skipped even if the co-located macroblock is skipped.

In the case of hierarchy_type=1(temporal scalability), hor_sampling_factor_n, hor_sampling_factor_m, vert_sampling_factor_n and vert_sampling_factor_m are ignored.

7.9.1.3.4 Decoding of arbitrary shaped VOPs

The vop_shape_coding_type for enhancement layer is '0' when vop_coding_type is '00'. Otherwise, the vop_shape_coding_type for enhancement layer is '1'. Prediction for arbitrary shape in P-VOPs or in B-VOP is same as the one in the base layer (see subclause 7.5.2.1.2).

For arbitrary shaped VOPs with the value of enhancement_type being "1", the shape of the reference VOP is defined as an all opaque rectangle whose size is the same as the reference layer when the shape of the reference layer is rectangular (video_object_layer_shape = "00").

When the value of ref_select_code is "11" and the value of enhancement_type is "1", modb shall always exist in each macroblock, i.e. the macroblock is not skipped even if the co-located macroblock is skipped.

7.9.1.3.5 Decoding of backward and forward shape

Backward shape and forward shape are used in the background composition process specified in subclause 8.1. The backward shape is the shape of the enhanced object at the next VOP in display order belonging to the reference layer. The forward shape is the shape of the enhanced object at the prevous VOP in display order belonging to the reference layer.

For the VOPs with the value of enhancement_type being "1", backward shape is decoded when the load_backward_shape is "1" and forward shape is decoded when load_forward_shape is "1".

When the value of load_backward_shape is "1" and the value of load_forward_shape is "0", the backward shape of the previous VOP is copied to the forward shape for the current VOP. When the value of load_backward_shape is "0", the backward shape of the previous VOP is copied to the backward shape for the current VOP and the forward shape of the previous VOP is copied to the forward shape for the current VOP.

The decoding process of backward and forward shape is the same as the decoding process for the shape of I-VOP with binary only mode (video_object_layer_shape = "10").

7.9.2 Spatial scalability

7.9.2.1 Binary shape spatial scalability

Refer to the subclause 7.6.4.

7.9.2.2 Texture Spatial Scalability

7.9.2.3 Base Layer and Enhancement Layer

In the case of spatial scalability, the enhancement bitstream is used to increase the resolution of the image. When the output with the lowest resolution is required, only the base layer is decoded. When the output with higher resolution isrequired, both the base layer and the enhancement layer are decoded.

Figure 7-34 is a diagram of the video decoding process with spatial scalability.

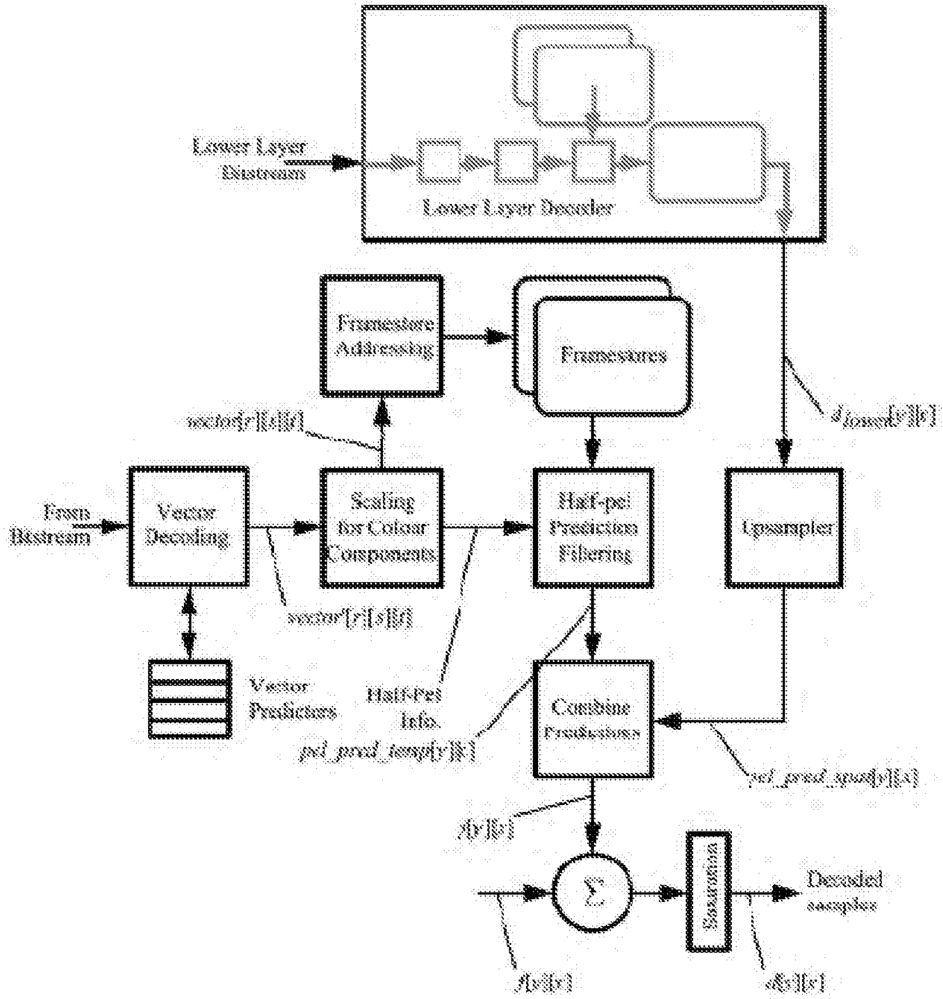


Figure 7-34 – Simplified motion compensation process for spatial scalability

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/PODJA Y. ADAMANE
ANSI Store order EX074451 Downloaded 7/18/2007 3:56:21 PM ET
Single user license only. Copying and networking prohibited.

7.9.2.4 Decoding of Base Layer

The decoding process of the base layer is the same as nonscalable decoding process.

7.9.2.5 Prediction in the enhancement layer

A motion compensated temporal prediction is made from reference VOPs in the enhancement layer. In addition, a spatial predicton is formed from the lower layer decoded frame ($d_{lower}[y][x]$). These predictions are selected individually or combined to form the actual prediction.

In the enhancement layer, the forward prediction in P-VOP and the backward prediction in B-VOP are used as the spatial prediction. The reference VOP is set to the temporally coincident VOP in the reference layer. The forward prediction in B-VOP is used as the temporal prediction from the enhancement layer VOP. The reference VOP is set to the most recently decoded VOP of the enhancement layer. The interpolated prediction in B-VOP is the combination of these predictions. If the reference VOP has arbitrary shape, padding process shall be done before resampling.

In the case that a macroblock is not coded, either because the entire macroblock is skipped or the specific macroblock 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].

7.9.2.6 Formation of spatial prediction

Forming the spatial prediction requires definition of the spatial resampling process. The formation is performed at the mid-processor. The resampling process is defined for a whole VOP, however, for decoding of a macroblock, only the 16x16 region in the upsampled VOP, which corresponds to the position of this macroblock, is needed.

The spatial prediction is made by resampling the reconstructed VOP in the reference layer to the same sampling grid as the enhancement layer. In the first step, the reconstructed VOP is subject to vertical resampling. Then, the vertically resampled image is subject to horizontal resampling. If the reference VOP in the reference lower layer has rectangular shape, the reference VOP is re-sampled without padding process. If the reference VOP in the reference lower layer has arbitrary shape, padding process shall be done before the re-sampling process. The padding process in the formation of spatial prediction is identical to the padding process in the base layer, which is described in subclause 7.5.

In the case of arbitrary shape VOPs, its size and locations change from time to time. In order to ensure the formation of the spatial prediction, it is necessary to identify the location of the resampled VOPs in the absolute coordinate system. If the reference layer is rectangular in shape, the spatial reference of the resampled VOP is set to (0,0).

In case of arbitrary shaped VOP, the VOP location of each layer should conserve the absolute location as described in Figure V2 - 25.

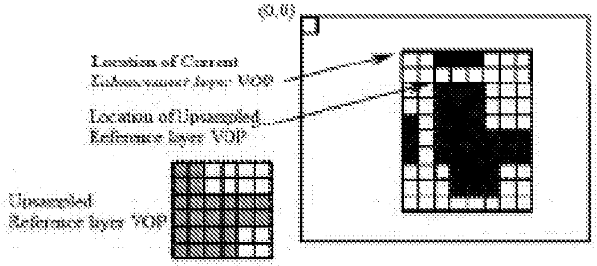

(0,0)
Location of Current Enhancement layer VOP
Location of Upsampled Reference layer VOP
Upsampled Reference layer VOP

Figure V2 - 25 – VOP location of each layer

7.9.2.6.1 Vertical resampling

The image subject to vertical resampling, $d_{lower}[y][x]$, is resampled to the enhancement layer vertical sampling grid using linear interpolation between the sample sites according to the following formula, where vert_pic is the resulting image:

vert_pic[y_h][x] = (16 - phase) * d_{lower}[y1][x] + phase * d_{lower}[y2][x]
where
y_h = output sample coordinate in ver_pic
y1 = (y_h * vertical_sampling_factor_n) / vertical_sampling_factor_n
y2 = y1 + 1 if y1 < video_object_layer_height - 1
y1 otherwise
phase = (16 * ((y_h * vertical_sampling_factor_m) %
vertical_sampling_factor_n)) // vertical_sampling_factor_n

where video_object_layer_height is the height of the reference VOL.

If the reference layer VOP is arbitrary shaped VOP, the following definition should be used for y1 and y2.

y1 = MIN(MAX((y_h * vertical_sampling_factor_m) / vertical_sampling_factor_n,
VOP_vertical_mc_spatial_ref), VOP_vertical_mc_spatial_ref + vop_height-1)
y2 = MIN(MAX((y_h * vertical_sampling_factor_m) / vertical_sampling_factor_n +1,
VOP_vertical_mc_spatial_ref), VOP_vertical_mc_spatial_ref + vop_height-1)

where vop_height is the height of the reference VOP.

Samples which lie outside the vertically upsampled reconstructed frame which are required for upsampling are obtained by border extension of the vertically upsampled reconstructed frame.

NOTE In case of rectangular shape, the calculation of phase assumes that the sample position in the enhancement layer at yh = 0 is spatially coincident with the first sample position of the lower layer. It is recognised that this is an approximation for the chrominance component if the chroma_format == 4:2:0.

7.9.2.6.2 Horizontal resampling

The image subject to horizontal resampling, $vert\_pic[y][x]$, is resampled to the enhancement layer horizontal sampling grid using linear interpolation between the sample sites according to the following formula, where hor_pic is the resulting image:

hor_pic[y][xh] = ((16 - phase) * vert_pic[y][x1] + phase * vert_pic[y][x2]) // 256
where
xh = output sample coordinate in hor_pic
x1 = (xh * horizontal_sampling_factor_m) / horizontal_sampling_factor_n
x2 = x1 + 1 if x1 < video_object_layer_width - 1
x1 otherwise
phase = (16 * ((xh * horizontal_sampling_factor_m) %
horizontal_sampling_factor_n)) // horizontal_sampling_factor_n

where video_object_layer_width is the width of the reference VOL.

If the reference layer VOP is arbitrary shaped VOP, the following definition should be used for x1 and x2.

x1 = MIN(MAX((x_h * horizontal_sampling_factor_m) / horizontal_sampling_factor_n,
vop_horizontal_mc_spatial_ref), vop_horizontal_mc_spatial_ref + vop_width-1)
x2 = MIN(MAX((x_h * horizontal_sampling_factor_m) / horizontal_sampling_factor_n +1,
vop_horizontal_mc_spatial_ref), vop_horizontal_mc_spatial_ref + vop_width-1)

where vop_width is the width of the reference VOP.

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/ROOJA Y. ADAWANE
ANSI Store order X107445 Downloaded: 7/18/2007 3:56:21 PM ET
Single user license only. Copying and networking prohibited.

Samples which lie outside the lower layer reconstructed frame which are required for upsampling are obtained by border extension of the lower layer reconstructed frame.

7.9.2.7 Selection and combination of spatial and temporal predictions

The spatial and temporal predictions can be selected or combined to form the actual prediction in B-VOP. The spatial prediction is referred to as "backward prediction", while the temporal predictior is referred to as "forward prediction". The combination of these predictions can be used as "interpolate prediction". In the case of P-VOP, only the spatial prediction (prediction from the reference layer) can be used as the forward prediction. The prediction in the enhancement layer is defined in the following formulae.

pel_pred[y][x] = pel_pred_temp[y][x] (forward in B-VOP)
pel_pred[y][x] = pel_pred_spat[y][x] = hor_pict[y][x] (forward in P-VOP and backward in B-VOP)
pel_pred[y][x] = (pel_pred_temp[y][x] + pel_pred_spat[y][x])/2 (Interpolate in B-VOP)

pel_pred_temp[y][x] is used to denote the temporal prediction (formed within the enhancement layer). pel_pred_spat[y][x] is used to denote the prediction formed from the lower layer. pe_pred[y][x] is denoted the resulting prediction.

7.9.2.8 Decoding process of enhancement layer

The VOP in the enhancement layer is decoded as either I-VOP, P-VOP or B-VOP.

7.9.2.8.1 Decoding of I-VOPs

The decoding process of the I-VOPs in the enhancement layer is the same as the non_scalable decoding process. ref_layer_id, ref_layer_sampling_direc, hor_sampling_factor_n, horsampling_factor_m, vertical_sampling_facor_n, vertical_sampling_factor_m and ref_select_code are ignored in the spatial scalability I-VOPs.

7.9.2.8.2 Decoding of P-VOPs

In P-VOP, the ref_select_code shall be "11", i.e., the prediction reference is set to the temporally coincident VOP in the reference layer. The reference layer is indicated by ref_layer_id in VideoObjectLayer class. In the case of spatial prediction, the motion vector shall be set to 0 at the decoding process and is not encoded in the bitstream.

A variable length codeword giving information about the macroblock type and the coded block pattern for chrominance is mcbpc. The codewords for mcbpc in the enhancement layer are the same as the base layer and shown in Table B-7. mcbpc shall be included in coded macroblocks.

The macroblock type gives information about the macroblock and which data elemens are present. Macroblock types and included elements in the enhancement layer bitstream are listed in subclause 5.1.1.

In the case of the enhancement layer of spatial scalability, INTER4V shall not be used. The macroblock of INTER or INTER+Q is encoded using the spatial prediction.

7.9.2.8.3 Decoding of B-VOPs

In B-VOP, the ref_select_code shall be '00', i.e., the backward prediction reference is set to the temporally coincident VOP in the reference layer, and the forward prediction reference is set to the most recently decoded VOP in the enhancement layer. In the case of spatial prediction, the motion vector shall be set to 0 at the decoding process and is not encoded in the bitstream.

modb shall be present in coded macroblocks belonging to B-VOPs. The codeword is the same as the base layer and is shown in Table B-3. In case mb_type does not exist the default shall be set to "Forward MC" (prediction from the last decoded VOP in the same reference layer). modb shall be encoded in all macroblocks. If its value is equal to '1', further information is not transmitted for this macroblock. The decoder treats the prediction of this macroblock as forward MC with motion vector equal to zero.

mb_type is present only in coded macroblocks belonging to B-VOPs. The mb_type gives information about the macroblock and which data elements are present. mb_type and included elemens in the enhancement layer bitstream are listed in Table B-5.

In the case of the enhancement layer of spatial scalability, direct mode shall not be used. The decoding process of the forward motion vectors are the same as the base layer.

7.9.2.9 Decoding of binary shape

The decoding process of the binary shape is the same as binary shape spatial scalable decoding process described in subclause 7.5.4.

7.10 Still texture object decoding

The block diagram of the decoder is shown in Figure 7-35.

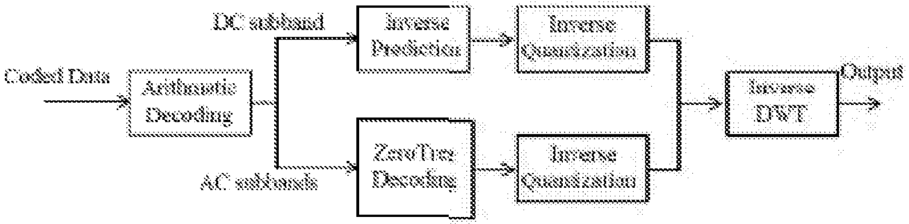


Figure 7-35 -- Block diagram of the decoder

The basic modules of a zero-tree wavelet based decoding scheme are as follows:

Arithmetic decoding of the DC subband using a predictive scheme.

Arithmetic decoding of the bitstream into quantized wavelet coefficients and the significance map for AC subbands.

Zero-tree decoding of the higher subband wavelet coefficients.

Inverse quantisation of the wavelet coefficients.

Composition of the texture using inverse discrete wavelet transform (IDWT).

7.10.1 Decoding of the DC subband

The wavelet coefficients of DC subband are decoded independently from the other bands. First the quantization step size decoded, then the offset value of the quantization indices "band_offset" and the maximum value of the differential quantization indices "band_max_value" are decoded from bitstream. The parameter "band_offset" is negative or zero integer and the parameter "band_max_value" is a positive integer, so only the magnitude of these parameters are read from the bitstream.

The arithmetic model is initialized with a uniform distribution. Then, the differential quantization indices are decoded using the arithmetic decoder in a bitplane-by-bitplane fashion, from MSB to LSB. In each bitplane, the indices are decoded in a raster scan order, starting from the upper left index and ending with the lowest right one. Separate probability model for each bitplane and color component is used. The model is updated with the decoding of each bits of the predicted wavelet quantization index to adopt the probability model to the statistics of DC band.

The "band_offset" is added to all the decoded quantisation indices, and an inverse predictive scheme is applied. Each of the current indices w_X is predicted from three quantisation indices in its neighborhood, i.e. w_A, w_B, and w_C (see Figure 7-36), and the predicted value is added to the current decoded coefficient. That is,

```
if  (|w_A-w_B|) < | w_B-w_C|)
        $\hat{w}_X = w_C$
else
        $\hat{w}_X = w_A$
$w_X = w_r + \hat{w}_X$
```

If any of nodes A, B or C is not in the image, its value is set to zero for the purpose of the inverse prediction. Finally, the inverse quantisation scheme is applied to all decoded values to obtain the wavelet coefficients of DC band.

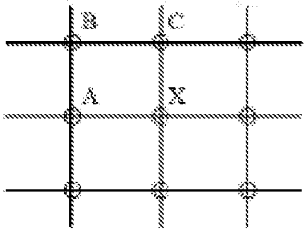


Figure 7-36 -- DPCM decoding of DC band coefficients

7.10.2 ZeroTree Decoding of the Higher Bands

The zero-tree algorithm is based on the strong correlation between the amplitudes of the wavelet coefficients across scales, and on the idea of partial ordering of the coefficients. The coefficient at the coarse scale is called the *parent*, and all coefficients at the same spatial location, and of similar orientation, at the next finer scale are that parent's children. Figure 7-37 shows a wavelet tree where the parents and the children are indicated by squares and connected by lines. *Since the DC subband (shown at the upper left in Figure 7-37) is coded separately using a DPCM scheme, the wavelet trees start from the adjacent higher bands.*

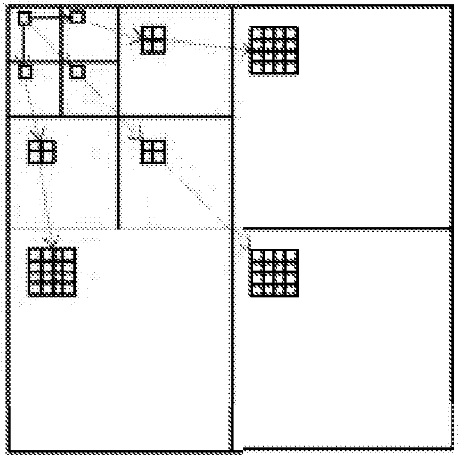


Figure 7-37 -- Parent-child relationship of wavelet coefficients

In transform-based coding, it is typically true that a large percentage of the transform coefficients are quantized to zero. A substantial number of bits must be spent either encoding these zero-valued quantized coefficients, or else encoding the location of the non-zero-valued quantized coefficients. ZeroTree Coding uses a data structure called a *zerotree*, built on the parent-child relationships described above, and used for encoding the location of non-zero quantized wavelet coefficients. The zerotree structure takes advantage of the principle that if a wavelet coefficient at a coarse scale is "insignificant" (quantized to zero), then all wavelet coefficients of the same orientation at the

© ISO/IEC 2001 – All rights reserved
Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/ROOJA, Y. ADMANE
ANSI Store order X337445 Downloaded 7/18/2007 2:56:21 PM ET
Single user license only. Copying and networking prohibited.

same spatial location at finer wavelet scales are also likely to be "insignificant". Zerotrees exist at any tree node where the coefficient is zero and all its descendents are also zero.

The wavelet trees are efficiently represented and coded by scanning each tree from the root in the 3 lowest AC bands through the children, and assigning one of four symbols to each node encountered: *zerotree root (ZTR)*, *value zerotree root (VZTR)*, *isolated zero (IZ)* or *value (VAL)*. A *ZTR* denotes a coefficient that is the root of a zerotree. Zerotrees do not need to be scanned further because it is known that all coefficients in such a tree have amplitude zero. A *VZTR* is a node where the coefficient has a nonzero amplitude, and all four children are zerotree roots. The scan of this tree can stop at this symbol. An IZ identifies a coefficient with amplitude zero, but also with some nonzero descendant. A *VAL* symbol identifies a coefficient with amplitude nonzero, and with some nonzero descendant. The symbols and quantized coefficients are losslessly encoded using an adaptive arithmetic coder. Table 7-15 shows the mapping of indices of the arithmetc decoding model into the zerotree symbols:

Table 7-15 -- The indecing of zerotree symbols

index	Symbol
0	IZ
1	VAL
2	ZTR
3	VZTR

In order to achieve a wide range of scalability levels efficiently as needed by different applications, three different zerotree scanning and associated inverse quantisation methods are employed. The encoding mode is specified in bitstream with quantisation_type field as one of 1) single_quant, 2) multi_quant or 3) bilevel_quant:

Table 7-16 -- The quantisation types

code	quantisation_type
01	single_quant
10	multi_quant
11	bilevel_quant

In single_quant mode, the bitstream contains only one zero-tree map for the wavelet coefficients. After arithmetic decoding, the inverse quantisation is applied to obtain the reconstructed wavelet coefficients and at the end, the inverse wavelet transform is applied to those coefficients.

In multi_quant mode, a multiscale zerotree decoding scheme is employed. Figure 7-38 shows the concept of this technique.

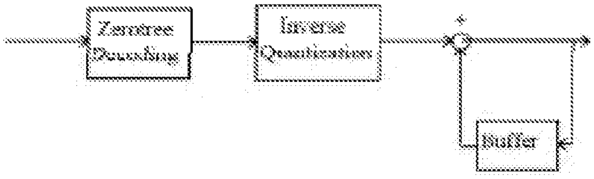


Figure 7-38 -- Multiscale zerotree decoding

The wavelet coefficients of the first spatial (and/or SNR) layer are read from the bitstream and decoded using the arithmetic decoder. Zerotree scanning is used for decoding the significant maps and quantized coefficients and locating them in their corresponding positions in trees. These values are saved in the buffer to be used for quantisation refinement at the next scalability layer. Then, an inverse quantisation is applied to these indices to

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA Y. ADAWANE
ANSI Store order X037445 Downloaded 7/16/2007 2:56:21 PM ET
Single user license only. Copying and networking prohibited.

obtain the quantized wavelet coefficients. An inverse wavelet transform can also be applied to these coefficients to obtain the first decoded image. The above procedure is applied for the next spatial/SNR ayers.

The bilevel_quant mode enables fine granular SNR scalability by encoding the wavelet coefficients in a bitplane-by-bitplane fashion. This mode uses the same zerotree symbols as the multi_quant mode. In this mode, a zero-tree map is decoded for each bitplane, indicating which wavelet coefficients are nonzero relative to that bitplane. The inverse quantization is also performed bitplane by bitplane. After the zero-tree map, additional bits are decoded to refine the accuracy of the previously decoded coefficients.

7.10.2.1 Zerotree Scanning

In all the three quantisation modes, the wavelet coefficients are scanned either in the tree-depth fashion or in the band-by-band fashion. In the tree-depth scanning order all coefficients of each tree are decoded before starting decoding of the next tree. In the band-by-band scanning order, all coefficients are decoded from the lowest to the highest frequency subbands.

Figure 7-39 shows the scanning order for a 16x16 image, with 3 levels of decomposition. In this figure, the indices 0,1,2,3 represent the DC band coefficients which are decoded separately. The remaining coefficients are decoded in the order shown in this figure. As an example, indices 4,5,..., 24 represent one tree. At first, coefficients in this tree are decoded starting from index 4 and ending at index 24. Then, the coefficients in the second tree are decoded starting from index 25 and ending at 45. The third tree is decoded starting from index 46 and ending at index 66 and so on.

0	1	4	67	5	6	68	69	9	10	13	14	72	73	76	77
2	3	130	193	7	8	70	71	11	12	15	16	74	75	78	79
25	88	46	109	131	132	194	195	17	18	21	22	80	81	84	85
151	214	172	235	133	134	196	197	19	20	23	24	82	83	86	87
26	27	89	90	47	48	110	111	135	136	139	140	198	199	202	203
28	29	91	92	49	50	112	113	137	138	141	142	200	201	204	205
152	153	215	216	173	174	236	237	143	144	147	148	206	207	210	211
154	155	217	218	175	176	238	239	145	146	149	150	208	209	212	213
30	31	34	35	93	94	97	98	51	52	55	56	114	115	118	119
32	33	36	37	95	96	99	100	53	54	57	58	116	117	120	121
38	39	42	43	101	102	105	106	59	60	63	64	122	123	126	127
40	41	44	45	103	104	107	108	61	62	65	66	124	125	128	129
156	157	160	161	219	220	223	224	177	178	181	182	240	241	244	245
158	159	162	163	221	222	225	226	179	180	183	184	242	243	246	247
164	165	168	169	227	228	231	232	185	186	189	190	248	249	252	253
166	167	170	171	229	230	233	234	187	188	191	192	250	251	254	255

Figure 7-39 -- Tree depth scanning order of a wavelet block in the all three modes

Figure 7-40 shows that the wavelet coefficients are scanned in the subband by subband fashion, from the lowest to the highest frequency subbands. Shows an example of decoding order for a 16x16 image with 3 levels of decomposition for the subband by subband scanning. The DC band is located at upper left corner (with indices 0, 1,2, 3) and is decoded separately as described in DC band decoding. The remaining coefficients are decoded in the order which is shown in the figure, starting from index 4 and ending at index 255. In multi_quant mode, at first scalability layer, the zerotree symbols and the corresponding values are decoded for the wavelet coefficients of that

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA V. ADAVANE
ANSI Store order X0?4487 Downloaded: 7/18/2007 3:58:21 PM ET
Single user license only. Copying and networking prohibited.

scalability layer. For the next scalability layers, the zerotree map is updated along with the corresponding value refinements. In each scalability layer, a new zerotree symbd is decoded for a coefficent only if it was decoded as ZTR or IZ in previous scalability layer or it is currently in SKIP mode. A node is said o be in SKIP mode when the number of quantisaton refinement levels for the current scalability layer is one. The detailed description of the refinement of quantisaton level is found in subclause 7.10.3. If the coefficient was decoded as VAL in prevous layer and it is not currently in SKIP mode, a VAL symbol is also assigned to it at he current layer and only its refinement value is decoded from bitstream.

0	1	4	7	16	17	28	29	64	65	68	69	112	113	116	117
2	3	10	13	18	19	30	31	66	67	70	71	114	115	118	119
5	8	6	9	40	41	52	53	72	73	76	77	120	121	124	125
11	14	12	15	42	43	54	55	74	75	78	79	122	123	126	127
20	21	32	33	24	25	36	37	160	161	164	165	208	209	212	213
22	23	34	35	26	27	38	39	162	163	166	167	210	211	214	215
44	45	56	57	48	49	60	61	168	169	172	173	216	217	220	221
46	47	58	59	50	51	62	63	170	171	174	175	218	219	222	223
80	81	84	85	128	129	132	133	96	97	100	101	144	145	148	149
82	83	86	87	130	131	134	135	98	99	102	103	146	147	150	151
88	89	92	93	136	137	140	141	104	105	108	109	152	153	156	157
90	91	94	95	138	139	142	143	106	107	110	111	154	155	158	159
176	177	180	181	224	225	228	229	192	193	196	197	240	241	244	245
178	179	182	183	226	227	230	231	194	195	198	199	242	243	246	247
184	185	188	189	232	233	236	237	200	201	204	205	248	249	252	253
186	187	190	191	234	235	238	239	202	203	206	207	250	251	254	255

Figure 7-40 -- The band-by-band scanning order for all three modes

In bilevel_quant mode, the band by band scanning is also employed, similar to the multi_quant mode. When bi-level quantisation is applied, the coefficients that are already found significant are replaced with zero symbols for the purpose of zero-tree forming in later scans.

7.10.2.2 Entropy Decoding

The zero-tree (or type) symbols, quantized coefficient values (magnitude and sign), and residual values (for the multi quant mode) are all decoded using an adaptive arithmetic decoder with a given symbol alphabet. The arithmetic decoder adaptively tracks the statistics of the zerotree symbols and decoded values. For both the single quant and multi quant modes the arithmetic decoder is initialized at the beginning of each color loop for band-by-band scanning and at the beginning of the tree-block loop for tree-depth scanning. Therefore, the decoder is initialized three times in each band for band-by-band scanning and the decoder is initialized once before going to the tree block-loop for tree-depth scanning. In order to avoid start code emulation, the arithmetic encoder always starts with stuffing one bit '1' at the beginning of the entropy encoding. It also stuffs one bit '1' immediately after it encodes every 22 successive '0's. It stuffs one bit '1' to the end of bitstream in the case in which the last output bit of arithmetic encoder is '0'. Thus, the arithmetic decoder reads and discards one bit before starts entropy decoding. During the decoding, it also reads and discards one bit after receiving every 22 successive '0's. The arithmetic decoder reads one bit and discards it if the last input bit to the arithmetic decoder is '0'. To reduce the number of bitplanes in encoding the magnitude of quantized coeffcient values, the encoder subtracts the magnitude by one first. Thus the decoder adds one back after decoding the magnitude of quantized coefficient values.

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA V. AGAWANE
ANSI Store order X207445 Downloaded: 7/18/2007 3:58:21 PM ET
Single user license only. Copying and networking prohibited.

The context models used for SQ and MQ are identical to the ones used in BQ mode.

For both scanning orders in the single quant and multi_quant modes separate probability models are kept for each color and wavelet decomposition layer for the type and sign symbols while separate probability models are kept for each color, wavelet decomposition layer, *and* bitplane for he magnitude and residual symbols. All the models are initialized with a uniform probability distribution.

The models and symbol sets for the non-zerotree type quantities to be decoded are as follows:

Model	Possible Values
Sign	POSITIVE (0), NEGATIVE (1)
Magnitude	0, 1
Residual	0, 1

The possible values of the magnitudes and residuals are only 0 or 1 because each bitplane is being decoded separately. The non-residual values are decoded in two steps. First, the absolute value is decoded in a bitplane fashion using the *magnitude* probability models, then its sign is decoded.

For the decoding of the type symbols different probability models are kept for the leaf and non-leaf coefficients. For the multi quant mode, context modeling, based on the zerotree type of the coefficient in the previous scalability layer, is used. The different zerotree type models and their possible values are as follows:

Context and Leaf/Non-Leaf	Possible Values
INIT	ZTR (2), IZ (0), VZTR (3), VAL (1)
ZTR	ZTR (2), IZ (0), VZTR (3), VAL (1)
ZTR DESCENDENT	ZTR (2), IZ (0), VZTR (3), VAL (1)
IZ	IZ (0), VAL (1)
LEAF INIT	ZTR (0), VZTR (1)
LEAF ZTR	ZTR (0), VZTR (1)
LEAF ZTR DESCENDENT	ZTR (0), VZTR (1)

For the single quant mode only the INIT and LEAF INIT models are used. For the multi quant mode for the first scalability layer only the INIT and LEAF INIT models are used. Subsequent scalability layers in the multi quant mode use the context associated with the type. If a new spatial layer is added then the contexts of all previous leaf band coefficients are switched to the corresponding non-leaf contexts. The coefficients in the newly added bands use the LEAF INIT context. The residual models are usec to decode the coefficient refinements if in the previous layer, a *VZTR or VAL* symbol was assigned. If a node is currently not in SKIP mode (meaning that no refinement is being done for the coefficient – see subclause 7.10.3 on inverse quantisation for details) only the magnitude of the refinements are decoded as these values are always zero or positive integers.

If a node is in SKIP mode, then its new zerotree symbol is decoded from bitstream, but no value is decoded for the node and its value in the current scalability layer is assumed to be zero.

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/[illegible]
Single user licence only. Copying and networking prohibited.

States in Previous Bitplane	Possibilities in current bitplane
ZTR	ZTR, VZTR, IZ, VAL
VZTR	SKIP
IZ	IZ, VAL
VAL	SKIP
DZTR	ZTR, VTRZ, IZ, VAL

For the bi-level quantisation mode, the zero-tree map is decoded for each bitplane, indicating which wavelet coefficients are zeros relative to the current quantisation step size. Different probability models for the arithmetic decoder are used and updated according to the local contexts. For decoding the zerotree symbols, five context models are used, which are dependent on the status of the current wavelet coefficients in the zerotree formed in the previous bitplane decoding. Specifically, the five models correspond to the following contexts of the current wavelet coefficient:

· IZ: the previous zerotree symbol is Isolated Zero.

· VAL: the previous zerotree symbol is Value.

· ZTR: the previous zerotree symbol is Zerotree Root.

· VZTR: the previous zerotree symbol is valued zerotree root.

. DZTR: in previous bitplane, the current coefficient is a descendant of a zerotree root.

The additional symbol DZTR is used for switching the models only, where DZTR refers to the descendant of a ZTR symbol. The context symbols DZTR can be inferred from the decoding process and are not included in the bitstream. They are used for switching the models only. At the beginning of the decoding the first bitplane, the contexts of the coefficients are initialized to be DZ. For the highest subband, only IZ and VAL are possible (no ZTR and VZTR are possible). Therefore, we initialize the arithmetic model for the last band differently (with zero probability for ZTR and VZTR symbols).

For decoding the sign information, another context model (the sign model) is used and updated. For decoding the refinement bits, another statistical model (the refinement model) is used.

Each decomposition levels have their own separate arithmetic models. Therefore, the above decoding process applies to each decomposition levels. All models are initalized at the beginning of coding each bitplane.

After the zero-tree map, additional bits are received to refine the accuracy of the coefficients that are already marked significant by previously received information at the decoder. For each significant coefficient, the 1-bit bi-level quantized refinement values are entropy coded using the arithmetic coder.

7.10.3 Inverse Quantisation

Different quantisation step sizes (one for each color component) are specified for each level of scalability. The quantizer of the DC band is a uniform mid-step quantizer with a dead zone equal to the quantisation step size. The quantisation index is a signed integer number and the quantisation reconstructed value is obtained using the following equation:

$$V = id * Qdc,$$

where V is the reconstructed value, id is the decoded index and Qdc is the quantisation step size.

[illegible]
[illegible]
Single user license only. Copying and networking prohibited.

All the quantizers of the higher bands (in all quantisation modes) are uniform mid-step quantizer with a dead zone 2 times the quantisation step size. For the single quantisation mode, the quantisation index is an signed integer. The reconstructed value is obtained using the following algorithm:

```
if (id == 0)
    V =0;
else if ( id > 0 )
    V = id*Q+Q/2;
else
    V = id*Q-Q/2;
```

where V is the reconstructed value, id is the decoded index and Q is the quantisation step size.

In the multi- quantisation mode each SNR layer within each spatial layer has an associated quantisation step-size value (Q value). These different Q Values are used for SNR scalability. A lower Q Value will result in a more accurate reconstruction.

If a coefficient is in a given spatial layer it is also in all higher numbered spatial layers. SNR scalability may be continued on these coefficients in the higher numbered spatial layers in the same way as is done in the spatial layer the coefficient first arises in. Thus, we can think of all the coefficients which first arise in a particular spatial layer as having a corresponding sequence of Q Values (call it a Q Sequence). The Q Sequence is made up of the quantisation values for all SNR layers in the spatial layer the coefficient first arises in plus the quantisation values in all SNR layers in all higher spatial layers. The order is from lower to higher numbered spatial layers and from lower to higher numbered SNR layers within each spatial layer.

EXAMPLE

Let the quantisation value of the *i*-th spatial layer and the *j*th SNR layer be denoted by *Q(i,j)*. Assume we have the following scenario:

Spatia	SNR Layer		
Layer	0	1	2
0	*Q(0,0)*	*Q(0,1)*	*Q(0,2)*
1	*Q(1,0)*	*Q(1,1)*	*Q(1,2)*
2	*Q(2,0)*	*Q(2,1)*	*Q(2,2)*

The Q Sequence which will be used to quantize all coefficients which first arise in spatiallayer 0 is:

<Q(0,0) Q(0,1) Q(0,2) Q(1,0) Q(1,1) Q(1,2) Q(2,0) Q(2,1) Q(2,2)>

while the sequence for all coefficients first arising in spatiallayer 1 is:

<Q(1,0) Q(1,1) Q(1,2) Q(2,0) Q(2,1) Q(2,2)>

and the sequence for all coefficients first arising in spatial layer 2:

<Q(2,0) Q(2,1) Q(2,2)>.

As in the single- quantisation case we would like to have a uniform quantizer for all layers. Due to the manner in which the Q Values are used to achieve scalability (described below), in order to have a (approximately) uniform quantizer at each layer, we may have to revise the Q Values extracted from the bitstream before reconstruction. This revision is necessary if the Q Values within each Q Sequence are not integer multiples of one another or if Q Value is greater than a Q Value occurring earlier in the Q Sequence.

EXAMPLES

Q Sequences needing no revision: *<24 8 2>* and *<81 81 27>*.

Q Sequences needing revision: *<31 9 2>* (non-integer mulliples) and *<31 162 4>* (increasing Q Value).

If a coefficient's quantisation indices have been zero for all previous scalability layers (spatial and SNR) or if it is the first scalability layer, then the reconstruction is the similar to the single-quantisation mode described above. The difference is in that the refined Q Values may be used instead of the ones extracted from the bitstream. The refinement process is described below in steps 1 and 2. If there has been a non-zero quantisation index in a previous scalability layer then the quantisation index specifies a refinement of the previous quantisaton. The indices are then called residuals. For every coefficient and scalability layer we know (1) the quantisation interval where the coefficient occurred in the last scalability layer (both size and location), (2) the spatial layer the coefficient first arose in (and thus, which Q Sequence to use), (3) the current Q Value and the previous Q Value in the Q Sequence, and (4) the refinment (if any) of the previous Q Value.

The reconstruction of the residual is calculated in the following five steps.

Step 1: Calculation of the Number of Refinement Intervals

The quantisation interval which was indexed in the previous layer is to be partitioned into disjoint intervals. The number of these "refinement" intervals is calculated based solely on the current Q Value (call it *curQ*) and the previous Q Value (call it *prevQ*). Note that *prevQ* may have been revised as mentioned above. Letting *m* be the number of refinement intervals we calculate

m = ROUND(prevQ÷curQ)
where ROUND(x) = MAX(nearest integer of x, 1).

If $m = 1$, no refinement is needed and no value will have been sent. If, at a certain scalability layer, a node has $m=1$ then it is said to be in SKIP mode. Thus, steps 2, 3, and 4 need not be performed for the coefficient.

Step 2: Calculation of the Maximum Refinement Interval Size

Using the number of refinement intervals, the current Q Value, *curQ*, is revised (if necessary).

curQ = CEIL(prevQ ÷ m)
where CEIL rounds up to the nearest integer.

curQ represents the maximum size of the intervals in the partition. Since *prevQ* is the previous layer's *curQ* (see step 5), we see that *prevQ* represents the maximum size of the intervals in the partiton used in the previous scalability layer.

Step 3: Construction of Refinement Partition

Using the values *m* and *curQ* calculated above and the size of quantisation interval where the coefficient occurred in the last scalability layer, we form the refinement partition.

The previous layer's quantisation interval is partitioned into *m* intervals which are of size *curQ* or *curQ-1*. The residual will be one of the values {*0, 1, ..., m-1*} which represent an index into this partition. A lower number index corresponds to an interval in the partition covering lower magnitude values. If the partition is made up of different size intervals (*curQ* and *curQ-1*) then the *curQ* size intervals correspond to the lower indices. Some combination of *m curQ* and *curQ-1* interval sizes are sufficient to cover the previous quantisation interval. From step 2 we know that the previous quantisation interval is of size *prevQ* or *prevQ-1*.

Step 4: Calculation of Reconstructed Value

The interval in the partition indexed by the residual is mapped to the reconstruction value. The reconstruction is just the middle point of the interval in the partiton that the residual indexes. That is, if *PartStart* is the start of the interval in the partitixn which is indexed by the residual, *PartStartSize* is the size of the interval, *sign* is the corresponding sign (knownfrom prior scalability layers), and // is integer division then the reconstructed value is:

ParStart + sign*(PartStartSize-1)//2

Step 5: Assignment of Maximum Size

If there is another scalability layer then *prevQ* is assigned the value of *curQ*.

Note that since steps 1, 2, and 5 depend entirely on the Q Values found in the Q Sequences they only need to be done once in each scalability layer for each Q Sequence being used in the current spatial layer.

FOUR EXAMPLES

In the examples:

1. let the Q Values be Q1 , Q2 , and Q3,
2. let two sample coefficients to be quantized be *C1* and *C2*,
3. let *Cq1* and *Cq2* denote the corresponding quantized coefficients or residuals, and
4. let *iC1* and *iC2* denote the corresponding reconstructed coefficient values.

1. Q Values not in need of revision

Q1 = 24 , Q2 = 8, Q3 = 2,
C1 = 16, and C2 = 28.

At first scalability layer we have

Cq1 = C1/Q1 = 0
iCq1 = 0
Cq2 = C2/Q1 = 1
iCq2 = 36

At second scalability layer we have

prevQ = Q1 = 24
curQ = Q2 = 8
m = R0UND(prevQ÷curQ) = ROUND(24÷8) = 3
curQ = CEIL(prevQ÷m) = CEIL(24÷3) = 8
partition sizes = {8, 8, 8}
Cq1 = C1/curQ = 2
iCq1 = 19
Cq2 = 0 (residual)
iCq2 = 27

At third scalability layer we have

prevQ = curQ = 8
curQ = Q3 = 2
m = R0UND(prevQ÷curQ) = ROUND(8÷2) = 4
curQ = CEIL(prevQ÷m) = CEIL(8÷2) = 4
partition sizes = {2, 2, 2, 2}
Cq1 = 0 (residual)
iCq1 = 16
Cq2 = 2 (residual)
iCq2 = 28

2. Q Values not in need of revision

Q1 = 81, Q2 = 31, Q3 = 27,
C1 = 115 , and C2 = 28.

© ISO/IEC 2001 – All rights reserved
Licensed to UNIVERSITY OF TEXAS AT ARLINGTON [illegible]
Single user licence only. Copying and networking prohibited.

At first scalability layer we have

Cq1 = C1/Q1 = 1
iCq1 = 121
Cq2 = C2/Q1 = 0
iCq2 = 0

At second scalability layer we have

prevQ = Q1 = 81
curQ = Q2 = 81
m = ROUND(prevQ÷curQ) = ROUND(81÷81) = 1
curQ = CEIL(prevQ÷m) = CEIL(81÷1) = 81
partion sizes = {81} (no refinement needed)
Cq1 = 0 (residual)
iCq1 = 121
Cq2 = C2/curQ = 0
iCq2 = 0

At third scalability layer we have

prevQ = curQ = 81
curQ = Q3 = 27
m = ROUND(prevQ÷curQ) = ROUND(81÷27) = 3
curQ = CEIL(prevQ÷m) = CEIL(81÷3) = 27
partion sizes = {27, 27, 27}
Cq1 = 1 (residual)
iCq1 = 121
Cq2 = C2/curQ = 1
iCq2 = 40

3. Q Values in need of revision

Q1 = 31 , Q2 = 9, Q3 = 2.
C1 = 115 , and C2 = 5.

At first scalability layer we have

Cq1 = C1/Q1 = 3
iCq1 = 108
Cq2 = C2/Q1 = 0
iCq2 = 0

At second scalability layer we have

prevQ = Q1 = 31
curQ = Q2 = 9
m = ROUND(prevQ÷curQ) = ROUND(31÷9) = 3
curQ = CEIL(prevQ÷m) = CEIL(31÷3) = 11
partion sizes = {11, 10, 10}
Cq1 = 2 (residual)
iCq1 = 118
Cq2 = C2/curQ = 0
iCq2 = 0

At third scalability layer we have

prevQ = curQ = 11
curQ = Q3 = 2
m = ROUND(prevQ÷curQ) = ROUND(11÷2) = 6
curQ = CEIL(prevQ÷m) = CEIL(11÷6) = 2
partition sizes = {2, 2, 2, 2, 2, 1} if value occurs in level with size 11
partition sizes = {2, 2, 2, 2, 1, 1} if value occurs in evel with size 10
Cq1 = 0 (residual)
iCq1 = 114
Cq2 = C2/curQ = 2
iCq2 =4

4. Q Values in need of revision

Q1 = 81, Q2 = 162, Q3 = 4.
C1 = 115 , and C2 = 5.

At first scalabilty layer we have

Cq1 = C1/Q1 = 1
iCq1 =121
Cq2 = C2/Q1 = 0
iCq2 =0

At second scalability layer we have

prevQ = Q1 = 81
curQ = Q2 = 162
m = ROUND(prevQ÷curQ) = ROUND(81÷162) = 1
curQ = CEIL(prevQ÷m) = CEIL(81÷1) = 81
partition sizes = {81} (no refinement needed)
Cq1 = 0 (not used)
iCq1 =121
Cq2 = C2/curQ = 0 (not used)
iCq2 =0

At third scalabilty layer we have

prevQ = curQ = 81
curQ = Q3 = 4
m = ROUND(prevQ÷curQ) = ROUND(81÷4) = 20
curQ = CEIL(prevQ÷m) = CEIL(81÷20) = 5
partition sizes = {5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4}
Cq1 = 8 (residual)
iCq1 =121
Cq2 = C2/curQ = 1
iCq2 =6

It is important to note that coefficients which first arose in different spatial layers may use different *prevQ* and *curQ* values. They are basically being quantized from different lists. That is, they have different corresponding Q Sequences.

7.10.3.1 Shape adaptive zerotree decoding

When the texture_object_layer_shape is not rectangular or **texture_object_layer_width!=** integer*$2^{wavelet\_decomposition\_levels}$ or **texture_object_layer_height!=** integer*$2^{wavelet\_decomposition\_levels}$, the inverse shape adaptive wavelet transform is chosen to reconstructed the arbitrary-shaped image object or rectangular texture object. Decoding shape adaptive wavelet coefficients is the same as decoding regular wavelet coefficients except keep track of the locations of where to put the decoded wavelet coefficents according to the shape information, or a generated mask. The mask is generated with **texture_object_layer_width*texture_object_layer_height** pixels of value 255 at the upper-left corner of a frame of size w*$2^{wavelet\_decomposition\_levels}$ * h*$2^{wavelet\_decomposition\_levels}$ and the rest of the pixels being value 0, where w is the smallest integer that makes $w*2^{\text{wavelet\_decomposition\_level}} \geq$ **texture_object_layer_width** and h is the smallest integer that makes $h*2^{\text{wavelet\_decomposition\_level}} \geq$ **texture_object_layer_height**. Similar to decoding of regular wavelet coefficients, the decoded zerotree symbols at a lower subband are used to determine whether decoding is needed at higher subbands. The difference is now that some zerotree nodes correspond to the pixel locations outside the shape boundary and no bits are to be decoded for these out_nodes. Root layer is defined as the lowest three AC subbands, leaf layer is defined as the highest three AC subbands. For decompositon level of one, the overlapped root layer and leaf layer shall be treated as leaf layer.

7.10.3.1.1 DC layer

The DC coefficient decoding is the same as that for rectangular image except the following,

1. Only those DC coefficients inside the shape boundary in the DC layer shall be traversed and decoded and DC coefficients outside the shape boundary may be set to zeros.
2. For the inverse DC prediction in the DC layer, if a reference coefficient (Figure 7-36) in the prediction context is outside the shape boundary, zero shall be used to form the prediction syntax.

7.10.3.1.2 Root layer

At the root layer (the lowest 3 AC bands), the shape information is examined for every node to determine whether a node is an out_node.

If it is an out_node,

- no bits are decoded for this node;
- the four children nodes of this node are marked "to_be_decoded" (TBD);

otherwise,

- a zerotree symbol is decoded for this node using an adaptive arithmetic decoder.

If the decoded symbol for the node is either isolated_zero (IZ) or value (VAL),

- the four children nodes of this node are marked TBD;

otherwise,

- the symbol is either zerotree_root (ZTR) or valued_zerotree_root (VZTR) and the four children nodes of this node are marked "no_code" (NC).

If the symbol is VAL or VZTR,

- a non-zero wavelet coefficient is decoded for this node by root model;

otherwise,

- the symbol is either IZ or ZTR and the wavelet coefficient is set to zero for this node.

7.10.3.1.3 Between root and leaf layer

At any layer between the root layer and the leaf layer, the shape information is examined for every node to determine whether a node is an out_node.

If it is an out_node,

- no bits are decoded for this node;
- the four children nodes of this node are marked as either TBD or NC depending on whether this node itself is marked TBD or NC respectively;

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA Y. ADAMANE
ANSI Store order #X37445; Downloaded 7/18/2007 3:58:21 PM ET
Single user license only. Copying and networking prohibited.

otherwise, if it is marked NC,

- no bits are decoded for this node;
- the waveletcoefficient is set to zero for this node;
- the four children nodes are marked NC;

otherwise,

- a zerotree symbol is decoded for this node using an adaptive arithmetic decoder.

If the decoded symbol for the node is either isolated_zero (IZ) or value (VAL),

- the four children nodes of this node are marked TBD;

otherwise,

- the symbol is either zerotree_root (ZTR) or valued_zerotree_root (VZTR) and the four nodes of this node are marked "no_code" (NC).

If the symbol is VAL or VZTR,

- a non-zero wavelet coefficient is decoded for this node by valnz model;

otherwise,

- the symbol is either IZ or ZTR and the wavelet coefficient is set to zero for this node.

7.10.3.1.4 Leaf layer

At the leaf layer, the shape information is examined for every node to determine whether a node is an out_node.

If it is an out_node,

- no bits are decoded for this node;

otherwise, if it is marked NC,

- no bits are decoded for this node;
- the wavelet coefficient is set to zero for this node;

otherwise,

- * a wavelet coefficient is decoded for this node by valz adaptive arithmetic model;

7.10.3.2 Shape decomposition

The shape information for both shape adaptive zerotree decoding and the inverse shape adaptive wavelet transform is obtained by decomposing the reconstructed shape from the shape decoder. Assuming binary shape with 0 or 1 indicating a pixel being outside or inside the arbitrarily shaped object, the shape decomposition procedure can be described as follows:

1. For each horizontal line, collect all even-indexed shape pixels together as the shape information for the horizontal low-pass band and collect all odd-indexed shape pixels together as the shape information for the horizontal high-pass band, except for the special case where the number of consecutive 1's is one.
2. For an isolated 1 in a horizontal line, whether at an even-indexed location or at an odd-indexed location, it is always put together with the shape pixels for the low-pass band and a 0 is put at the corresponding position together with the shape pixels for the high-pass band.
3. Perform the above operations for each vertical line after finishing all horizontal lines.

4. Use the above operations to decompose the shape pixels for the horizontal and vertical low-pass band further until the number of decomposition levels is reached.

7.10.4 Still Texture Error Resilience

In error resilience mode, start code will always be used. The decoder is expecting packetized bitstream. The first packet is the image header information, then the shape coding information (for shape coding combined with error resilience, each scalable shape layer will be one packet starting with the texture_shape_layer_start_code), followed by the packets for DC and AC bands. Each packet start with texture marker (texture_marker = '0000000000000001') or a startcode (if the packet is the first in the layer), followed by the number of the first and last texture units encoded in the texture packet, named TU_first and TU_last, and by one bit header_extension_code, which is optionally followed by header information.

The TU_first and TU_last provide absolute reference for the texture units in the packet. This parameter specifies the texture unit number within a texture object. These two parameters are decoded by the function get_param(). The labeling of the texture units starts from 0 in DC band and following the same decoding order as in the non-error resilient case. In AC band, the TU number starts from 50 (> maximum number of bitplanes in DC band).

The header_extention_code bit has value 1 to indicate that additional header information is sent in the packet, zero otherwise, depending on error resilience needs. The information following the header_extention_code repeats the header information contained in the TextureObjectLayer().

Within each packet, segment_markers (a ZTR symbol coded by arithmetic coding) are expected when large segment is encountered. A segment is formed at the end of the currently decoded *sub_unit* (a part of a texture unit, to be defined in the following) whenever the total number of bits from the end of the last segment or from the beginning of the packet exceeds a threshold target_segment_length. The segment marker is decoded using the function decode_segment_marker().

7.10.4.1 Decoder Constraints

In order to support error resilience, the following constrains are used in the TOL.

To prevent error propagation, DPCM prediction in DC part is disabled.

To assure that the symbol identifying the resynch marker is unique, we impose the following constraint to the arithmetic decoder (AD): when the AD decodes the 000000000000000 (15 zero bits) symbol while decoding the wavelet coefficients, it expect the above bits to be followed by one '1' bit.

The AD has to reset all the models used in the adaptation process when starting a new texture packet. This assures that there are no dependencies across the texture packets. The models will be reset to uniform distribution.

7.10.4.2 Texture packet structure

When the TOL is operating in error resilient mode, the basic unit of a texture packet is the stream generated by encoding one texture unit. In the TOL error resilient case we define four types of texture units. Each texture unit lies entirely within one scalability layer:

7.10.4.2.1 DC band

In DC band, the texture unit is defined as a bitplane of all DC coefficients, starting from the MSB. The sub-unit is defined as one row of pixels in a texture unit.

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA Y. ADAPANE
Single user license only. Copying and networking prohibited.

7.10.4.2.2 Tree-depth scanning

In the tree-depth scanning mode the AC texture unit consists of the stream generated when encoding one tree structure, as specified in Figure V2 -26. Black blocks represent possible resynch marker positions. The total number of texture block is a function of the image size and of the number of levels in the wavelet decomposition. The texture bocks are formed following the scanning used to encode the coefficient. This is done to avoid degradation in coding efficiency performance.

The sub_unit are defined as follows:

All pixels in levels 0 and 1 within a tree.

The $2^l \times 2^l$ block within a tree for levels $l = 2, 3$.

The 16x16 block within a tree for levels $l > 3$.

An example is shown in Figure V2 -27.

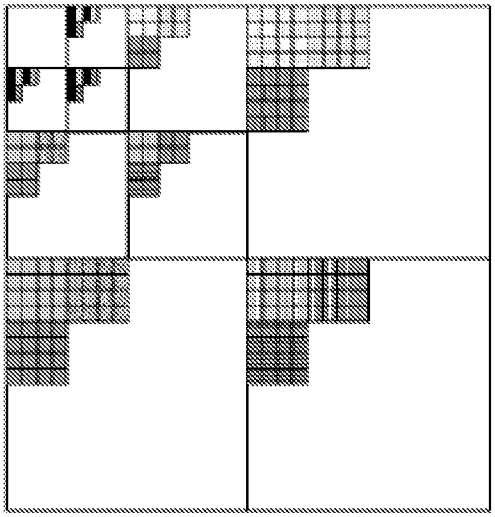


Figure V2 -26 -- Texture units for the single_quant mode.
All coefficients with the same shade belong into the same texture unit.

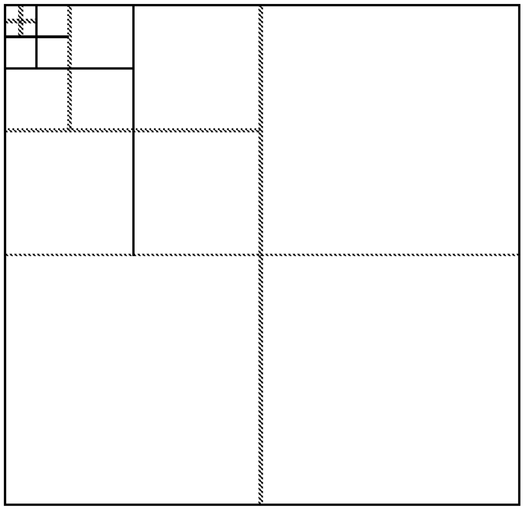


Figure V2 -27 – Sub-unit formation in a wavelet decomposed image. Each type of shade represents a sub-unit.

7.10.4.2.3 Subband-by-subband scanning

In the subband-by-subband scanning mode, the AC texture unit consists of the bit stream generated when encoding a slice of subband coefficients from all three subbands in the same wavelet decomposition layer, as specified in Figure V2 -26. The subbands are scanned using the same scanning order used to encode the coefficients.

The sub_unit is defined in as follows:

1) All pixels at level 0 within a TU.

2) The $2^l \times 2^l$ blocks within a TU for levels l = 1, 2, 3.

3) The 16x16 blocks within a TU for levels l >3.

Please note that this approach does not break the scalability, as the order of the bits generated by the encoder is preserved. Also, note that in the subband-by-subband scanning mode there is a strong dependency across subbands. The dependency consists of the having a part of the zero-tree significance map coded together with the subband coefficients. Because of this, losing a part of the subband means to lose part of the coefficients in the lower subbands.

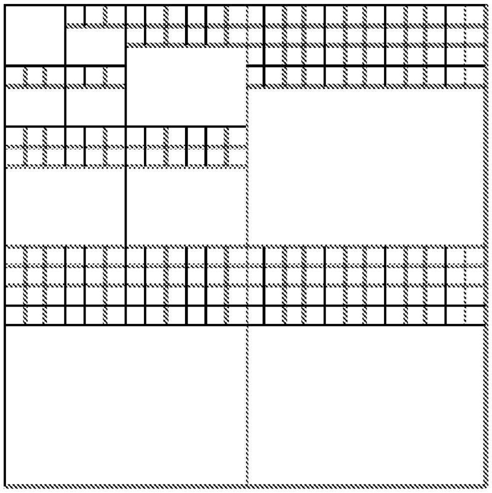


Figure V2 -28 -- Texture units for the band_by_band mode.
All coefficients with the same shade belong to the same texture unit.

7.10.5 Wavelet Tiling

The block diagram of the decoder supporting the Wavelet Tiling is shown in Figure V2 -29.

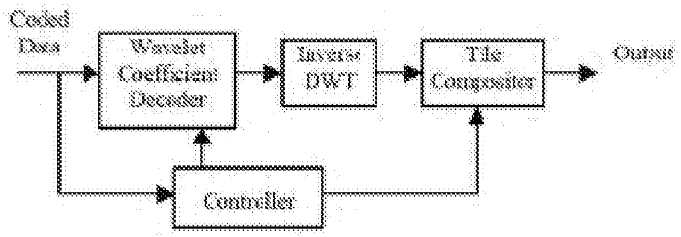


Figure V2 -29 -- Block diagram of the decoder with tiling opton

The module 'Controller' extracts the control informaton from coded data. It signals the 'Wavelet coefficient Decoder' module to decode a target tile based on the information. The target tile is decided by a system layer request. 'Controller' also signals 'Tile Compositor' module in order to specify a position of a decoded tile described by 'tile_id'.

The module of 'Wavelet Coefficient Decoder' is basically equivalent to the al modules without Inverse DWT appeared in Figure 7-35. 'Inverse DWT' is equivalent to the modules appeared in figure 7-35. 'Tile Compositor' composes tile decoded tiles based on the signal from 'Controller' and output single Still Texture Object.

7.10.5.1 Sructure of Tile

The structure of Still Texture Object with wavelet tiling is shown in Figure V2 - 30. This picture describes the case when texture_object_layer_shape=='00'. When texture_object_layer_shape=='01', object_width and object_height are used instead of texture_object_layer_width and textre_object_layer_height.

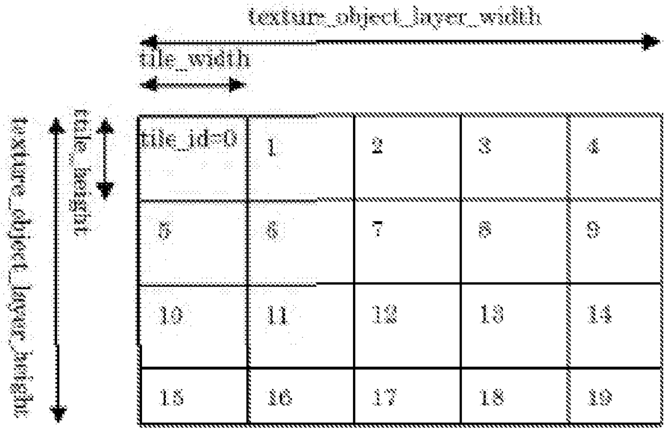


Figure V2 - 30 – Structure of tile

The relative position of each tile is identified by tile_id.

The number of tiles in row is computed by CEIL(texture_object_layer_width÷tile_width) and the number of tiles in column is computed by CEIL(texture_object_layer_height÷tile_height) when texture_object_layer_shape=='00'. Otherwise, object_width and object_height are used instead of texture_object_layer_width and texture_object_layer_height. Where CEIL() rounds up to the nearest integer.

7.10.5.2 The tile size of last row and last column

With regard to the tile size of last row and las column, texture_object_layer_width % tile_width and texture_object_layer_height % tile_height are used when texture_object_layer_shape=='00' if texture_object_layer_width and texture_object_layer_height are not a multiple of tile_width and tile_height. When texture_object_layer_shape=='01', object_width and object_height are used instead of texture_object_layer_width and texture_object_layer_height.

7.10.5.3 Searching a start position of the tile

The starting position of the target tile in the bitstream is found by using a look-up table before starting to decode. This way is applicable in tiling_jump_table_enable=1. In case of tiling_jump_table_enable = 0, texture_tile_start_code is used to find the start point of bitstream for each tile. For the look-up table generation, a size of each tile is stored as 'tile_size_low' and 'tile_size_high' in a header field. The decoding start position; *position[i]* for the *i*-th tile can be calculated by

$$position[i] = \sum_{j=0}^{i-1} tile\_size[j]$$

where the bitstream size of *i*-th tile is expressed by *tile_size[j]*. This mechanism enables to decode the wavelet coefficients of the tile at any place in the entire image.

7.10.6 Scalable binary shape object decoding

At first binary_arithmetic_decode() is called for decoding the base layer shape. For decoding the shape object of each layer in the enhancement layer, enh_binary_arithmetic_decode() is performed in the macroblock decoding manner. For decoding each shape block, SI_bab_type is decoded in first. SI_bab_type denotes the BAB type of SI(Scan Interleaving). And then decoding for BAB pixels based on CAE is performed as the BAB type of SI.

7.10.6.1 Decoding of base layer

It is identical as the video non-scalable shape coding. Refer to the shape decoding for base layer in the subclause 7.5 Shape Coding.

7.10.6.2 Decoding of enhancement layer

7.10.6.2.1 Block size control

The BAB size is controlled before decoding shape object of each enhancement layer. Decision of BAB size depends on the object size of each scalable layer. The resolutions of objects are divided into 3 classes. If the current coding layer is the *L*-th layer in the bitstream, the BAB size in each object resolution is obtained as follows:

```
layer_width = object_width >> (wavelet_decomposition_levels – L – 1);

layer_height = object_height >> (wavelet_decomposition_levels – L – 1);

if( layer_width >= 1024 || layer_height >= 1024)      bab_size = 64;

else if( layer_width >= 512 || layer_height >= 512)   bab_size = 32;

else                                                  bab_size = 16;
```

7.10.6.2.2 Border formation

When decoding a BAB, pixels from neighbouring BABs can be used to make up the context. A 2 pixel wide border about the current BAB is used as depicted in Figure V2 -31. In the figure, the size of the current BAB can be changed as described in subclause 7.10.6.2.1.

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON [illegible]
ANSI Store order [illegible] Downloaded 7/18/2007 [illegible] PM ET
Single user license only. Copying and networking prohibited.

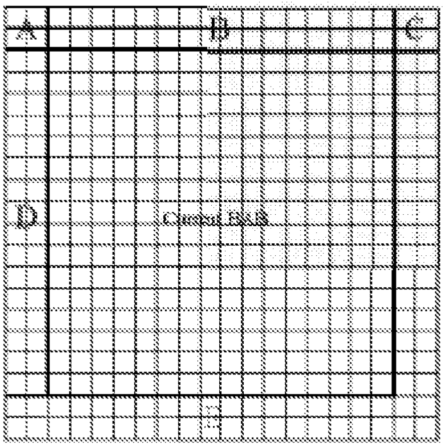


Figure V2 -31 -- Bordered BAB. A: TOP_LEFT_BORDER. B: TOP_BORDER. C: TOP_RIGHT_BORDER. D: LEFT_BORDER. E: BOTTOM_OR_RIGHT_BORDER

The border of the current BAB is partitioned into 5:

- TOP_LEFT_BORDER, which contains pixels from the BAB located to the upper-left of the current BAB and which consists of 2 lines of 2 pixels
- TOP_BORDER, which contains pixels from the BAB located above the current BAB and which consists of 2 lines of the number of bab_size pixels
- TOP_RIGHT_BORDER, which contains pixels from the BAB located to the upper-right of the current BAB and which consists of 2 lines of 2 pixels
- LEFT_BORDER, which contains pixels from the BAB located to the left of the current BAB and which consists of 2 columns of the number of bab_size pixels
- BOTTOM_OR_RIGHT_BORDER, which contains pixels from the collocated pixels in the lower layer and which consists of 2 lines in bottom and right of the current BAB. The pixel in (i,j) of the lower layer is repeatedly used for this border pixels in (2i,2j), (2i+1,2j), (2i,2j+1), and (2i+1,2j+1) of the current layer.

7.10.6.2.3 Decoding of BAB type of Scan Interleaving

For decoding a BAB of enhancement layer shape object, BAB type of SI is determined in advance. SI_bab_type denotes the two fields of BAB type of SI that are transitional BAB and exceptional BAB. Transitional BAB and exceptional BAB of SI are defined as follow.

transitional BAB: When the input BAB has no ESD and there exists no swapping pixel from wavelet *decomposition, the input BAB is determined as transitional BAB. For decoding transitional BAB, only the TSD data* of the BAB are decoded by BAC. All_0 and all_255 BAB are also included in this SI_bab_type. Since all_0 or all_255 BAB has no ESD or TSD, the BAB can be reconstructed by decoding only SI_bab_type.

exceptional BAB: When the input BAB has ESD or When there exist some swapping pixels from wavelet decomposition, the input BAB is determined as exceptional BAB. For decoding exceptional BAB, all of the pixels in the BAB are decoded by BAC.

The first decoded symbol of enh_binary_arithmetic_decode() is SI_bab_type. SI_bab_type is set to '0' for transitonal BAB and is set to '1' for exceptional BAB. For decoding this flag by BAC, SI_bab_type_prob[] is used for the probability table of odd symmetry filter case, and sto_SI_bab_type_prob_even[] is used for the probability table of even symmetry filter case.

7.10.6.2.4 Scan order decision

If SI_bab_type is '0' (transitional BAB) the scan order should be determined before perorming BAC. The decision method is similar to the method in the subclause 7.5.4.11 except that it uses right and bottom neighboring pixels instead of left and upper neighboring pixels. The bottom and right borders of the colocated block of the lower layer are extended with its neighboring pixels like Figure V2 - 32. If an extended border is outside VOP, the pixel values of the extended border are set to zero as in the subclause 7.5.4.11.

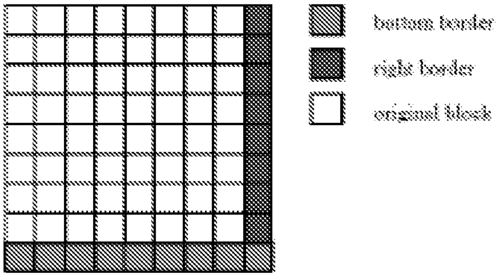


Figure V2 - 32 – Extension of border pixels

7.10.6.2.5 Decoding of transitional BAB

The decoding of transitonal BAB is almost same with the scalable shape decoding of transitional BAB for P-VOP in the subclause 7.5.4 except that the decoding order of pixels inside the transitional BAB for still texture objects is different from he method in the subclause 7.5.4.8.

An example of decoding order of transitional BAB with size 16 x 16 (bab_size==16) is shown in Figure V.2. In the figure, the "R"s indicate the pixels copied from collocated pixels in the reference layer. Apart from that, the actual decoding is based on the scan interleaving (SI) method as in the subclause 7.5.4.10. The binary arithmetic decoding and the usage of context values for arithmetic decoding are the same as thoseof the subclauses 7.5.4.13 and 7.5.4.14, respectively.

© ISO/IEC 2001 – All rights reserved
Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/[illegible]
Single user licence only. Copying and networking prohibited.

R	0	R	8	R	16	R	24	R	32	R	40	R	48	R	56
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
R	1	R	9	R	17	R	25	R	33	R	41	R	49	R	57
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
R	2	R	10	R	18	R	26	R	34	R	42	R	50	R	58
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
R	3	R	11	R	19	R	27	R	35	R	43	R	51	R	59
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
R	4	R	12	R	20	R	28	R	36	R	44	R	52	R	60
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
R	5	R	13	R	21	R	29	R	37	R	45	R	53	R	61
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
R	6	R	14	R	22	R	30	R	38	R	46	R	54	R	62
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
R	7	R	15	R	23	R	31	R	39	R	47	R	55	R	63
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191

Figure V2 - 33 -- The decoding order of transitional BAB (bab_size==16)

7.10.6.2.6 Decoding of exceptional BAB

The exceptional BAB coding includes two steps. First the collocated $BAB_{1/2}$ (vertically enhanced exceptional BAB with size bab_size/2 x bab_size) in the half-higher resolution layer is coded. Then the exceptional BAB in the current layer is coded based on the $BAB_{1/2}$ at half-higher resolution layer. Figure V2 - 34 illustrates the decoding steps for exceptional BABs.

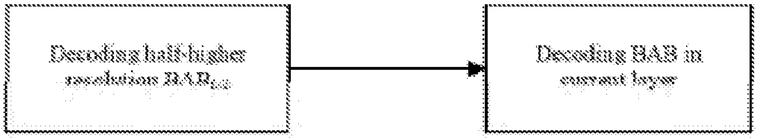

Decoding half-higher resolution $BAB_{1/2}$
Decoding BAB in current layer

Figure V2 - 34 -- Decoding steps for exceptional BABs

When decoding a vertically enhanced BAB, pixels from neighboring BABs in the half resolution layer can be used to make up the context. A 1 pixel wide border about the current $BAB_{1/2}$ is used as depicted in Figure V2 - 35.

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/ROOJA Y. ADAMANE
ANSI Store order X07445. Downloaded 7/18/2007 3:56:21 PM ET
Single user license only. Copying and networking prohibited.

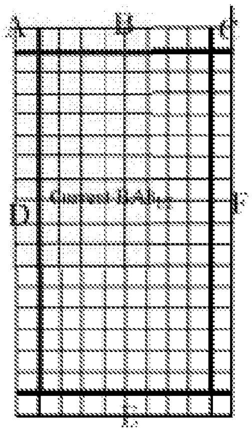


Figure V2 - 35 -- Bordered half-higher resolution BAB$_{1/2}$
A: TOP_LEFT_BORDER, B: TOP_BORDER, C: TOP_RIGHT_BORDER,
D: LEFT_BORDER, E: BOTTOM, F: RIGHT_BORDER

The border of the current BAB$_{1/2}$ is partitioned into 6 regions:

1. TOP_LEFT_BORDER, which contains pixels from the BAB$_{1/2}$ located to the upper-left of the current BAB$_{1/2}$ and which consists 1 pixel
2. TOP_BORDER, which contains pixels from the BAB$_{1/2}$ located above the current BAB$_{1/2}$ and which consists of 1 lines of the number of bab_size/2 pixels
3. TOP_RIGHT_BORDER, which contains pixels from the BAB$_{1/2}$ located to the upper-right of the current BAB$_{1/2}$ and which consists of 1 pixels
4. LEFT_BORDER, which contains pixels from the BAB$_{1/2}$ located to the left of the current BAB$_{1/2}$ and which consists of 1 columns of the number of bab_size pixels
5. BOTTOM_BORDER, which contains pixels from the colocated pixels in the lower layer and which consists of 1 lines in bottom of (bab_size/2+2) pixels. The pixels at (2i,j) in this border are filled by of the pixels at (i, j) in lower layer.
6. RIGHT_BORDER, which contains pixels from the collocated pixels in the lower layer and which consists of 1 column of the number of bab_size pixels. The pixel at (2i,j) in this border is filled by pixel at (i,j) from lower layer. And the pixel at (2i+1, j) in this border is filled by (i,j) from lower layer if the pixel at (i,j-1) in lower layer is not zero, otherwise it is filled by the logic OR of pixel at (i,j) and pixel at (i+1,j) in the lower layer.

By using 8-bits contexts, a pixel is arithmetic decoded as in CAE. The contexts for exceptional BAB decoding of shape object are described in Figure V2 -36(a), (b) and Figure V2 -37(a), (b). The contexts are calculated as below. And for each context, the probability table, sto_enh_odd_prob0[], sto_enh_even_prob0[], sto_enh_odd_prob1[], and sto_enh_even_prob1[] are used, respectively.

Also note that all the probability tables in the Appendix give the probability of zeros under given contexts. The probability tables are normalized to within range [0, 65536]. There are three special cases.

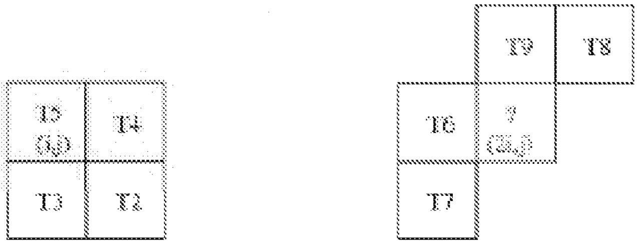
(1) Probability of zero under certain context is zero. This means that under that context it is certain that the coded value and can only be one.

(2) Probability of zero under certain context is 65536. This means that under that context, it is certain that the coded value can only be zero.

(3) Probability of zero under certain context is 65537. This means that this specific context can never occur.

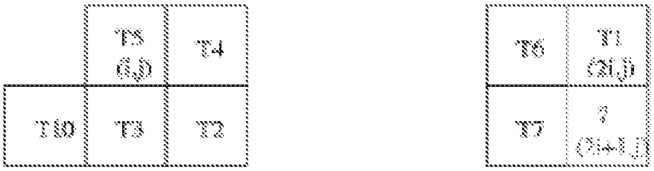
$\text{Context}_a = (T_9<<7)|(T_8<<6)|(T_7<<5)|(T_6<<4)|(T_5<<3)|(T_4<<2)|(T_3<<1)|T_2$

$\text{Context}_b = (T_1<<7)|(T_{10}<<6)|(T_7<<5)|(T_6<<4)|(T_5<<3)|(T_4<<2)|(T_3<<1)|T_2$



lower spatial layer　　half higher spatial layer

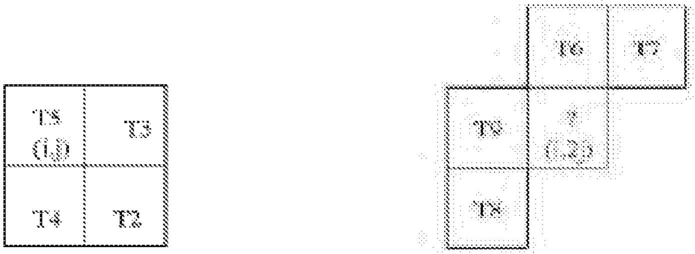
(a)



lower spatial layer　　half higher spatial layer

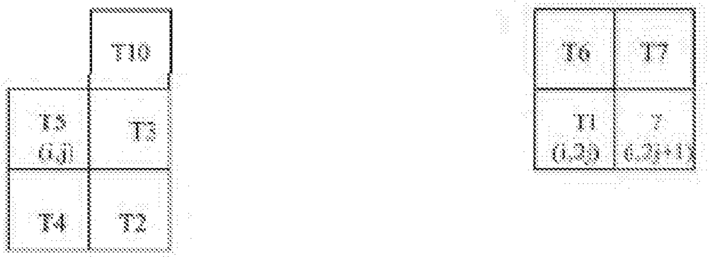
(b)

Figure V2 -36 -- The context used for vertically enhanced exceptional BABs



half higher spatial layer　　current spatial layer

(a)



half higher spatial layer　　current spatial layer

(b)

Figure V2 -37 -- The context used for enhanced exceptional BABs

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/RODOLFA Y. ADAMANE
ANSI Store order #X174457 Downloaded 7/18/2007 3:08:21 PM ET
Single user license only. Copying and networking prohibited.

7.11 Mesh object decoding

An overview of the decoding process is show in Figure 7-41.

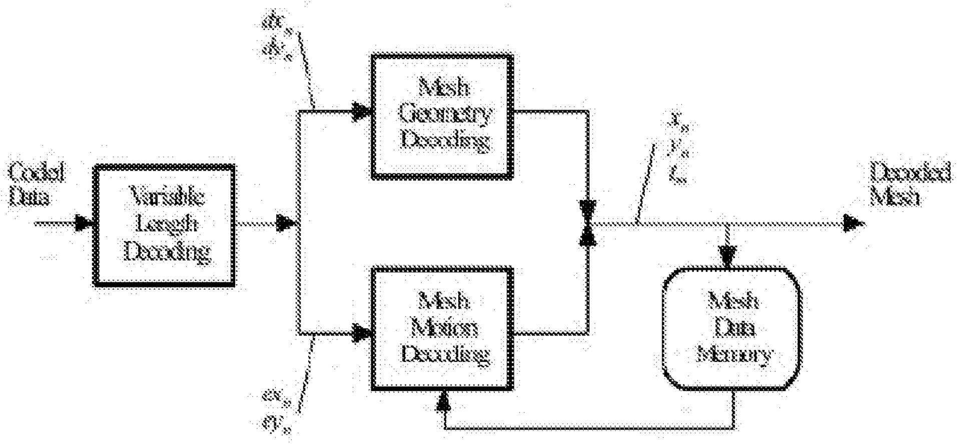


Figure 7-41 – Simplified 2D Mesh Object Decoding Process

Variable length decoding takes the coded data and decodes either node point location data or node point motion data. Node point location data is denoted by dx_n, dy_n and node point motion data is denoted by ex_n, ey_n, where n is the node point index (n = 0, ..., N-1). Next, either mesh geometry decoding or mesh motion decoding is applied. Mesh geometry decoding computes the node point locations from the location data and reconstructs a triangular mesh from the node point locations. Mesh motion decoding computes the node point motion vectors from the motion data and applies these motion vectors to the node points of the previous mesh to reconstruct the current mesh.

The reconstructed mesh is stored in the mesh data memory, so that it may be used by the motion decoding process for the next mesh. Mesh data consists of node point locations (x_n, y_n) and triangles t_m, where m is the triangle index (m = 0, ..., M-1) and each triangle t_m contains a triplet <i, j, k> which stores the indices of the node points that form the three vertices of that triangle.

A mesh object consists of a sequence of mesh object planes. The is_intra flag of the mesh object plane class determines whether the data that follows specifies the initial geometry of a new dynamic mesh, or that it specifies the motion of the previous mesh to the current mesh, in a sequence of meshes. Firstly, the decoding of mesh geometry is described; then, the decoding of mesh motion is described. In this part of ISO/IEC 14496, a pixel-based coordinate system is assumed, where the x-axis points to the right from the origin, and the y-axis points down from the origin.

7.11.1 Mesh geometry decoding

Since the initial 2D triangular mesh is either a uniform mesh or a Delaunay mesh, the mesh triangular structure (i.e. the connections between node points) is not coded explicitly. Only a few parameters are coded for the uniform mesh; only the 2D node point coordinates $\vec{p}_n = (x_n, y_n)$ are coded for the Delaunay mesh. In each case, the coded information defines the triangular structure of the mesh implicitly, such that it can be computed uniquely by the decoder. The mesh_type_code specifies whether the initial mesh is uniform or Delaunay.

7.11.1.1 Uniform mesh

A 2D uniform mesh subdivides a rectangular object plane area into a set of rectangles, where each rectangle in turn is subdivided into two triangles. Adjacent triangles share node points. The node points are spaced equidistant horizontally as well as vertically. An example of a uniform mesh is given in Figure 7-42.

Five parameters are used to specify a uniform mesh. The first two parametes, nr_mesh_nodes_hor and nr_mesh_nodes_vert, specify the number of node points of the mesh in the horizontal, resp. vertical direction. In the example of Figure 7-42, nr_mesh_nodes_hor is equal to 5 and nr_mesh_nodes_vert is equal to 4. The next two parameters, mesh_rect_size_hor and mesh_rect_size_vert, specify the horizontal, resp. vertical size of each rectangle in half pixel units. The meaning of these parameters is indicated in Figure 7-42. The last parameter, triangle_split_code, specifies how each rectangle is split to form two triangles. The four methods of splitting that are allowed are indicated in Figure 7-43. The top-left node point of a uniform mesh coincides with the origin of a local coordinate system.

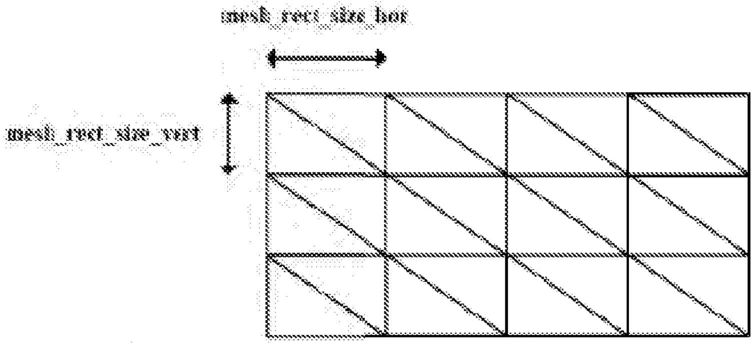


Figure 7-42 -- Specification of a uniform 2D mesh

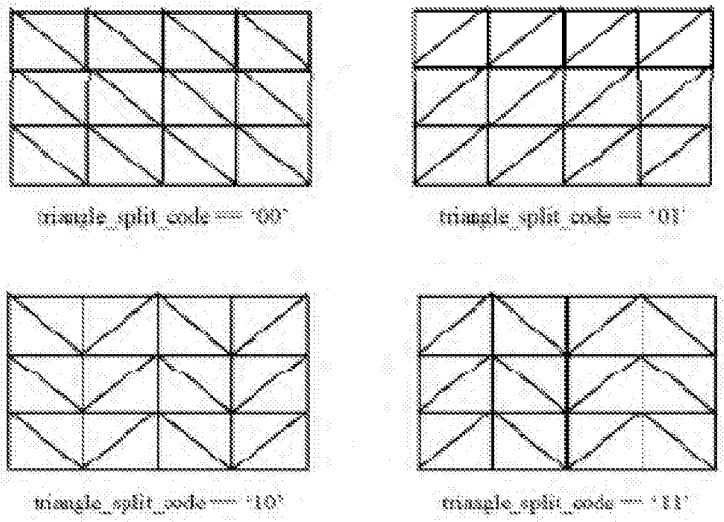

triangle_split_code == '00'
triangle_split_code == '01'
triangle_split_code == '10'
triangle_split_code == '11'

Figure 7-43 -- Illustration of the types of uniform meshes defined

7.11.1.2 Delaunay mesh

First, the total number of node points in the mesh N is decoded; then, the number of node points that are on the boundary of the mesh N_b is decoded. Note that N is the sum of the number of nodes in the interior of the mesh, N_i and the number of nodes on the boundary, N_b,

$$N = N_i + N_b .$$

Now, the locations of boundary and interior node points are decoded, where we assume the origin of the local coordinate system is at the top left of the bounding rectangle surrounding the initial mesh. The x-, resp. y-coordinate of the first node point, $\vec{p}_0 = (x_0, y_0)$, is decoded directly, where x_0 and y_0 are specified w.r.t. to the origin of the local coordinate system. All the other node point coordinates are computed by adding a dx_n, resp. dy_n value to, resp. the x- and y-coordinate of the previously decoded node point. Thus, the coordinates of the initial node point $\vec{p}_0 = (x_0, y_0)$ is decoded as is, whereas the coordinates of all other node points, $\vec{p}_n = (x_n, y_n)$, $n = 1, \ldots, N - 1$, are obtained by adding a decoded value to the previously decoded node point coordinates:

$$x_n = x_{n-1} + dx_n \quad \text{and} \quad y_n = y_{n-1} + dy_n .$$

The ordering in the sequence of decoded locations is such that the first N_b locations correspond to boundary nodes. Thus, after receiving the first N_b locations, the decoder is able to reconstruct the boundary of the mesh by connecting each pair of successive boundary nodes, as well as the first and the last, by straight-line edge segments. The next $N - N_b$ values in the sequence of decoded locations correspond to interior node points. Thus, after receiving N nodes, the locations of both the boundary and interior nodes can be reconstructed, in addition to the polygonal shape of the boundary. This is illustrated with an example in Figure 7-44.

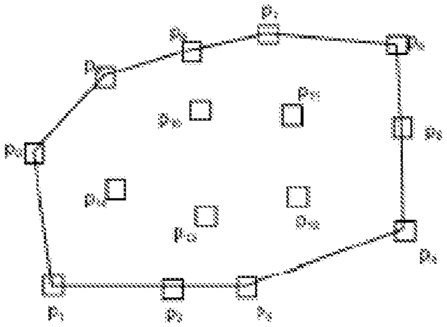


Figure 7-44 -- Decoded node points and mesh boundary edge

The mesh is finally obtained by applying constrained Delaunay triangulation to the set of decoded node points, where the polygonal mesh boundary is used as a constraint. A constrained triangulation of a set of node points $\vec{p}_n$ contains the line segments between successive node points on the boundary as edges and contains triangles only in the interior of the region defined by the boundary. Each triangle $t_k = (\vec{p}_l, \vec{p}_m, \vec{p}_n)$ of a constrained Delaunay triangulation furthermore satisfies the property that the circumcircle of t_k does not contain in its interior any node point $\vec{p}_r$ visible from all three vertices of t_k. A node point is visible from another node point if a straight line drawn between them falls entirely inside or exactly on the constraining polygonal boundary. The Delaunay triangulation process is defined as any algorithm that is equivalent to the following.

a. Determine any triangulation of the given node points such that all triangles are contained in the interior of the polygonal boundary. The triangulation shall contain $2N + N_b - 2$ triangles.

b. Inspect each interior edge, shared by two opposite triangles, of the triangulation and test if the edge is locally Delaunay. If there is an interior edge that is not localy Delaunay, the two opposite triangles $<p_a, p_b, p_c>$ and $<p_a, p_c, p_d>$ sharing this edge are replaced by triangles $<p_a, p_b, p_d>$ and $<p_b, p_c, p_d>$. Continue until all interior edges of the triangulation are locally Delaunay.

An interior edge, shared by two opposite triangles $<p_a, p_b, p_c>$ and $<p_a, p_c, p_d>$, is locally Delaunay if point p_d is outside the circumcircle of triangle $<p_a, p_b, p_c>$. If point p_d is inside the circumcircle of triangle $<p_a, p_b, p_c>$, then the edge is not locally Delaunay. If point p_d is exactly on the circumcircle of triangle $<p_a, p_b, p_c>$, then the edge between points p_a and p_c is deemed locally Delaunay only if point p_b or point p_d is the point (among these four points) with the maximum x-coordinate, or, in case there is more than one point with the same maximum x-coordinate, the point with the maximum y-coordinate among these points. An example of a mesh obtained by constrained triangulation of the node points of Figure 7-44 is shown in Figure 7-45.

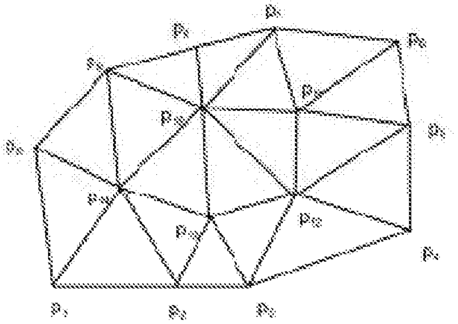


Figure 7-45 -- Decoded triangular mesh obtained by constrained Delaunay triangulation

7.11.2 Decoding of mesh motion vectors

Each node point $\vec{p}_n$ of a 2D Mesh Object Plane numbered k in the sequence of Mesh Object Planes has a 2D motion vector $\vec{v}_n = (vx_n, vy_n)$, defined from Mesh Object Plane k to k+1. By decoding these motion vectors, one is able to reconstruct the locations of node points in Mesh Object Plane numbered k+1. The triangular topology of the mesh remains the same throughout the sequence. Node point motion vectors are decoded according to a predictive method, i.e., the components of each motion vector are predicted using the components of already decoded motion vectors of other node points.

7.11.2.1 Motion vector prediction

To decode the motion vector of a node point $\vec{p}_n$ that is part of a triangle $t_k = (\vec{p}_l, \vec{p}_m, \vec{p}_n)$, where the two motion vectors vectors $\vec{v}_l$ and $\vec{v}_m$ of the nodes $\vec{p}_l$ and $\vec{p}_m$ have already been decoded, one can use the values of $\vec{v}_l$ and $\vec{v}_m$ to predict $\vec{v}_n$ and add the prediction vector to a decoded prediction error vector. Starting from an initial triangle t_k of which all three node motion vectors have been decoded, there must be at least one other, neighboring, triangle t_w that has two nodes in common with t_k. Since the motion vectors of the two nodes that t_k and t_w have in common have already been decoded, one can use these two motion vectors to predict the motion vector of the third node in t_w. The actual prediction vector $\vec{w}_n$ is computed by averaging of the two prediction motion vectors and the components of the prediction vector are rounded to half-pixel accuracy, as follows:

$$\vec{w}_n = 0.5 \bullet (\mathrm{floor}(vx_m + vx_l + 0.5), \mathrm{floor}(vy_m + vy_l + 0.5)),$$
$$\vec{v}_n = \vec{w}_n + \vec{e}_n$$

Here, $\vec{e}_n = (ex_n, ey_n)$ denotes the prediction error vector, the components of which are decoded from variable length codes. This procedure is repeated while traversing the triangles and nodes of the mesh, as explained below. While visiting all triangles of the mesh, the motion vector data of each node is decoded from the bitstream one by one. Note that no prediction is used to decode the first motion vector,

$$\vec{v}_{n_0} = \vec{e}_{n_0},$$

and that only the first decoded motion vector is used as a predictor to code the second motion vector,

$$\vec{v}_{n_1} = \vec{v}_{n_0} + \vec{e}_{n_1}.$$

Note further that the prediction error vector is specified only for node points with a nonzero motion vector. For all other node points, the motion vector is simply $\vec{v}_n = (0,0)$.

Finally, the horizontal and vertical components of mesh node motion vectors are processed to lie within a certain range, equivalent to the processing of video block motion vectors described in subclause 7.6.3.

7.11.2.2 Mesh traversal

We use a *breadth-first traversal* to order all the triangles and nodes in the mesh numbered *k*, and to decode the motion vectors defined from mesh *k* to *k*+1. The breadth-first traversal is determined uniquely by the topology and geometry of an intra-coded mesh. That is, the ordering of the triangles and nodes shall be computed on an intra-coded Mesh Object Plane and remains constant for the following predictive-coded Mesh Object Planes. The breadth-first traversal of the mesh triangles is defined as follows (see Figure 7-46 for an illustration).

First, define the *initial triangle* as follows. Define the top left mesh node as the node *n* with minimum $x_n + y_n$, assuming the origin of the local coordinate system is at the top left. If there is more than one node with the same value of $x_n + y_n$, then choose the node point among these with minimum *y*. The initial triangle is the triangle that contains the edge between the top-left node of the mesh and the next clockwise node on the boundary. Label the initial triangle with the number 0.

Next, all other triangles are iteratively labeled with numbers 1, 2, ..., *M* - 1, where *M* is the number of triangles in the mesh, as follows.

> Among all labeled triangles that have adjacent triangles which are not yet labeled, find the triangle with the lowest number label. This triangle is referred to in the following as the *current triangle*. Define the *base edge* of this triangle as the edge that connects this triangle to the already labeled neighboring triangle with the lowest number. In the case of the initial triangle, the base edge is defined as the edge between the top-left node and the next clockwise node on the boundary. Define the *right* edge of the current triangle as the next counterclockwise edge of the current triangle with respect to the base edge; and define the *left edge* as the next clockwise edge of the current triangle with respect to the base edge. That is, for a triangle $t_k = \left(\vec{p}_l, \vec{p}_m, \vec{p}_n\right)$, where the vertices are in clockwise order, if $\left(\vec{p}_l \vec{p}_m\right)$ is the base edge, then $\left(\vec{p}_l \vec{p}_n\right)$ is the right edge and $\left(\vec{p}_m \vec{p}_n\right)$ is the left edge.
>
> Now, check if there is an unlabeled triangle adjacent to the current triangle, sharing the right edge. If there is such a triangle, label it with the next available number. Then check if there is an unlabeled triangle adjacent to the current triangle, sharing the left edge. If there is such a triangle, label it with the next available number.

This process is continued iteratively until all triangles have been labeled with a unique number *m*.

The ordering of the triangles according to their assigned label numbers implicitly defines the order in which the motion vector data of each node point is decoded, as described in the following. Initially, motion vector data for the top-left node of the mesh is retrieved from the bitstream. No prediction is used for the motion vector of this node, hence this data specifies the motion vector itself. Then, motion vector data for the second node, which is the next clockwise node on the boundary w.r.t. the top-left node, is retrieved from the bitstream. This data contains the prediction error for the motion vector of this node, where the motion vector of the top-left node is used as a prediction. Mark these first two nodes (that form the base edge of the initial triangle) with the label 'done'.

Next, process each triangle as determined by the label numbers. For each triangle, the base edge is determined as defined above. The motion vectors of the two nodes of the base edge of a triangle are used to form a prediction for the motion vector of the third node of that triangle. If that third node is not yet labeled 'done', motion vector data is retrieved and used as prediction error values, i.e. the decoded values are added to the prediction to obtain the actual motion vector. Then, that third node is labeled 'done'. If the third note is already labeled 'done', then it is simply ignored and no data is retrieved. Note that due to the ordering of the triangles as defined above, the two nodes on the base edge of a triangle are guaranteed to be labeled 'done' when that triangle is processed, signifying that their motion vectors have already been decoded and may be used as predictors.

© ISO/IEC 2001 – All rights reserved
Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA V. AGAWANE
ANSI Store order X037445? Downloaded: 7/18/2007 3:58:21 PM ET
Single user license only. Copying and networking prohibited.

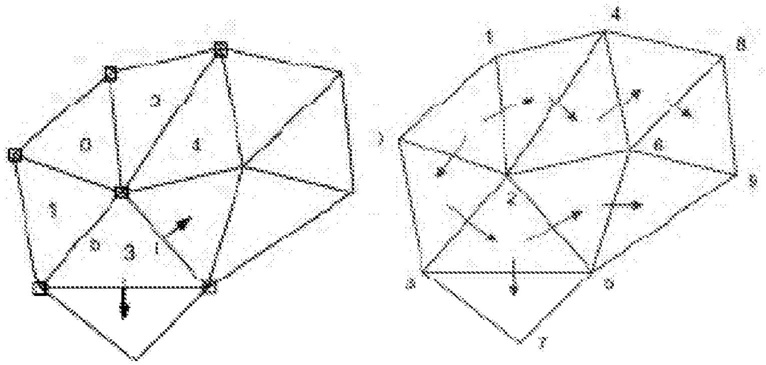


Figure 7-46 – Breadth-first traversal of a 2D triangular example mesh

In Figure 7-46 an example is shown of breadth-first traversal. On the left, the traversal is halfway through the mesh - five triangles have been labeled (with numbers) and the motion vectors of six node points have been decoded (marked with a box symbol). The triangle which has been labeled '3' is the 'current triangle'; the base edge is 'b'; the right and left edge are 'r' and 'l'. The triangles that will be labeled next are the triangles sharing the right, resp. left edge with the current triangle. After those triangles are labeled, the triangle which has been labeled '4' will be the next 'current triangle' and another motion vector will be decoded. On the right, the traversed 2D triangular mesh is shown, illustrating the transitions between triangles and final order of node points according to which respective motion vectors are decoded.

7.12 FBA object decoding

The frame rate, time code and skip frames informaton is independently associated with face, body or both depending on the fba_object_mask. For example, if the frame rate is set to 30Hz during a frame with fba_object_mask='01' (face) followed by a frame with frame rate set to 15Hz with fba_object_mask='10' (body) then the frame rate for the face remains set to 30Hz. If the fba_object_mask='11' the temporal information is applied to both face and body.

7.12.1 Frame based face object decoding

This subclause specifies the additional decoding process required for face object decoding.

The coded data is decoded by an arithmetic decoding process. The arithmetic decoding process is described in detail in annex B. Following the arithmetic decoding, the data is de-quantized by an inverse quantisation process. The FAPs are obtained by a predictive decoding scheme as shown in Figure 7-47.

The base quantisation step size QP for each FAP is listed in Table C-1. The quantisation parameter fap_quant is applied uniformly to all FAPs. The magnitude of the quantisation scaling factor ranges from 1 to 8. The value of fap_quant == 0 has a special meaning, it is used to indicate lossless coding mode, so no dequantisation is applied. The quantisation stepsize is obtained as follows:

```
if (fap_quant)
    qstep = QP * fap_quant
else
    qstep = 1
```

The dequanized FAP'(t) is obtained from the decoded coefficient FAP''(t) as follows:

FAP'(t) = qstep * FAP''(t)

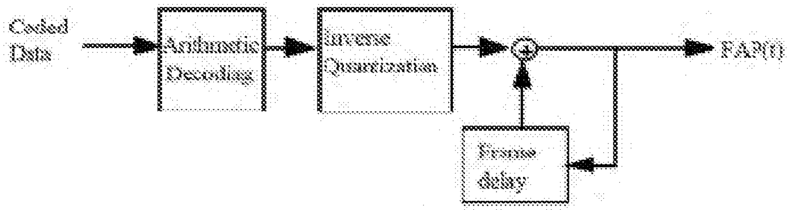


Figure 7-47 – FAP decoding

7.12.1.1 Decoding of faps

For a given frame FAPs in the decoder assume one of three of the following states:

set by a value transmitted by the encoder

retain a value previously sent by the encoder

interpolated by the decoder

FAP values which have been initialized in an intra coded FAP set are assumed to retain those values if subsequently masked out unless a special mask mode is used to indicate interpolation by the decoder. FAP values which have never been initialized must be estimated by the decoder. For example, if only FAP group 2 (inner lip) is used and FAP group 8 (outer lip) is never used, the outer lip points must be estimated by the decoder. In a second example the FAP decoder is also expected to enforce symmetry when only the left or right portion of a symmetric FAP set is received (e.g. if the left eye is moved and the right eye is subject to interpolation, it is to be moved in the same way as the left eye).

7.12.2 DCT based face object decoding

The bitstream is decoded into segments of FAPs, where each segment is composed of a temporal sequence of 16 FAP object planes. The block diagram of the decoder is shown in Figure 7-48.

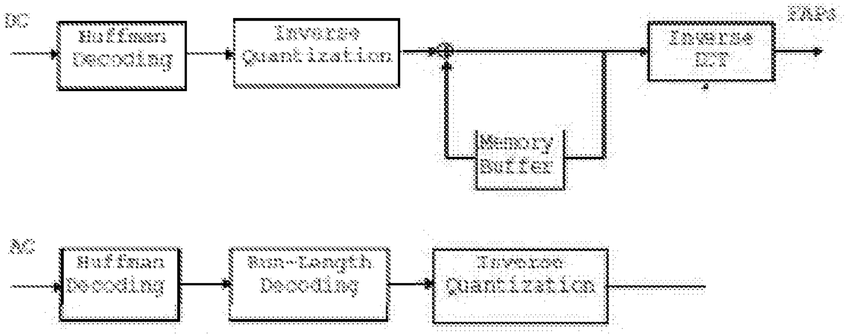


Figure 7-48 – Block diagram of the DCT-based decoding process

The DCT-based decoding process consists of the following three basic steps:

Differential decoding the DC coefficient of a segment.

Decoding the AC coefficients of the segment.

Determining the 16 FAP values of the segment using inverse discrete cosine transform (IDCT).

A uniform quantisation step size is used for all AC coefficients. The quantisation step size for AC coefficients is obtained as follows:

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

where DCTQP[i] is the base quantisation step size and its value is defined in subclause 6.3.10.10. The quantisation step size of the DC coefficient is one-third of the AC coefficients. Different quantisation step sizes are used for different FAPs.

The DCT-based decoding process is applied to all FAP segments except the viseme (FAP #1) and expression (FAP #2) parameters. The latter two parameters are differential decoded without transform. The decoding of viseme and expression segments are described at the end of this subclause.

For FAP #3 to FAP #68, the DC coefficient of an intra coded segment is stored as a 16-bit signed integer if its value is within the 16-bit range. Otherwise, it is stored as a 31-bit signed integer. For an inter coded segment, the DC coefficient of the previous segment is used as a prediction of the current DC coefficient. The prediction error is decoded using a Huffman table of 512 symbols. An "ESC" symbol, if obtained, indicates that the prediction error is out of the range [-255, 255]. In this case, the next 16 bits extracted from the bitstream are represented as a signed 16-bit integer for the prediction error. If the value of the integer is equal to -256*128, it means that the value of the prediction error is over the 16-bit range. Then the following 32 bits from the bitstream are extracted as a signed 32-bit integer, in twos complement format and the most significant bit first

The AC coefficients, for both inter and intra coded segments, are decoded using Huffman tables. The run-length code indicates the number of leading zeros before each non-zero AC coefficient. The run-length ranges from 0 to 14 and proceeds the code for the AC coefficient. The symbol 15 in the run length table indicates the end of non-zero symbols in a segment. Therefore, the Huffman table of the run-length codes contains 16 symbols. The values of non-zero AC coefficients are decoded in a way similar to the decoding of DC prediction errors but with a different Huffman table.

The bitstreams corresponding to viseme and expression segments are basically differential decoded without IDCT. For an intra coded segment, the quantized values of the first viseme_select1, viseme_select2, viseme_blend, expression_select1, expression_select2, expression_intensity1, and expression_intensity2 within the segment are decoded using fixed length code. These first values are used as the prediction for the second viseme_select1, viseme_select2, ... etc of the segment and the prediction error are differential decoded using Huffman tables. For an inter coded segment, the last viseme_select1, for example, of the previous decoded segment is used to predict the first viseme_select1 of the current segment. In general, the decoded values (before inverse quantisation) of differential coded viseme and expression parameter fields are obtained

```
byviseme_segment_select1q[k] = viseme_segment_select1q[k-1] +
    viseme_segment_select1q_diff[k] - 14
viseme_segment_select2q[k]        = viseme_segment_select2q[k-1] +
    viseme_segment_select2q_diff[k] - 14
viseme_segment_blendq[k]       = viseme_segment_blendq[k-1] +
    viseme_segment_blendq_diff[k] - 63
expression_segment_select1q[k]     = expression_segment_select1q[k-1] +
    expression_segment_select1q_diff[k] - 6
expression_segment_select2q[k]     = expression_segment_select2q[k-1] +
    expression_segment_select2q_diff[k] - 6
expression_segment_intensity1q[k] = expression_segment_intensity1q[k-1] +
    expression_segment_intensity1q_diff[k] - 63
expression_segment_intensity2q[k] = expression_segment_intensity2q[k-1] +
    expression_segment_intensity2q_diff[k] - 63
```

7.12.3 Decoding of the viseme parameter fap 1

Fourteen visemes have been defined for selection by the Viseme Parameter FAP 1, the definition is given in annex C. The viseme parameter allows two visemes from a standard set to be blended together. The viseme parameter is composed of a set of values as follows.

Table 7-17 -- Viseme parameter range

viseme () {	Range
viseme_select1	0-14
viseme_select2	0-14
viseme_blend	0-63
viseme_def	0-1
}	

Viseme_blend is quantized (step size = 1) and defines the blending of viseme1 and viseme2 in the decoder by the following symbolic expression where viseme1 and 2 are graphical interpretations of the given visemes as suggested in the non-normative annex.

$$\text{final viseme} = (\text{viseme 1}) * (\text{viseme\_blend} / 63) + (\text{viseme 2}) * (1 - \text{viseme\_blend} / 63)$$

The viseme can only have impact on FAPs that are currently allowed to be interpolated.

If the viseme_def bit is set, the current mouth FAPs can be used by the decoder to define the selected viseme in terms of a table of FAPs. This FAP table can be used when the same viseme is invoked again later for FAPs which must be interpolated.

7.12.4 Decoding of the viseme parameter fap 2

The expression parameter allows two expressions from a standard set to be blended together.The expression parameter is composed of a set of values as follows.

Table 7-18 – Expression parameter range

expression () {	Range
expression_select1	0-6
expression_intensity1	0-63
expression_select2	0-6
expression_intensity2	0-63
init_face	0-1
expression_def	0-1
}	

Expression_intensity1 and expression_intensity2 are quanized (step size = 1) and define excitation of expressions 1 and 2 in the decoder by the following equations where expressions 1 and 2 are graphical interpretations of the given expression as suggested by the non-normative reference.

$$\text{final expression} = \text{expression1} * (\text{expression\_intensity1} / 63) + \text{expression2} * (\text{expression\_intensity2} / 63)$$

The decoder displays the expressions according to the above formula as a superposition of the 2 expressions.

The expression can only have impact on FAPs that are currently allowed to be interpolated. If the init_face bit is set, the neutral face may be modified within the neutal face constraints of mouth closure, eye opening, gaze direction, and head orientation before FAPs 3-68 are applied. If the expression_def bit is set, the current FAPs can be used to define the selected expression in terms of a table of FAPs. This FAP table can then be used when the same expression is invoked again later.

7.12.5 Fap masking

The face is animated by sending a stream of facial animation parameters. FAP masking, as indicated in the bitstream, is used to select FAPs. FAPs are selected by using a two level mask hierarchy. The first level contains two bit code for each group indicating the following options:

1. no FAPs are sent in the group
2. a mask is sent indicating which FAPs in the group are sent. FAPs not selected by the group mask retain their previous value if any previously set value (not interpolated by decoder if previously set)
3. a mask is sent indicating which FAPs in the group are sent. FAPs not selected by the group mask retain must be interpolated by the decoder
4. all FAPs in the group are sent.

7.12.6 Frame Based Body Decoding

This clause specifies the additional decoding process required for body model decoding.

The BAPs are quantized and coded by a predictive coding scheme, similar to the FAPs. For each parameter to be coded in the current frame, the decoded value of this parameter in the previous frame is used as the prediction. Then the prediction error, i.e., the difference between the current parameter and its prediction, is computed and coded by arithmetic coding. This predictive coding scheme prevents the coding error from accumulating. The arithmetic decoding process is described in detail in clause B.2.3.

Similar to FAPs, each BAP has a different precision requirement. Therefore different quantisation step sizes are applied to the BAPs. The base quantisation step size for each BAP is defined in the tables below. The bit rate is controlled by adjusting the quantisation step via the use of a quantisation index parameter called **bap_pred_quant_index** is applied uniformly to all BAPs. The magnitude of the quantisation index parameter ranges from 0 to 31 and is an index to a bap_pred_scale_table as described in clause 6.3.10.1 to obtain the BAP_QUANT parameter.

The base quantisation step size BQP for each BAP is listed in annex C. The quantisation parameter BAP_QUANT is applied uniformly to all BAPs. The magnitude of the quantisation scaling factor ranges from 1 to 31. The value of BAP_QUANT == 0 has a special meaning, it is used to indicate lossless coding mode, so no dequantisation is applied. The quantisation stepsize is obtained as follows:

```
BAP_QUANT = bap_pred_scale[bap_pred_quant_index]
If (BAP_QUANT)
    qstep = BQP * BAP_QUANT
Else
    qstep = 1
```

The dequanized BAP'(t) is obtained from the decoded coefficient BAP''(t) as follows:

BAP'(t) = qstep * BAP''(t)

Licensed to [illegible]
Single user license only. Copying and networking prohibited.

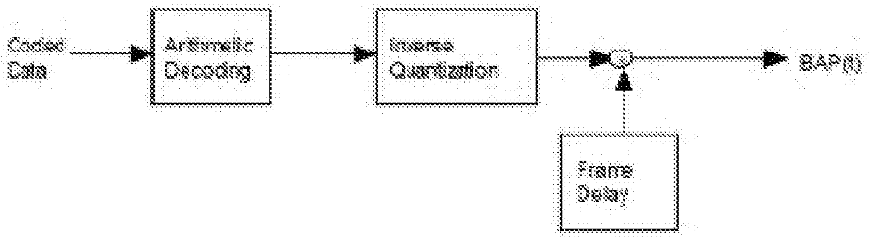


Figure V2 -38 – BAP predictive coding

7.12.6.1 Decoding of BAPs

For a given frame, BAPs in the decoder assume one of the three following states:

1. set by a value transmitted by the encoder

2. retain a value previously sent by the encoder

BAP values which have been initialized in an intra coded BAP set are assumed to retain those values if subsequently masked out.

7.12.7 DCT based body object decoding

The bitstream is decoded into segments of BAPs, where each segment is composed of a temporal sequence of 16 BAP object planes. The block diagram of the decoder is shown in Figure V2 -39.

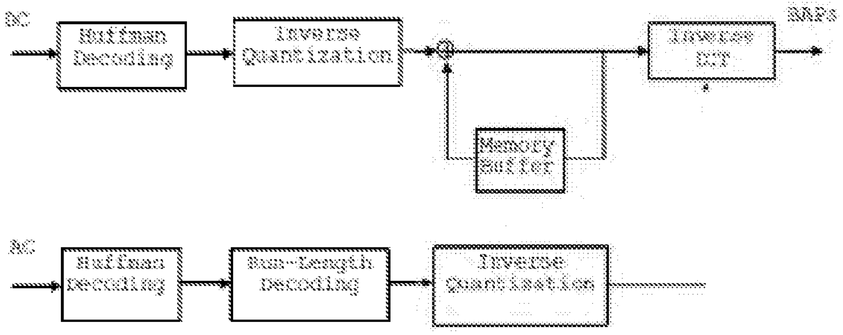


Figure V2 -39 – Block diagram of the DCT-based BAPs decoding process

The DCT-based decoding process consists of the following three basic steps:

1. Differential decoding the DC coefficient of a segment.
2. Decoding the AC coefficients of the segment.
3. Determining the 16 BAP values of the segment using inverse discrete cosine transform (IDCT).

A uniform quantisation step size is used for all AC coefficients. The quantisation step size for AC coefficients is obtained as follows:

qstep[i] = bap_scale[bap_quant_index] * BQP[i] where BQP[] is the base quantization step size and its value is defined in clause 6.3.10.8. The quantisation step size of the DC coefficient is one-third of the AC coefficients.

The DCT-based decoding process is applied to all BAPs. The DC coefficient of an intra coded segment is stored as a 16-bit signed integer if its value is within the 16-bit range. Otherwise, it is stored as a 31-bit signed integer. For an inter-coded segment, the DC coefficient of the previous segment is used as a prediction of the current DC coefficient. The prediction error is decoded using a Huffman table of 512 symbols. An "ESC" symbol, if obtained, indicates that the prediction error is out of the range [-255, 255]. In this case, the next 16 bits extracted from the bitstream are represented as a signed 16-bit integer for the prediction error. If the value of the integer is equal to -256*128, it means that the value of the predicton error is over the 16-bit range. Then the following 32 bits from the bitstream are extracted as a signed 32-bit integer, in twos complement format and the most significant bit first.

The AC coefficients, for both inter and intra coded segments, are decoded using Huffman tables. The run-length code indicates the number of leading zeros before each non-zero AC coefficient. The run-length ranges from 0 to 14 and proceeds the code for the AC coefficient. The symbol 15 in the run length table indicates the end of non-zero symbols in a segment. Therefore, the Huffman table of the run-length codes contains 16 symbols. The values of non-zero AC coefficients are decoded in a way similar to the decoding of DC prediction errors but with a different Huffman table.

7.13 3D Mesh Object Decoding

The Topological Surgery decoder is composed of four main modules, as shown in Figure V2 - 40, namely:

An arithmetic decoder, which reads a section of the input stream and outputs a bit stream.

A vertex graph decoder, which reads a section of the bit stream and outputs a bounding loop look-up table. However, when has_stitches flag is on, stitches are decoded and a vertex clustering array is built.

A triangle tree decoder, which reads a section of the bit stream and outputs the length of each run and the size of each sub-tree of the triangle tree.

A triangle data decoder, which reads a section of the input bit and outputs a stream of triangle data. This stream of triangle data contains the geometry and the properties associated with each triangle.

When a 3D mesh bitstream is received, the control information including stitches and partition types, as well as the header information including properties are obtained through the demultiplexer. The remaining bitstream is fed into the arithmetic decoder.

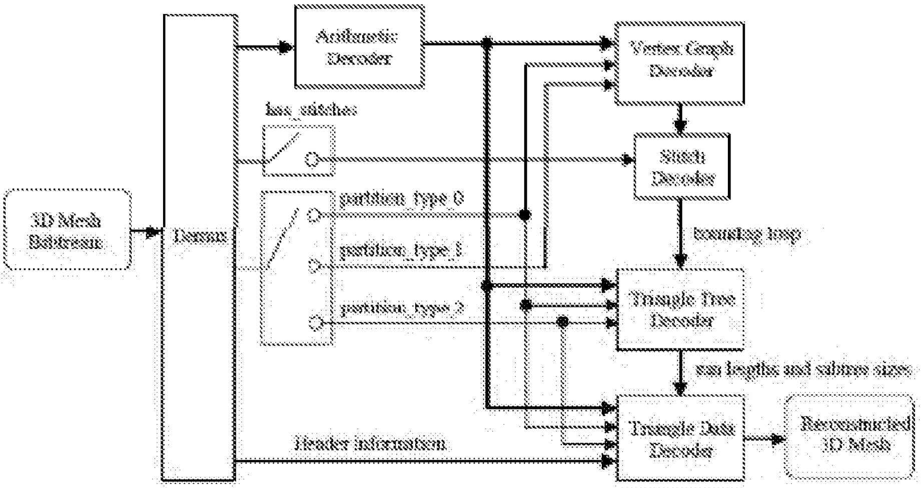


Figure V2 - 40 – Block diagram of the Topological Surgery decoder

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA [illegible]
Single user license only. Copying and networking prohibited.

In the hierarchical mode, the Topological Surgery decoder is used to decode the first (coarsest) level of detail mesh (base mesh), and the Forest Split decoder is used to recursively refine each level of detail to produce a new, higher resolution, level of detail. The Forest Split decoder is composed of four blocks, as shown in Figure V2 - 41, namely:

An arithmetic decoder, which reads a section of the input stream and outputs a bit stream.

A forest decoder, which reads a section of the bit stream and determines which edges of the current level mesh are part of the forest of edges.

A simple polygon decoder, which reads a section of the bit stream and outputs a sequence of simple polygons, including not only connectivity information, but also vertex coordinates, and property values.

A forest split operator, which takes the output data produced by the forest decoder and the simple polygon decoder, and applies the corresponding forest split operation, producing a new level of detail mesh.

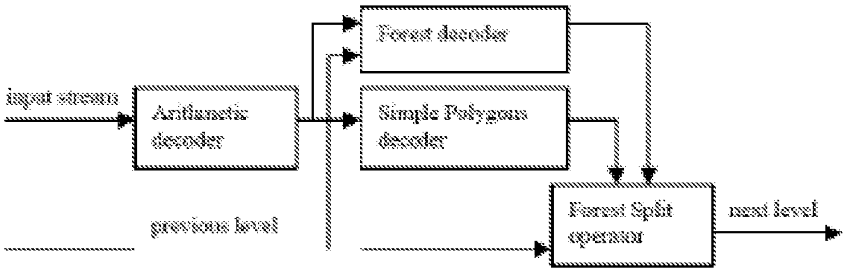


Figure V2 - 41 -- Block diagram of the Forest Split decoder

7.13.1 Start codes and bit stuffing

A start code is a two-byte code of the form '0000 0000 00xx xxxx'. Several such codes are inserted into the bitstream for synchronisation purposes. To prevent any wrongful synchronisation, such codes shall not be emulated by the data. This is guaranteed by the insertion of a stuffing bit '1' after each byte-aligned sequence of eight 0's. The decoder shall skip these stuffing bits when parsing the bit stream. Note that the arithmetic coder is designed such as to never generate a synchronisation code. Therefore the bit skipping rule does not need be applied to the portions of the bit stream that contain arithmetic coded data.

7.13.2 The Topological Surgery decoding process.

The connectivity of a 3D mesh is represented as a Simple Polygon (triangulated with a single boundary loop) with zero or more pairs of boundary vertices identified, and with zero or more internal edges labeled as polygon edges. When the pairs of corresponding boundary edges are identified, the edges of the resulting reconstructed mesh corresponding to boundary edges of the Simple Polygon form the Vertex Graph. The correspondence among pairs of Simple Polygon boundary edges is recovered by the decoding process from the structure of Vertex Graph, producing the Vertex Loop look-up table. The Vertex Graph is represented as (i) a rooted spanning tree, (ii) the Vertex Tree, and (iii) zero or more jump edges. The Simple Polygon is represented as (i) a rooted spanning tree, (ii) the Triangle Tree, which defines an order of traversal of the triangles, (iii) a sequence of marching patterns, and (iv) a sequence of polygon_edges. The marching patterns are used to reconstruct the trianges by marching on the left or on the right along the polygon bounding loop, starting from an inital edge called the root edge. The polygon edges are used to join or not to join triangles sharing a marching edge to form the faces of the mesh. As the triangles of the simple polygon are visited in the order of traversal of the Triangle Tree, simple polygon boundary edges are put in correspondence using the information contained in the Vertex Loop look-up table, and so, reconstructing the original connectivity.

© ISO/IEC 2001 – All rights reserved
Licensed to UNIVERSITY OF TEXAS AT ARLINGTON [illegible]
Single user licence only. Copying and networking prohibited.

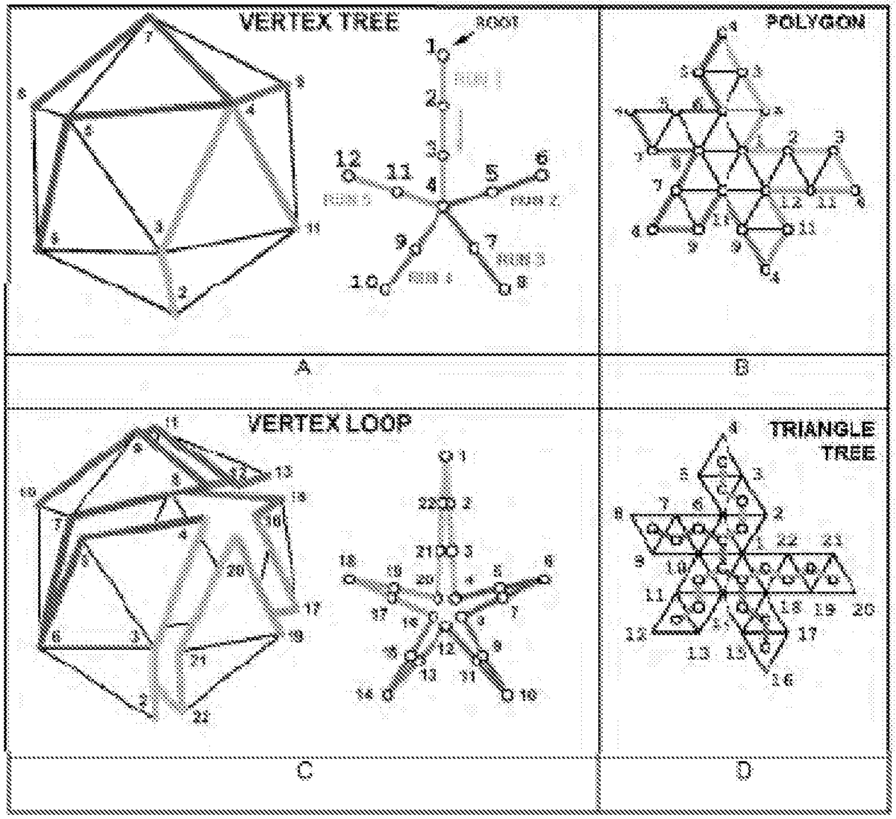


Figure V2 - 42 -- Topological Surgery representation of a simple mesh.
A : for a simple mesh, the vertex graph is a tree.
B : when a simple mesh is cut through the vertex tree, the result is a simple polygon.
C : each edge of the vertex tree corresponds to exactly two boundary edges of the simple polygon.
D : the triangle tree spanning the dual graph of the simple polygon. In the general case, the face forest is first constructed spanning the dual graph of the 3D mesh. The vertex graph is composed of the remaining edges of the 3D mesh. If the 3D mesh has polygonal faces, they are subsequently triangulated by inserting virtual marching edges, converting the face forest into a triangle forest with one triangle tree per connected component.

The geometry and property data are quantized and predicted as a function of ancestors in the order of traversal of the triangles. The corresponding prediction errors are transmitted in the same order of traversal, interlaced with the corresponding marching and polygon_edges, so that, as soon as all these triangle data are received, the decoder can output the corresponding triangle.

A non-manifold 3D mesh is represented as a manifold 3D mesh and a Vertex Clustering array. The Vertex Clustering array is represented in the bitstream as a sequence of stitching commands (or Stitches) with one command per vertex. The decoding process decodes the Stitches after the vertex graph and constructs the Vertex Clustering array using the Stitches. Figure V2 - 43 shows a 3D mesh with a singular vertex and a singular edge, with the polygons incident to the singular vertex highlighted.

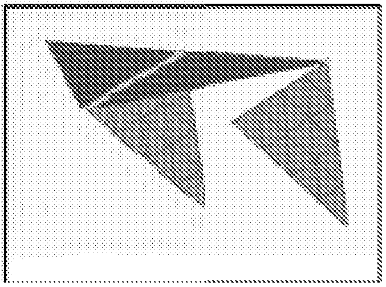


Figure V2 - 43 – 3D meshes with singular vertices and singular edges (each one with three incident faces) and singular edges (yellow)

At a high level, the Topological Surgery decoder performs he following steps:

```
Decode header information
for each partition {
 if (partition_type is 0) {
     for each connected component {
         decode vertex graph and construct bounding loop table
         if (has_stitches)
             decode stitches and build vertex clustering array
         decode triangle tree and construct table of triangle tree run lengths
         for each triangle in simple polygon {
             decode marching field and polygon field, and reconstruct connectivity
of triangle
             decode and reconstruct vertex coordinates and properties
         }
     }
 }
 else if (partition_type is 1) {
     for each vertex graph {
         decode vertex graph and construct bounding loop table
         if (has_stitches)
             decode stitches and build vertex clustering array
     }
 }
 else if (partition_type is 2) {
     decode partition header information
     decode triangle tree and construct table of triangle tree run lengths
     for each triangle in simple polygon {
         decode marching field, orientation field, and polygon field, and
reconstruct connectivity of triangle
         decode and reconstruct vertex coordinates and properties
     }
 }
}
```

7.13.3 The Forest Split decoding process.

The hierarchical mode representation is based on the Progressive Forest split (PFS) scheme. In this scheme a 3D mesh is represented as a low resolution base mesh followed by a sequence of forest split operations. A forest split operation is represented by a forest in the graph of vertices and edges of a mesh, a sequence of simple polygons, and a sequence of vertex coordinates and property updates. The mesh is refined by cutting the mesh through the forest, splitting the resulting boundaries apart, filling each of the resulting tree boundary loops with one of the simple polygons, and finally, displacing the new vertices and properties. An optional pre-smoothing step can be used as a gobal predictor for the vertex coordinates, after the connectivity and properties of the new faces bound per face and per corner are updated. Also, and optional post-smoothing step can be used to reduce the quantisation artifacts after all the updates have been applied.

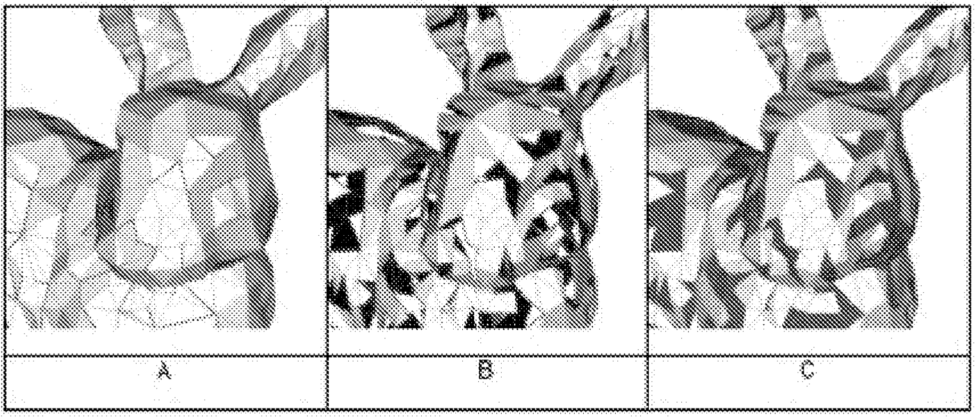


Figure V2 - 44 -- The Forest Split operation. A 3D mesh with a forest of edges marked (A). The tree loops resulting from cutting the 3D mesh through the forest edges (B). Each tree loop is filled by a simple polygon composed of polygonal faces (C).

At a high level, the Forest Split decoder performs the following steps:

```
Decode pre and post-smoothing parameters.
Decode Forest.
For each tree in the forest {
    Cut mesh through the tree edges, and enumerate mesh components.
    Decode Triangle Tree.
    Decode per triangle connectivity data and property updates
    for properties bound per face and per corner.
    Update connectivity by triangulating the tree loop
    according to the decoded simple polygon.
}
Optionally, apply pre-smoothing step to vertex coordinates as a global predictor.
For each tree in the forest {
    Decode property updates for all the tree loop vetices, faces, and corners
}
```

Optionally, decode vertex coordinates, and property updates for all the vertices, faces, and corners not incident to any triangle loop.

Optionally, apply post-smoothing step to vertex coordinates.

7.13.4 Header decoder

The header information is essentially divided into two parts. The first part contains information about the high level shading properties of the model, and the second defines the properties that are bound to the mesh.

7.13.4.1 High level shading properties

These are the **ccw**, **solid**, **convex**, **creaseAngle** fields defined in an indexedFaceSet node. The crease angle is quantized to a 6-bit value and is reconstructed as 2*pi***creaseAngle**/63.

7.13.4.2 Property bindings and quantisation parameters

There are four kinds of properties: vertex coordinates (coord), normals (normal), colors (color) and texture coordinates (texCoord). Properties can be bound to the mesh in four different ways: no binding, per vertex, per face and per corner. Not all combinations of bindings are valid. Table V2 - 32 lists the valid combinations. The binding of each property is obtained from coord_binding, normal_binding, color_binding and texCoord_binding, respectively.

Table V2 - 32 -- List of valid combinations of properties and bindings

	no binding	per vertex	per face	per corner
coord	forbidden	valid	forbidden	forbidden
normal	valid	valid	valid	valid
color	valid	valid	valid	valid
texCoord	valid	valid	forbidden	valid

For each property for which there is a binding, i.e. the binding field does not contain the no binding value, the following information is further decoded: a bounding box, a quantisation step, a prediction mode, a list of coefficients used for linear prediction. The decoding process for the vertex coordinates is further described below. The same decoding process applies to the other properties.

The presence of a bounding box is given by the field **coord_bbox**. The bounding box is a cube represented by the position of its lower left corner and the length of its side. For geometry these parameters are given by **coord_xmin**, **coord_ymin**, **coord_zmin** and **coord_size**. If no bounding box is coded, as default bounding box is assumed. The default bounding box has its lower left corner at the origin, and a unit size.

The next field **coord_quant** indicates the number of bits to which each coordinate is quantized to. The **coord_pred_type** field indicates the prediction mode. It should always be equal to '10'.

The **coord_nlambda** field indicates the number of coefficients used for linear prediction. It can take one of three values, namely '01', '10', and '11'. **coord_nlambda**-1 number of values are then read from the **coord_lambda** field. The last coord_lambda value is computed so that the sum of all the coord_lmabda value is equal to one. This field indicates the weight given to an ancestor for prediction. The floating point value of a coefficient is given by the decoded signed integer divided by 2 to the power **coord_quant**. The last coefficient is never transmited and is defined to be 1 minus the sum of all other coefficients.

The are some restrictions for the normal property: the **normal_bbox** field is always false, and the **normal_quant** value must always an odd number.

7.13.5 partition type

There are three partition types to convey vertext graph(vg), triangle tree(tt), and triangle data(td). The partition type specifies adnissible combinations of these three pieces of data in the bitstream. See figure V2 -45.

sc	vg	tt	td	last_component	…	vg	tt	td	last_component

(a)

sc	vg	last_vg	…	vg	last_vg

(b)

sc	codap_vg_id	left/codap_right_bloop_idx	codap_bdry_pred	tt	codap_branch_len	td

(c)

Figure V2 -45 – Types of data partition : (a) one or more group of vg, tt and td included in a partition. (b) one or more vgs included in a partition. (c) the partition has one pair of tt and td.

7.13.5.1 connected component partition (partition_type_0)

This partition indicates the bitstream with one or more sequence of connected component consisting of vertex graph, triangle tree, and triangle data. The end of the connected component sequence is determined by the **last_component** field. The following partition may be either type 0 or 1.

7.13.5.2 vertex graph partition (partition_type_1)

This partition indicates the bitstream with one or more sequence of vertex graph. The end of the vertex graph sequence is determined by the **last_vg** field. The partition following the vertex graph partition shall be tt/td pair partition.

7.13.5.3 tt/td pair partition (partition_type_2)

This partition indicates the bitstream with one pair of triangle tree and triangle data. The tt/td pair partition is characterized by the vertex graph id, visiting indices, and boundary prediction mode. These variables are given fields **codap_vg_id, left/codap_right_bloop_idx**, and **codap_bdry_pred. codap_vg_id** field indicates a vertex graph corresponding to the tt/td pair. It is used to get the bounding loop information from the vertex graph. **codap_left_bloop_idx** and **codap_right_bloop_idx** fieds indicate the left and right starting points of the bounding loop. They are also used to decide if a vertex in the partition is already visited in the previous partition. **codap_bdry_pred** field indicates if the vertices on the boundary of the partition should be decoded when the vertex is shared with previous partitions. If the partition ends at a branch, **codap_branch_len** field is added to the bitstream. Any type of partition may follow the tt/td pair partition.

7.13.5.3.1 Restricted boundary prediction mode (codap_bdry_pred=0)

The restricted boundary prediction mode does not duplicate vertices between partitions. Since the vertices predicted in the previous partitions may not be available, prediction is done only with the available vertices which are predicted in the current partition. If the previous partitions are lost, the triangles at the boundaries may not be reconstructed due to the fact that the vertices from the previous partitions are not available. However, the vertices predicted in the current partition may be reconstructed and decoding can continue.

The following process is applied for the prediction with a subset of all ancestors. Let the current method for prediction with 3 ancestors (a,b,c) be $d' = f(a,b,c)$. Then the equation is $d = d' + e$ and the value e is encoded. However, when the ancestors are visited in the previous partitions, the prediction method is as follows.

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA V. ABAWANE
ANSI Store order X[illegible] Downloaded: 7/18/2007 3:58:21 PM ET
Single user license only. Copying and networking prohibited.

```
if (all ancesors (a, b, and c) are not available) then $d' = 0$
else if (onlyone ancestor, say $t$, is available) then $d' = t$
else if (two ancestors, say $t_1$ and $t_2$, are available) then
        if (both ancestors are edge distance 1 from current vertex) then $d' = (t_1 + t_2)/2$
        else if ($t_1$ is edge distance 1 from current vertex) then  $d' = t_1$
        else $d' = t_2$
else  $d' = f(a,b,c)$.
```

7.13.5.3.2 Extended boundary prediction mode (codap_bdry_pred=1)

When this mode is selected, the decoder assumes that every vertex is not visited in the previous partition. Thus, after the three vertices of the root triangle are predicted, the vertices in the rest of the triangles in the partition will have all three ancestors for prediction.

7.13.6 Vertex Graph Decoder

The vertex graph decoder reconstructs a sequence of vertex runs. It then constructs the Vertex Loop look-up table.

For meshes that have simple topology and that do not have a boundary, the vertex graph does not contain any loops and is hence a tree. The field **vg_simple**, if set, indicates that the graph contains no loop. Some fields below are skipped when **vg_simple** is set.

Each vertex run is characterized by several variables, namely length, last, forward, loop and loop index. These variables are given by the fields **vg_run_length**, **vg_last**, **vg_forward_run**, **vg_loop** and **loop_index**. **vg_forward_run**, **vg_loop** and **loop_index** contain information relative to loops. When **vg_simple** is true, these three fields are not coded.

The process to reconstruct the vertex loop uses an auxiliary loop queue variable, initialy empty. When **vg_loop** is set the current run is put into the loop queue. When **vg_forward_run** is not set a run is pulled out from the queue. **loop_index** determines the position of the pulled out run in the queue. **vg_forward_run** is not coded when the loop queue is empty. **loop_index** is coded by its unary representation. When the index is equal to the number of elements in the queue minus 1, the trailing bit of its unary representation is not coded. Therefore when there is a single element in the queue, **loop_index** is not coded.

The end of the graph is determined by the **vg_last** and **vg_leaf** fields. An auxiliary depth variable is used for this purpose. It is decremented each time **vg_last** is set and incremented each time **vg_leaf** is not set. The depth is initialised to zero at the beginning of the graph. The end of the graph is reached when the depth variable becomes negative.

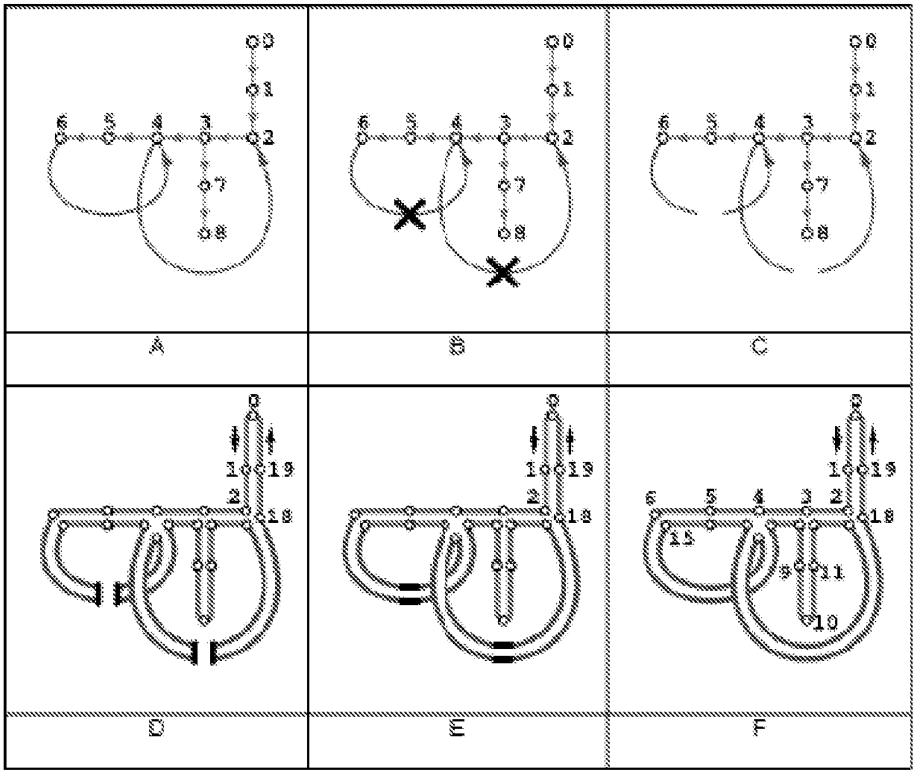


Figure V2 -46 – Steps to build the bounding loop from the vertex graph.
A : vertex graph decomposed into vertex tree (green) and jump edges (red).
B : extended vertex tree is created by cutting jump edges in half.
C : the extended vertex tree has two leaves for each jump edge.
D : build the extended vertex tree loop.
E : connect the start and end of each jump edge.
F : resulting boundary loop.

Vertex numbers are assigned according to a depth-first traversal of the graph. The bounding loop look-up table is constructed by going around the graph and for each traversed vertex recording its number.

7.13.6.1 Stitches Decoder

The Stitches decoder reconstructs the Vertex Clustering that should be applied to the vertices in the Vertex Graph to reconstruct the 3D mesh. The following pseudo-code summarizes the operation of the Stitches decoder: if the boolean **has_stitches** flag in the current component is true, then for each vertex of the Vertex Graph in decoder order, the Stitches decoder decodes a stitching command from the bitstream. If the boolean value **stitching_command** is true, then the boolean value **pop_or_get** is decoded; if the boolean value **pop_or_get** is false, an unsigned integer is decoded, and associated to the current vertex i as an anchor (to stitch to). The current vertex i is then pushed to the back of the anchor_stack.

If **pop_or_get** is true, then the boolean value **pop** is decoded, followed with the unsigned integer value **stack_index**.

```
decode(anchor_stack) {
 if (has_stitches == true)
  for (i = nV0; i < nV1; i++) {
            // nV0 is the first vertex of the current component,
            // and nV1 - 1 is the last vertex
   decode stitch_cmd;
    if (stitch_cmd) {
     decode stitch_pop_or_get;
     if (stitch_pop_or_get) {
      decode stitch_pop;
      decode stitch_stitch_stack_index;
      retrieve stitching_anchor from anchor_stack;
      if (stitch_pop) {
       remove stitching_anchor from anchor_stack;
      } // end if
      decode stitch_incr_length;
      if (stitch_incr_length != 0) {
       decode stitch_incr_length_sign;
      } // end if
      decode stitch_push;
      if (stitch_push)
       push i to the back of anchor_stack
      retrieve stitch_length at anchor;
      total_length = stitch_length + stitch_incr_length;
      if (total_length > 0)
       decode stitch_reverse;
      stitch i to anchor for length of total_length
      and in a reverse fashion if (stitch_reverse);
    }// end if (stitch_pop_or_get)

    decode stitch_length

    push i to the back of anchor_stack;

    save stitch_length at anchor i;

   } // end if (stitch_cmd)

 } // end for

}
```

Using **stack_index**, a stitching_anchor is retrieved from the anchor_stack. This is the anchor that the current vertex i will be stitched to. If the **pop** boolean variable is true, then the stitching_anchor is removed from the anchor_stack. Then, an integer **incremental_length** is decoded as an unsigned integer. If it is different from zero, its sign (boolean **incremental_length_sign**) is decoded, and is used to update the sign of **incremental_length**. A **push_bit** boolean value is decoded. If **push_bit** is true, the current vertex i is pushed to the back of the anchor_stack. An integer **stitch_length** associated with the stitching_anchor is retrieved. A total_length is computed by adding **stitch_length** and **incremental_length**; if total_length is greater than zero, then a **reverse_bit** boolean value is decoded. Then the Vertex Clustering array is updated by stitching the current vertex i to the stitching anchor with a length equal to total_length and potentially using a reverse stitch. The decoder uses a v_father array to perform this operation. The v_father array provides for each vertex in the Vertex Graph the index of its father (predecessor) in the directed Vertex Graph without jumps (potential the vertex itself if it is a root). To stitch the current vertex i to the stitching anchor with a length equal to total_length, starting from both i and the anchor at the same time, we walk along the Vertex Graph as indicated in the v_father entries for both i and the anchor by looking up the v_father entries total_length-1 times, and for each corresponding entries ((i, anchor), (v_father[i],v_father[anchor]), (v_father[v_father[i]], v_father[v_father[anchor]])) record in the Vertex Clustering array that the entry with the largest order of traversal should be the same as the entry with the lowest order of traversal. For decoding the stitches, For instance if (j > k), then Vertex Clustering[j] = Vertex Clustering[k]. The Vertex Clustering array defines a graph that is a forest. Each time an entry in the Vertex Clustering array is changed, we perform path compression on the forest by updating the Vertex Clustering array such that each element refers directly to the root of the tree of the forest it belongs to. The following figure shows both a (forward) stitch and reverse stitch.

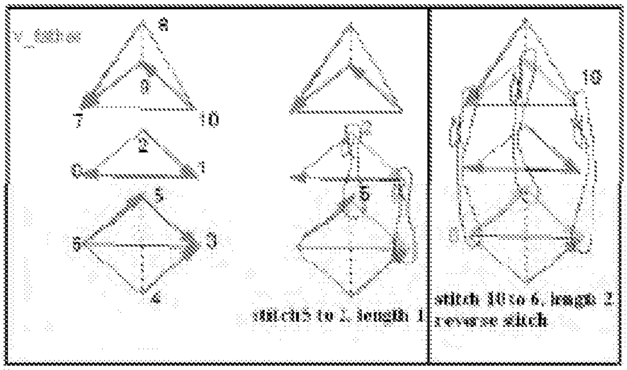


Figure V2 -47 – Examples of a (forward) stitch and a reverse stitch

If the stitch is a reverse stitch, then we first follow the v_father entries starting from the anchor for a length equal to total_length (from vertex 6 to vertex 3 in the figure above), recording the intermediate vetices in a temporary array. We then follow the v_father entries starting from the vertex i and for each corresponding entry stored in the temporary array (from the last entry to the first entry), we update the Vertex Clustering array as explained above.

The Vertex Clustering array is used to indicate when a coord or property should be decoded or was already decoded whendecoding triangle data as explained in clause 7.13.8 below.

7.13.7 Triangle Tree Decoder

The triangle tree decoder reconstructs a sequence of triangle runs. For each run it generates a run length and the size of its sub-tree.

The triangle tree is a sequence of triangle runs. For each run a length **tt_run_length** and a boolean flag **tt_leaf** is given. A branching run is a run for which **tt_leaf** is false and a leaf run a run for which **tt_leaf** is true. The decoder stops decoding the sequence of runs when the number of decoded leaf runs is superior to the number of branching runs. In the syntax a variable depth is used that counts the number of branching runs minus the number of leaf runs. This implies that the total number of runs is always an odd number.

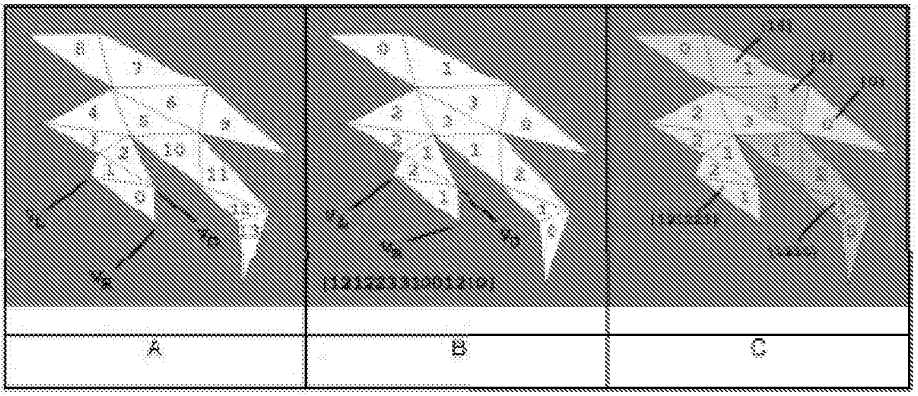


Figure V2 - 48 -- Representing a simple polygon as a table of triangle runs.
A: depth-first order of traversal of the triange tree starting from the root edge (green).
B: each triangle is classified by a 2-bit code as leaf(0), advance-right (1), advance-left (2), or branching (3).
C : The sequence of 2-bit codes is partitioned into runs ending in branching or leaf triangles. Each run is encoded as a (tt_run_length,tt_leaf) pair, plus a sequence of marching_edge bits.
Optionally, is the case of 3D meshes with polygonal faces, an additional sequence of polygon_edge bits is used to indicate which internal edges of the simple polygon correspond to original internal edges of the mesh, and which ones tc virtual internal edges.

The **tt_leaf** flag is also used for the computation of the length of each run.

If the length of each run is correctly decoded and does not have virtual triangles, then the sum of the length of all runs must be equal to the length of the bounding loop constructed by the vertex graph decoder minus 2 (assuming that the bounding loop is correctly reconstructed).

However, if a run is partitioned, the triangle tree bitstream containes one or two virtual leaf triangles. Hence when decoding triangle data, the triangle data corresponding to the virtual leaf triangles shall not be decoded.

There are one or two virtual triangles if the number of triangles in the current partition is less than **codap_right_bloop_idx - codap_left_bloop_idx** - 1. Otherwise there are no virtual tiangles. If there are virtual triangles and the third-last triangle is a branching triangle, the last two triangles in the partition are the virtual triangles. Other wise only the last triangle is virtual.

Licensed to UNIVERSITY OF TEXAS AT [illegible]
Single user license only. Copying and networking prohibited.

7.13.8 Triangle Data Decoder

The triangle data decoder operates in three steps. It first decodes a header, then a root triangle, which is followed by a sequence of triangles. Note that this sequence is empty if there is a single triangle in the current connected component.

The header is a boolean flag **triangulated** that indicates if all faces in the connected component are triangles. If this flag indicates true, the **polygon_edge** field is not coded.

The decoding of a root triangle and of a non-root triangle are essentially the same. However the **polygon_edge** field is never coded for a root triangle of a connected component, and the number of property samples may differ.

Two fields indicate connectivity information for the current triangle: **marching_edge** and **polygon_edge**. **marching_edge** is not coded for branching and leaf triangles. **polygon_edge** is not coded for the root triangle or if the **triangulated** flag is set true. However in a partition of partition_type_2, the **polygon_edge** is coded for the root triangle if the **triangulated** field is set false

The number of per vertex property samples may differ depending on the boundary prediction mode and visited vertices. If the boundary prediction mode is restricted, then the two starting points are already visited in the previous partition and only one root sample is coded. However, if the remaining vertex is also visited, there are no samples coded. If the boundary prediction mode is extended, the root triangle always has all 3 samples.

Table V2 - 33 – Number of samples to decode in a root triangle for each binding

Binding	**# samples**	**Detail**
None	0	
per vertex	0, 1, or 3	1 root and 2 non-root samples
per face	1	1 root sample
per corner	3	2 root and 1 non-root samples

For non-root triangles, the triangle data decoder decodes zero or one sample for each property bound per vertex. If the opposite vertex of the current triangle has been visited before, the value of the sample is already known and does not need to be decoded again. If the opposite vertex of the current triangle has never been visited before, then a sample is read. The bounding loop look-up table and the sizes of subtrees of the triangle tree are necessary to determine whether the opposite vertex of the current triangle has been visited previously. The boundary prediction mode is necessary to determine whether to consider the visited vertices within the partition or including every vertices visited in the previous partitions.

Table V2 - 34 – Number of samples to decode in a non-root triangle for each binding. The number of sampes may be conditioned.

Binding	**# samples**	**Condition**
None	0	
pervertex	0 or 1	vertex never visited
perface	0 or 1	**polygon_edge** is set
percorner	1 or 3	**polygon_edge** is set

The traversal order is fixed in basic topological surgery. However, in order to change traversal order, **td_orientation** field is coded at a branching triangle on the main stem of the triangle tree in the partition. In other words, if the number of previously decoded branching triangles is equal to that of leaf triangles within the current partition, i.e. the depth value is zero, the **td_orientation** field is coded.

7.13.8.1 Decoding a root sample

A root sample is the first decoded sample in a root triangle. If there are several samples in a root triangle only the first sample is a root sample. The remaining samples in the root triangle are non-root samples.

The **coord_quant** bits of each coordinate of a root geometry sample are received starting with the most significant.

7.13.8.2 Decoding a non-root sample

A coordinate of a non-root geometry sample is composed of **coord_quant** bits, plus a sign bit. The coordinate is coded bit by bit starting from the most significant. The leading 0's, if any, and the leftmost 1, if any, are **coord_leading_bit**'s are coded using an adaptive probability estimation model. If not all **coord_leading_bits** are 0's a **coord_sign_bit** is coded. Finally **coord_quant** minus the number of **coord_leading_bits** **coord_trailing_bits** are coded using a fixed probability model.

3*****coord_quant** probability estimation models are defined to code the **coord_leading_bit** field. The probability model is selected according to the bit position and to the coordinate number.

The decoded value is not the actual value of the sample, but a correction that has to be applied to the prediction to obtain the reconstructed sample. The reconstructed sample is equal to the sum of the prediction and of the decoded value. The computation of the prediction is described below.

It may happen that a reconstructed value is outside the bounding box, in which case a masking operation is carried out to insure that the final reconstructed value is indeed inside the bounding box. The masking operation depends on the quantisation step **coord_quant**. The **coord_quant** less significant bits of the reconstructed value are kept as is, and all other (more significant) bits are cleared.

The same structure applies to the other properties.

7.13.8.3 Property Prediction

For increased coding efficiency, the value of a property is coded relative to a prediction. This subclause describes the computation of the prediction for each type of binding. Table V2 - 35 and Table V2 - 36 describe all the valid combinations of property binding, property type, and prediction type.

Table V2 - 35 -- Valid property bindings for each property type

binding	**meaning**	**coord**	**normal**	**color**	**texCoord**
00	not encoded		X	X	X
01	per vertex	X	X	X	X
10	per face		X	X	
11	per corner		X	X	X

Table V2 - 36 -- Valid combinations of prediction type and property binding

pred_type	**nlambda**	**binding**
no_prediction	not coded	bound_per_vertex, bound_per_face or bound_per_corner
tree_prediction	1	bound_per_face or bound_per_corner
parallelogram_prediction	3	bound_per_vertex

For tree and parallelogram prediction, the prediction of the properties in the current triangle are always based on the values in the previous triangle in the triangle tree. The prediction is weighted by the lambda coefficients decoded by the header decoder.

[illegible]
[illegible]
Single user license only. Copying and networking prohibited.

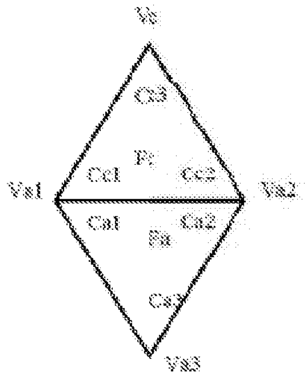


Figure V2 -49 – Prediction of property values.
The value at the current vertex Vc is predicted from the values at ancestor vertices Va1, Va2 and Va3.
The value at the current face Fc is predicted from the value at ancestor face Fa.
The value at the current corner Cc1 is predicted from the value at ancestor corner Ca1.
The value at current corner Cc2 is predicted from the value at ancestor corner Ca2.
The value at current corner Cc3 is predicted from the value at current corner Cc1.

7.13.8.3.1 Tree prediction

If the property is bound per face, the prediction of Fc is equal to Fa. If the current triangle is the root of the triangle tree, Fc is not predicted.

If the property is bound per corner, the prediction of Cc1 is equal to Ca1, the prediction of Cc2 to Ca2, and the prediction of Cc3 to Cc1. If the current triangle is the root of the triangle tree, Cc1 is not predicted, Cc2 is predicted by Cc1, and Cc3 by Cc1.

7.13.8.3.2 Parallelogram prediction

If the property is bound per vertex, the prediction of Vc is equal to lambda1*Va1 + lambda2*Va2 + lambda3*Va3. If the current triangle is the root of the triangle tree, Va1 and Va2 must be encoded too, Va1 is not predicted, Va2 is predicted by Va1, and Vc by Va1.

7.13.8.4 Inverse quantisation

There are two inverse quantisation procedures. The first one applies to geometry, colors and texture coordinates. The second applies to normals.

Given a quantised geometry sample (qx, qy, qz), the reconstructed geometry sample (x,y,z) is obtained as:

x = coord_xmin + ((1<<coord_quant)-1)*coord_size*xq
y = coord_ymin + ((1<<coord_quant)-1)*coord_size*yq
z = coord_zmin + ((1<<coord_quant)-1)*coord_size*zq
The procedure for colors and texture coordinates is the same.

There is a special inverse quantisation procedure for normals. Normals are always quantised to an odd number of bits **normal_quant**. The 3 most significant bits of **normal_quant** indicate the octant the normal lies in, and the remaining bits define an index within the octant.

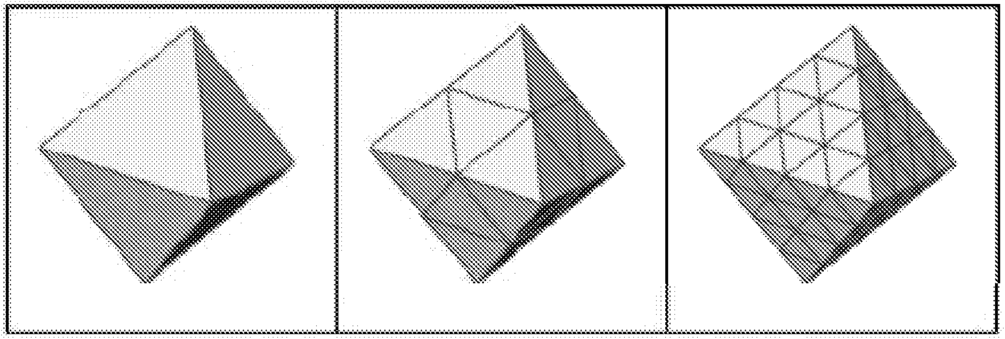


Figure V2 -50 -- The quantisation of normals is based on
the hierarchical subdivision of the unit octahedron.
Left: unit octahedron.
Middle: unit octahedron after one subdivision.
Right: unit octahedron after two subdivisions.

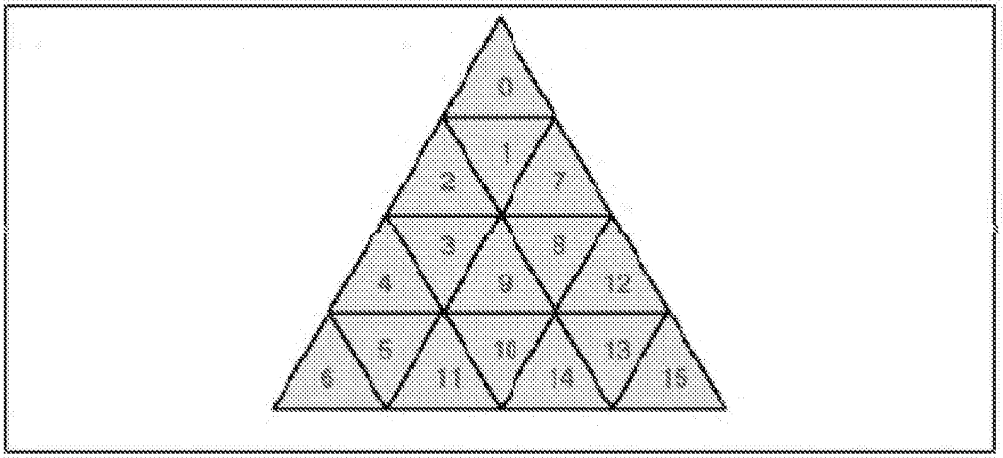


Figure V2 -51 -- Numbering of the triangles within an octant.
Note that the top corner corresponds to |z|=1,
the leftmost corner to |x|=1 and the rightmost corner to |y|=1.

Given an index i, and the 3 sign bits sx, sy and sz, the normal (x,y,z) can be analytically reconstructed as follows:

```
nbins = 1 << ((normal_quant - 3) / 2);

y0 = nbins - ceilsqrt(nbins*nbins - i);
x0 = i + y0*y0;
skew = (x0 & 1)*2.0/3.0;
x0 = (x0 >> 1) &(nbins - 1);

x = (float)x0 + skew;
y = (float)y0 + skew;
z = (float)nbins - x - y;

n = 1.0/sqrt(x*x + y*y + z*z);
x = (sx) ? -x*n : x*n;
y = (sy) ? -y*n : y*n;
z = (sz) ? -z*n : z*n;
```

7.13.9 Forest Split decoder

In the progressive forest split (PFS) scheme, a 3D model is represented as a low resolution mesh followed by a sequence of forest split operations. A forest split operaton is represented by a forest in the graph of vertices and edges of a mesh, a sequence of simple polygons, and asequence of vertex coordinates and property updates. The mesh is refined by cutting the mesh through the forest, splitting the resulting boundaries apart, filling each of the resulting tree boundary loops with one of the simple polygons, and finally, dispacing the new vertices and properties. An optional pre-smoothing step can be used as a global predictor for the vertex coordinates, after the and properies of the new faces bound per face and per corner are updated. Also, and optional post-smoothing step can be used to reduce the quantisation artifacts after all the updates have been applied.

7.13.9.1 Enumeration of Mesh Elements

Given a manifold triangular 3D mesh with V vertices, T triangles and no boundaries, we assume that the vertices have consecutive *vertex indices* in the range $0, \ldots, V-1$ and the triangles have consecutive *triangle indices* in the range $0,\ldots,T-1$. The edges of the mesh, which are represented by pairs of vertex indies (i,j), with $i < j$, are ordered lexicographically and assigned consecutive *edge indices* in the range $0,\ldots,E-1$. The trees in the forest are ordered according to the minimum vertex index of each tree. The *root vertex* of each tree in the forest is the leaf of the tree with the minimum index. Starting at the root, the boundary loop created by cutting along the tree can be traversed in cyclic fashion in one of the two directions. The *root edge* of the tree is the only edge of the tree that has the root vertex as an endpoint. Of the two triangles incident to the root edge of the tree, the *root triangle* of the tree is the one with minimum triangle index. Of the two edges of the tree boundary loop corresponding to the root edge of the tree, the *root edge* of the tree boundary loop is the one incident to the root triangle.

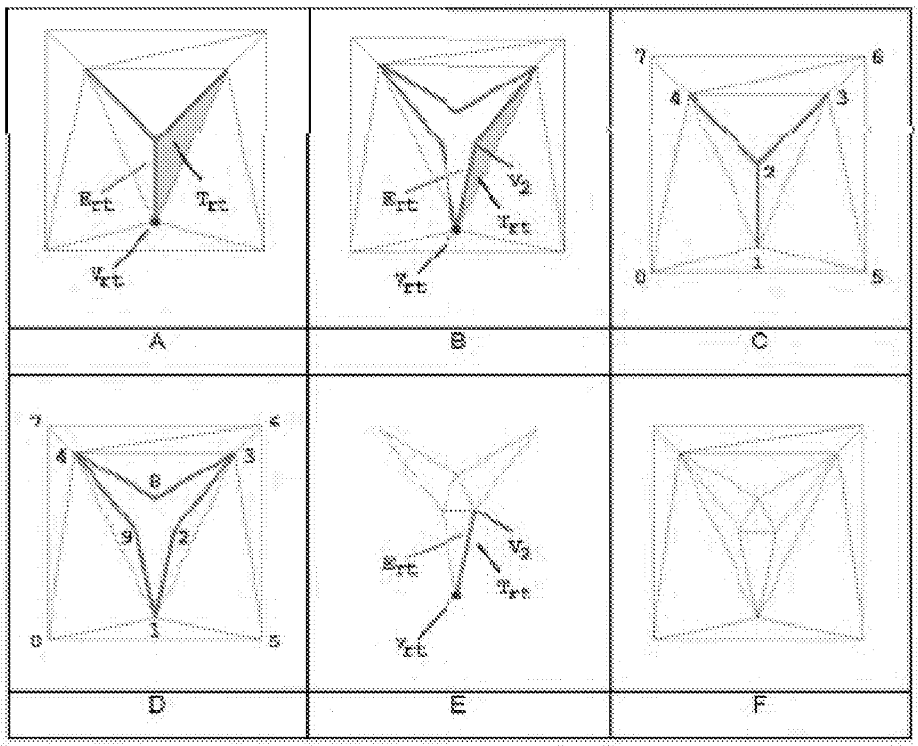


Figure V2 -52 – When a mesh is cut through a tree of edges (red and green edges in A), a tree boundary loop (red and green edges in B) is created with each edge of the tree corresponding to two edges of the boundary loop. Some vertex indices are assigned before cutting (in C) to new tree boundary loop vertices, others are assigned subsequent indices (in D). The hole created by the cutting operation is filled by triangulating the boundary loop using a simple polygon (E) resulting in a refined mesh (F) with the same topological type as the initial mesh.

7.13.9.2 Decoding the forest

Figure V2 -53 describes the algorithm to decode the forest of edges from the bitstream. Initially, the forest is empty. As edge marks are decoded, the corresponding marked edges are included in the forest. Edges which if marked would create loops are skiped, because they can be predicted as not being marked. The order of the edge bits in the bitstream correspond to the enumeration of edges defined above.

```
// forest decoder

// marked edges form a forest in the graph of the mesh



int nV = number of vertices of mesh

Partition p(nV)

// p defines a partition of a set of nV elements

// initially each part contains a single element

// the method find(i) retrieves the number of the part containing element i

// the method join(i,j) joins the parts containing elements i and j respectively



for ( every edge e={i,j} of mesh ) {

  bool skip_edge = false;

  bool edge_bit  = false;

  if(p.find (i)!=p.find(j)) {

    edge_bit = decode_bit();

  } else {

    // forest => edge_bit==false

    skip_edge = true;

  }

  e.isInforest = edge_bit;

  if(edge_bit==true)

    p.join(i,j);

}
```

Figure V2 -53 – forest decoder pseudo-code

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA Y. AGRAWAL
ANSI Store order X037445. Downloaded: 7/18/2007 3:56:21 PM ET
Single user license only. Copying and networking prohibited.

7.13.9.3 Cutting Through Forest Edges

Cutting through a forest of edges can be performed sequentially, cutting through one tree at a time. Each cut is typically a local operation, affecting only the triangles incident to vertices and edges of the tree. However, as in the Topological Surgery method, a single cut could involve all the triangles of the mesh. Cutting requires duplicating some tree vertices, assigning additional indices to the new vertices and fixing the specification of the affected triangles. If no tree vertex is a boundary vertex of the mesh, then the tree is completely surrounded by triangles. This is illustrated in Figure V2 -54--A,B.

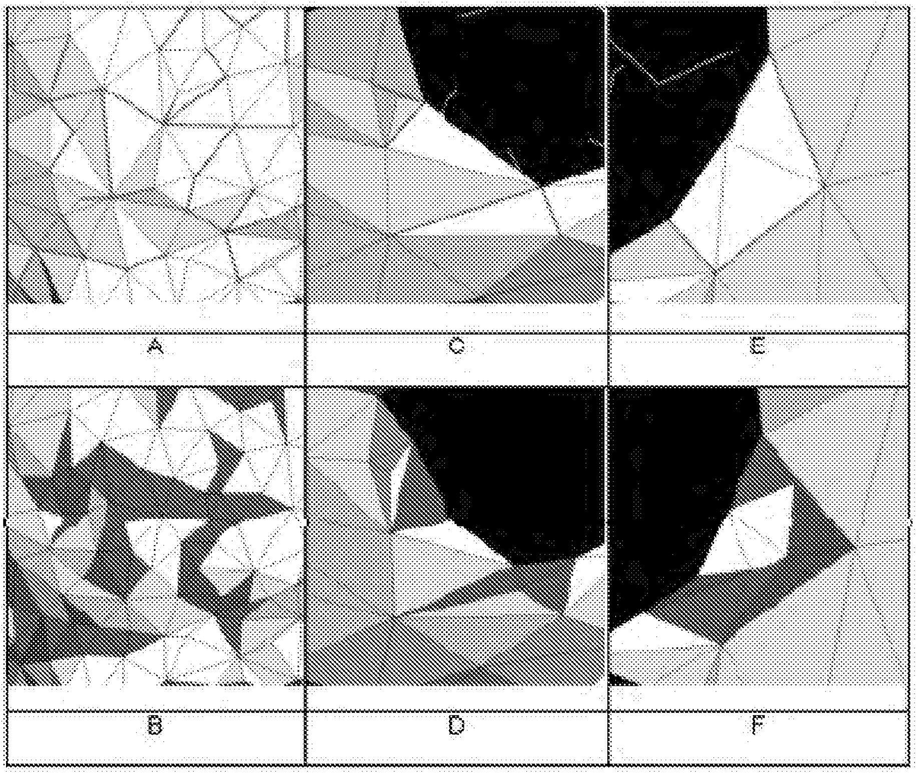


Figure V2 -54 -- Construction and triangulation of tree boundary loops.
A,B: No tree vertices belong to the mesh boundary.
C,D: A tree vertex isolated in the mesh boundary requires an extra tree loop edge.
E,F: A tree edge on the mesh boundary does not require an extra tree loop edge, but some of the new vertex indices may only be used by new triangles.
Note that the tree may have several contacts with the mesh boundary.

Starting at the root triangle, all the corners of affected triangles can be visited in the order of traversal of the tree boundary loop, by jumping from triangle to neighboring triangle, while always keeping contact with the tree. The direction of traversal is defined by going from the root edge *er* towards the root triangle *tr*. This process produces a list of triangle corners, called the *corner loop*, whose values need to be updated with the new vertex indices. While traversing this list, we encounter runs of corners corresponding to the same vertex index before the cut. A new vertex index must be assigned to each one of these runs. To prevent gaps in the list of vertex indices we first need to reuse the vertex indices of the tree vertices, which otherwise would not be corner values of any triangles. The first visited run corresponding to one of these vertices is assigned that vertex index. The next visited run corresponding to the same vertex index is assigned the first vertex index not yet assigned above the number of vertices of the mesh before the cut. This procedure performs the topological cut. For example, in Figure V2 -54--C, the vertex index values of the corners in the corner loop list are:

[12333332444444421111]

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA V. ABAWANE
ANSI Store order X037445? Downloaded: 7/18/2007 3:58:21 PM ET
Single user license only. Copying and networking prohibited.

The list can be decomposed into 6 runs [11111], [2], [33313], [2], [444444] and [2]. As shown in Figure V2 -54–D, the vertex indices that are assigned to these runs are 1,2,3,8,4,9. A tree with *m* edges containing no mesh boundary vertices creates a tree boundary loop of *2m* edges. When one or more vertices belong to the mesh boundary, the boundary loop has some edges without incident triangles. Cases when vertices belong to the mesh boundary are illustrated in Figure V2 -54–C, D, E, F. There are essentially three cases: tree edges belonging to the boundary (Case 1, shown in Figure V2 -54–E,F), a tree vertex that is isolated on the mesh boundary (Case 2, shown in Figure V2 -54–C, D) or an isolated tree vertex on the boundary that happens to be the root vertex v_r (Case 3, which is a sub-case of Case 2). Case 1 is treated similarly to the general case: we loop around the vertex tree, creating 2 new loop edges for each marked edge. Some of these edges do not belong to any existing triangle. The method that is used for looping around boundary edges must take into account that there are no triangles on the other side. One possibility is to create fake faces on the other side of the boundary and keep the same method for jumping from triangle to triangle. Another possibility, which is exploited in the sample implementation, is to start jumping in the other direction when we hit the boundary. Except from these difference in the algorithm, the same method is used to assign new vertex indices for some of the loop corners. In Case 2, an extra edge should be added to the loop. In Case 3, an extra edge is added as well, but it remains to determine which of the two endpoints of the created edge will be the root vertex of the loop. By convention, it will be the endpoint to which the root triangle is incident. These three cases are illustrated in Figure V2 -55. In summary, a tree with *m* edges containing *n* isolated boundary vertices creates a tree boundary loop of *2m+n* edges

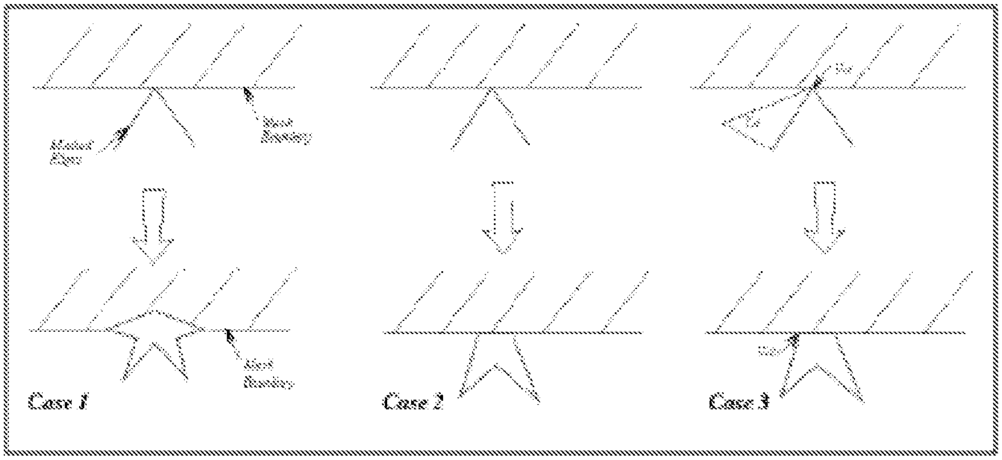


Figure V2 -55 -- Special cases encountered while constructing the tree boundary loop corresponding to a tree touching the boundary of the mesh

7.13.9.4 Triangulating Tree Boundary Loops

By replacing each run of corners in the corner loop with the assigned vertex index, we construct a new list representing the tree boundary loop. If the tree boundary loop has m vertices, so does the corresponding polygon boundary loop. Each triangle *t = (i,j,k)* of the simple polygon defines a new triangle of the refined mesh by replacing the polygon boundary loop indices *i,j,k* with their corresponding tree boundary loop indices. *Figure V2 -54 – E,F* illustrates this. This is done using the list representing the tree boundary loop as a lookup table. The triangles of the simple polygon are visited in the order of a depth first traversal of its dual tree. The traversal starts with the triangle opposite to the root triangle and always traverses the left branch of a branching triangle first.

7.13.9.5 Simple polygon properties bound per face and per corner

The properties bound per face and per corners to the faces of the simple polygons used to fill the tree boundary loops are encoded as in the case of a connected component of a single resolution 3D mesh, predicting per face values along the face tree, and per corner values along the derived corner tree.

© ISO/IEC 2001 – All rights reserved
Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/ROOP,K.Y. ADMIN
ANSI Store order X037445 Downloaded 7/18/2007 3:56:21 PM ET
Single user license only. Copying and networking prohibited.

7.13.9.6 Tree loop vertex, face, and corner properties

Vertex coordinates and properties bound per vertex corresponding to new tree loop vertices are first assigned the same values as the corresponding tree vertex coordinates and properties before the cut. To prevent the appearance of holes, these vertices are displaced after the boundary loops are triangulated. Rather than encoding the new absolute positions of the tree loop vertices and corresponding per vertex properties, we encode their displacement with respect to the position before the forest split operation, and optionally after pre-smoothing (pre-smoothing is only applied to vertex coordinates). These values are encoded in the order of traversal of the tree loop vertices.

Properties bound per face and per corner to tree loop faces and corners are encoded in a similar fashion, except that properties bound per face (a face may be a tree loop face for two or more trees) are encoded with no repetition, skipping values associated with faces previously visited during the current update cycle.

7.13.9.6.1 Pre and post-smoothing

The differences between vertex positions before and after each forest split operation can be made smaller by representing these errors as the sum of a global predictor plus a correction. We use the signal processing smoothing method [4] in Annex O as a global predictor. Figure V2 -56 shows a pseudo-code implementation of the scheme used for both pre and post smoothing

```
// pre and post smoothing



int    nV     // number of vertices of mesh

Vertex v[nV]  // array of vertices of the mesh

Vertex dv[nV] // auxiliary array

Int   nv[nV] // auxiliary array



// determine sharp vertices from sharp edges

for(i=0;i<nV;i++)

  v[i].isSharp = false

for (every edge e={i,j} of mesh ) {

  if(e.isSharp) {

    v[i].isSharp = true

    v[j].isSharp = true

  }

}



// main loop

for(n=0;n<n_smooth;n++) {
```

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/[illegible]
ANSI Store order X[illegible] Downloaded: 7/18/2007 [illegible] PM ET
Single user license only. Copying and networking prohibited.

```
// accumulate displacements
 dv = 0;
 nv = 0;
 for ( every edge e=(i,j) of mesh ) {
  dve = v[j] - v[i];
  if(v[i].isFixed==false && (e.isSharp==true | v[i].isSharp==false)) {
   dv[i] += dve;
   nv[i] += 1;
  }
  if(v[j].isFixed==false && (e.isSharp==true | v[j].isSharp==false)) {
   dv[j] -= dve;
   nv[j] += 1;
  }
 }

 // normalize displacements and apply
 float a= (n is even)?lambda_smooth:mu_smooth;
 for(i=0;i<nV;i++)
  if(nv[i]>0) {
    dv[i] /= nv[i]
    v[i] += a*dv[i];
  }
}
```

Figure V2 -58 – Pseudo-code for smoothing operator with discontinuity edges and fixed vertices

An optional pre-smoothing step is applied as a global predictor. After the connectivity refinement step of a forest split operation is applied, the new vertices are positioned where their corresponding vertices in the previous level were positioned and the mesh has many edges of zero length (all the new triangles have zero surface area). The signal processing smoothing method, which tends to equalize the size of neighboring triangles, brings the endpoints of most such edges apart, most often reducing the distance to the desired vertex positions. The corrections, the differences between the vertex positions after the split operation and the result of smoothing the positions before the split operation, are then quantized.

If the 3D mesh approximates a smooth shape, a post-smoothing step can be applied to reduce the visual artifacts produced by the quantisation process. In many cases, even fewer bits per coordinate can be used in the quantisation process without a significant perceptual difference, also reducing the total length of the encoded vertex displacements.

In both the pre and post-smoothing steps optional discontinuity edges and/or fixed vertices can be specified in the bitstream, together with the smoothing parameters. If so, the same discontinuity edges and/or fixed vertices are used for both steps.

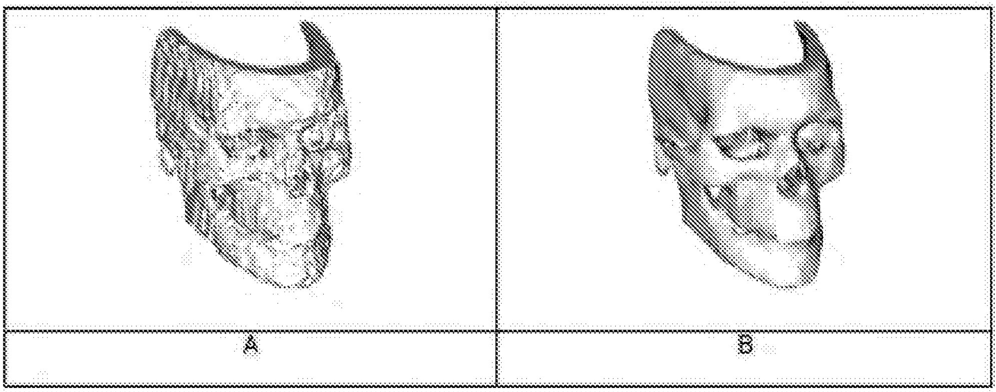

A
B

Figure V2 -57 -- Smoothing a 3D model.
(A) Original 3D model. (B) 3D model after applying the smoothing operator introduced in [4] in Annex O.

7.13.10 Arithmetic decoder

This section describes the QF-coder for arithmetic decoding. C++-style routines are provided. The arithmetic decoder relies on a set of variables that are described in the table below.

Register	Description
A	size of current interval
C	current arithmetic code
count	number of renormalisations before reading next byte
symbol	value of decoded symbol
Qe	probability estimate for least probable symbol
MPS	value of most probable symbol

The registers Qe and MPS are dependent upon a context. For each context, there is a state variable including the Qe and MPS registers.

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/POOJA V. ADAWANE
Single user license only. Copying and networking prohibited.

7.13.10.1 Initialisation

The arithmetic coder reads data from the input on a per byte basis. Therefore the stream pointer shall be byte aligned before initialising the arithmetic decoder.

In the initialisaion process, all variables are initialised to poper values. 2 bytes are read from the input stream into register C.

```
void ql_start() {

  A = 0x00010000;

  C = 0x00010000;

  count = 0;

  sw_renorm_decode();

}
```

7.13.10.2 Decoding a symbol

When a binarysymbol is decoded, the interval is split into two. The half in which C lies determines the value of the bit. The interval is then reduced to the corresponding half. If the size of the new interval is smaller than 0x80000000, then the interval is renormalised and the probability estimator updated.

```
void ql_decode(int *symbol, QState *state) {

  *symbol = state->MPS;


  A -= state->Qe;


  if (C <= A) {

   if (A & 0x80000000)

     return;

   *symbol ^= A < state->Qe;

  }

  else {

   *symbol ^= A >= state->Qe;

   C = C - A;

   A = state->Qe;

  }


  state->update(*symbol);

  renorm_decode();

}
```

7.13.10.3 Renormalisation

In the renormalisation step, the size of the interval is doubled until it reaches at least 0x80000000. Each time the size of the interval is doubled a bit is consumed. When no more bit is available for consumption in the less significant bits of the register C, a new byte is retrieved fom the stream.

```
void qf_renorm() {

  do {

    if (count == 0) {

      int b = get_byte();

      C -= 0xff << 8;

      C += b << 8;

       count =(b == 0) ? 7 : 8;

    }

    count–;

    C = C << 1;

    A = A << 1;

  } while (!(A & 0x80000000));

}
```

7.13.10.4 Probability estimation

A majority of the fields that are arithmetic coded benefit from adaptive probability estimation. This adaptive estimation is defined by a Markov Model. Each state of the model defines a probability of the LPS, a next state if the currently coded symbol is the MPS, a next state if the currently coded symbol is the LPS, and a flag that indicates if the value of the MPS should be changed if the currently coded symbol is the LPS. The Markov model to be used is defined in Figure V2 - 40. It is named FA-JPEG for Fast-Attack JPEG, and has been designed such as to minimize the number of states.

A fixed probability of the LPS and a fixed MPS are defined for the fields for which there is no adaptive probability estimation.

Table V2 - 37 lists the probability estimation used for each of the fields.

The models are global and initialised to index = 0, MPS = 0 before the first connected component.

The context of order 1 models is reset before each component.

The state is updated every time the renormalisation procedure is called.

Table V2 - 37 -- List of fields that are arithmetic coded and their associated probability estimation

Field	probability estimation	context
last_component	FA-JPEG	last_component_context
has_stitches	FA-JPEG	has_stitches_context
codap_last_vg	FA-JPEG	codap_last_vg_context
codap_vg_id	FA-JPEG	zero_context
codap_left_bloop_idx	FA-JPEG	zero_context
codap_right_boop_idx	FA-JPEG	zero_context
codap_bdry_pred	FA-JPEG	zero_context
vg_simple	FA-JPEG	vg_simple_context
vg_last	FA-JPEG	vg_last_context
vg_forward_run	FA-JPEG	vg_forward_run_context
vg_loop_index	FA-JPEG	vg_loop_index_context
vg_run_length	FA-JPEG	vg_run_length_context
vg_leaf	FA-JPEG	vg_leaf_context
vg_loop	FA-JPEG	vg_loop_context
tt_run_length	FA-JPEG	tt_run_length_context
tt_leaf	FA-JPEG	tt_leaf_context
codap_branch_len	FA-JPEG	codap_branch_len_context
triangulated	FA-JPEG	triangulated_context
marching_edge	FA-JPEG, order 1, initial context = 1	marching_edge_context[0..1]
td_orientation	FA-JPEG	td_orientation_context
polygon_edge	FA-JPEG, order 1, initial context = 1	polygon_edge_context[0..1]
coord_bit	fixed Qe = 0x5601, MPS = 0	zero_context
coord_leading_bit	FA-JPEG, 3*coord_quant contexts	coord_leading_bit_context[0..3*coord_quant-1]
coord_sign_bit	fixed Qe = 0x5601, MPS = 0	zero_context
coord_trailing_bit	fixed Qe = 0x5601, MPS = 0	zero_context
normal_bit	fixed Qe = 0x5601, MPS = 0	zero_context
normal_leading_bit	FA-JPEG, normal_quant contexts	normal_leading_bit_context[0..normal_quant-1]
normal_sign_bit	fixed Qe = 0x5601, MPS = 0	zero_context
normal_trailing_bit	fixed Qe = 0x5601, MPS = 0	zero_context
color_bit	fixed Qe = 0x5601, MPS = 0	zero_context
color_leading_bit	FA-JPEG, 3*color_quant contexts	color_leading_bit_context[0..3*color_quant-1]
color_sign_bit	fixed Qe = 0x5601, MPS = 0	zero_context
color_trailing_bit	fixed Qe = 0x5601, MPS = 0	zero_context
texCoord_bit	fixed Qe = 0x5601, MPS = 0	zero_context
texCoord_leasing_bit	FA-JPEG, 2*texCoord_quant contexts	textCoord_leading_bit_context[0..2*texCoord_quant-1]
texCoord_sign_bit	fixed Qe = 0x5601, MPS = 0	zero_context
texCoord_trailing_bit	fixed Qe = 0x5601, MPS = 0	zero_context
stitch_cmd	FA-JPEG	stitch_cmd_context
stitch_pop_or_get	FA-JPEG	stitch_pop_or_get_context
stitch_pop	FA-JPEG	stitch_pop_context
stitch_stack_index	FA-JPEG	stitch_stack_index_context
stitch_incr_length	FA-JPEG	stitch_incr_length_context
stitch_incr_length_sgn	FA-JPEG	stitch_incr_length_sgn_context

stitch_push	FA-JPEG	stitch_push_context
stitch_reverse	FA-JPEG	stitch_reverse_context
stitch_length	FA-JPEG	stitch_length_context
pfs_forest_edge	FA-JPEG	pfs_forest_edge_context
smooth_with_sharp_edges	fixed Qe = 0x5601, NPS = 0	zero_context
smooth_sharp_edge	FA-JPEG	smooth_sharp_edge_context
smooth_with_fixed_vertices	fixed Qe = 0x5601, NPS = 0	zero_context
smooth_fixed_vertex	FA-JPEG	smooth_fixed_vertex_context
other_update	fixed Qe = 0x5601, NPS = 0	zero_context

Table V2 - 38 – The Fast Attack JPEG (FA-JPEG) Markov model to estimate probabilities. For each state, we define the probability of the LPS, the next state in the event of an MPS, the next state in the event of an LPS, and a switch MPS flag in the event of an LPS.

State	Probability of LPS	Next state if MPS	Next state if LPS	Switch MPS if LPS
0	0x5601	1	1	yes
1	0x3401	2	6	no
2	0x1801	3	9	no
3	0x0ac1	4	12	no
4	0x0521	5	29	no
5	0x0221	38	33	no
6	0x5601	7	6	yes
7	0x5401	8	14	no
8	0x4801	9	14	no
9	0x3801	10	14	no
10	0x3001	11	17	no
11	0x2401	12	18	no
12	0x1c01	13	20	no
13	0x1601	29	21	no
14	0x5601	15	14	yes
15	0x5401	16	14	no
16	0x5101	17	15	no
17	0x4801	18	16	no
18	0x3801	19	17	no
19	0x3401	20	18	no
20	0x3001	21	19	no
21	0x2801	22	19	no
22	0x2401	23	20	no
23	0x2201	24	21	no
24	0x1c01	25	22	no
25	0x1801	26	23	no
26	0x1601	27	24	no
27	0x1401	28	25	no
28	0x1201	29	26	no
29	0x1101	30	27	no
30	0x0ac1	31	28	no
31	0x09c1	32	29	no

32	0x08a1	33	30	no
33	0x0521	34	31	no
34	0x0441	35	32	no
35	0x02a1	36	33	no
36	0x0221	37	34	no
37	0x0141	38	35	no
38	0x0111	39	36	no
39	0x0085	40	37	no
40	0x0049	41	38	no
41	0x0025	42	39	no
42	0x0015	43	40	no
43	0x0009	44	41	no
44	0x0005	45	42	no
45	0x0001	45	43	no

7.14 NEWPRED mode decoding

NEWPRED is a technique in which the reference picture for inter-frame coding is replaced adaptively. This may be according to the upstream messages from the decoder.

7.14.1 Decoder Definition

In NEWPRED mode, decoder may have the additional reference VOP memory to store previously decoded VOPs. When the reference VOP is indicated by vop_id_for_prediction, the decoder select reference VOP from reference VOP memory or additional reference VOP memory to switch the reference picture to the indicated one.

The unit of switching reference VOP, which is NEWPRED segment (NP segment), is indicated by newpred_segment_type. When this flag is "Video Packet", reference VOP shall be switched by Video Packets. In this case, after switching reference VOP, outside of current Video Packet shall be padded in the same way as Unrestricted MC. Each Video Packet boundaries are treated as VOP boundaries, including the treatment of motion vectors which cross those boundaries. The unrestricted motion vector technique is used for Video Packet boundaries. MV prediction and intra AC/DC prediction are restricted to prediction within the Video Packet. When newpred_segment_type flag is "VOP", reference VOP shall be switched by VOPs.

Regardless of newpred_segment_type, the decoder shall not refer to any NP segment which is decoded before the last I_VOP.

7.14.1.1 Position of resynchronization marker

When newpred_segment_type is "Video Packet", it is implied that the positions of Resynchronization Markers are not changed between I_VOPs.

7.14.2 Upstream message

The decoder may return the upstream message to the encoder based on the upstrean syntax defined in 6.2.12. Which type of message is required for the encoder is indicated in requested_upstream_message_type in the VOL header of the downstream data. To return the upstream message is not normative procedure.

7.15 Output of the decoding process

This subclause describes the output of the theoretical model of the decoding process that decodes bitstreams conforming to this part of ISO/IEC 14496.

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

7.15.1 Video data

The output of the video decoding process is a series of VOPs that are normally the input of a display process. The order in which fields or VOPs are output by the decoding process is called the display order, and may be different from the coded order (when B-VOPs are used).

7.15.2 2D Mesh data

The output of the decoding process is a series of one or more mesh object planes. The mesh object planes are normally input to a compositor that maps the texture of a related video object or still texture object onto each mesh. The coded order and the composited order of the mesh object planes are identical.

7.15.3 Face animation parameter data

The output of the decoding process is a sequence of facial animation parameters. They are input to a display process that uses the parameters to animate a face object.

8 Visual-Systems Composition Issues

8.1 Temporal Scalability Composition

Background composition is used in forming the background region for objects at the enhancement layer of temporal scalability when the value of both enhancement_type and background_composition is one. This process is useful when the enhancement VOP corresponds to the partial region of the VOP belonging to the reference layer. In this process, the background of a current enhancement VOP is composed using the previous and the next VOPs in display order belonging to the reference layer.

Figure 8-1 shows the background composition for the current frame at the enhancement layer. The dotted line represents the shape of the selected object at the previous VOP in the reference layer (called "forward shape"). As the object moves, its shape at the next VOP in the reference layer is represented by a broken line (called "backward shape").

For the region outside these shapes, the pixel value from the nearest VOP at the reference layer is used for the composed background. For the region occupied only by the forward shape, the pixel value from the next VOP at the reference layer is used for the composed frame. This area is shown as lightly shaded in Figure 8-1. On the other hand, for the region occupied only by the backward shape, pixel values from the previous VOP in the reference layer are used. This is the area shaded dark in Figure 8-1. For the region where the areas enclosed by these shapes overlap, the pixel value is given by padding from the surrounding area. The pixel value which is outside of the overlapped area should be filled before the padding operation.

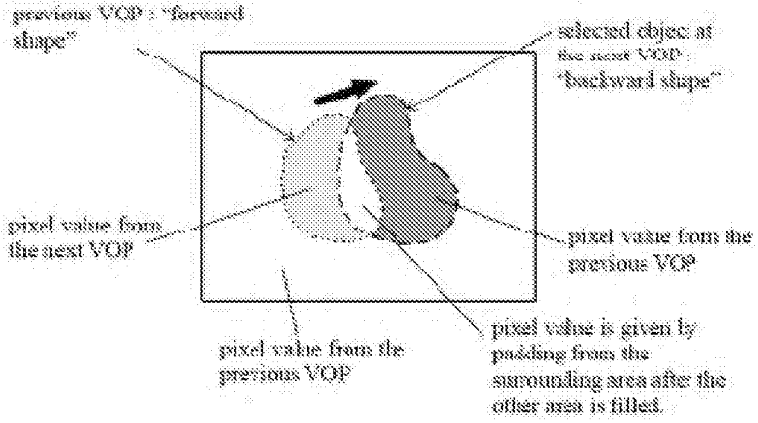

selected object at the previous VOP : "forward shape"
selected objec at the next VOP : "backward shape"
pixel value from the next VOP
pixel value from the previous VOP
pixel value from the previous VOP
pixel value is given by padding from the surrounding area after the other area is filled.

Figure 8-1 – Background composition

The following process is a mathematical description of the background composition method.

```
if s(x,y,ta)=0 and s(x,y,td)=0
    fc(x,y,t) = f(x,y,td)    (|t-ta|>|t-td|)
    fc(x,y,t) = f(x,y,ta)    (otherwise),
if s(x,y,ta)=1 and s(x,y,td)=0
    fc(x,y,t) = f(x,y,td)
if s(x,y,ta)=0 and s(x,y,td)=1
    fc(x,y,t) = f(x,y,ta)
if s(x,y,ta)=1 and s(x,y,td)=1
    The pixel value of fc(x,y,t) is given by repetitive padding from the surrounding area.
```

where

fc composed background
f decoded VOP at the reference layer
s shape information (alpha plane) , 0: transparent, 1: opaque
(x,y) the spatial coordinate
t time of the current VOP
ta time of the previous VOP
td time of the next VOP

Two types of shape information, *s*(*x*, *y*, *ta*) and *s*(*x*, *y*, *td*), are necessary for the background composition. *s*(*x*, *y*, *ta*) is called a "forward shape" and *s*(*x*, *y*, *td*) is called a "backward shape". If f(*x*, *y*, ta) does not exist, the pixel value of fc(*x*, *y*, *t*) for s(*x*, *y*, *td*) = 0 is given by f(*x*, *y*, *td*) and the pixel value of fc(*x*, *y*, *t*) for s(*x*, *y*, *td*) = 1 is given by repetitive padding from the boundary pixel value. If f(*x*, *y*, *td*) does not exist, pixel value of fc(*x*, *y*, *t*) for s(*x*, *y*, *ta*) = 0 is given by f(*x*, *y*, *ta*) and the pixel value of fc(*x*, *y*, *t*) for s(*x*, *y*, *ta*) = 1 is given by repetitive padding from the boundary pixel value.

8.2 Sprite Composition

The static sprite technology enables to encode very efficiently video objects which content is expected not to vary in time along a video sequence. For example, it is particularly well suited to represent backgrounds of scenes (decor, landscapes) or logos.

A static sprite (sometimes referred as mosaic in the literature) is a frame containing spatial information for a single object, obtained by gathering information for this object throughout the sequence in which it appears. A static sprite can be a very large frame: it can correspond for instance to a wide angle view of a panorama.

The ISO/IEC 14496-2 syntax defines a dedicated coding mode to obtain VOPs from static sprites: the so-called "S-VOPs". S-VOPs with sprite_enable == 'static' are extracted from a static sprite using a warping operation consisting in a global spatial transformation driven by few motion parameters (0,2,4, 6 or 8).

For compostion with other VOPs, there are no special rules for S-VOPs with sprite_enable == 'static'. However, it is classical to use S-VOPs with sprite_enable == 'static' as background objects over which "classical" objects are superimposed.

8.3 Mesh Object Composition

A Mesh Object represents the geometry of a sequence of 2D triangular meshes. This data can be used along with separately coded image texture data to render texture-mapped images, e.g., by the composition process as defined in ISO/IEC 14496-1:1999. A Mesh Object stream may be contained in part of a BIFS animation stream, as defined in ISO/IEC 14496-1:1999. In terminals implementing mesh animation functionality using both ISO/IEC 14496-1:1999 and this part of ISO/IEC 14496, the decoded mesh data is used to update the appropriate fields of a BIFS IndexedFaceSet2D node, defined in ISO/IEC 14496-1:1999, for composition purposes. In this case, the appropriate fields of the IndexedFaceSet2D BIFS node are updated as described in the following.

a.) The coordinates of the mesh points (vertices) are obtained from the output of the Mesh Object decoder. The Mesh Object uses a pixel-based local coordinate system with *x*-axis pointing to the right and *y*-axis pointing down. However, ISO/IEC 14496-1:1999 specifies a coordinate system with *y*-axis poining up. Therefore, a simple coordinate transform shall be applied to the *y*-coordinates of mesh points to ensure the proper orientation of the object after composition. The *y*-coordinate y_n of a decoded mesh node point *n* shall be transfomed as follows:

$$Y_n = -y_n \; .$$

where Y_n is the *y*-coordinate of this mesh node point in the coordinate system as specified in ISO/IEC 14496-1:1999. The origin of this object is at the top-left point. The same transform shall be applied to the coordinates of node points of each Mesh Object Plane (MOP).

b.) The coordinate indices are the indices of the mesh points forming faces (triangles) obtained from the output of the Mesh Object decoder. All decoded faces are triangles. The topology of a Mesh Object is constant starting from an intra-coded MOP, throughout a sequence of predictive-coded MOPs (until the next intra-coded MOP); therefore, the coordinate indices shall be updated only for intra-coded MOPs.

c.) Texture coordinates for mapping textures onto the mesh geometry are computed from the decoded node point locations of an intra-coded Mesh Object Plane and its bounding rectangle. Let x_{min}, y_{min} and x_{max}, y_{max} define the bounding rectangle of all node points of an intra-coded MOP. Then the width *w* and height *h* of the texture map shall be:

$$w = \mathrm{ceil}(x_{max}) - \mathrm{floor}(x_{min}) \; ,$$

$$h = \mathrm{ceil}(y_{max}) - \mathrm{floor}(y_{min}) \; .$$

A texture coordinate pair (s_n, t_n) is computed for each node point $p_n = (x_n, y_n)$ as folows:

$$s_n = (x_n - \mathrm{foor}(x_{min}))/w \; ,$$

$$t_n = 1.0 - (y_n - \mathrm{floor}(y_{min}))/h \; .$$

The topology of a Mesh Object is constant starting from an intra-coded MOP, throughout a sequence of predictive-coded MOPs (until the next intra-coded MOP); therefoe, the texture coordinates shall be updated only for intra-coded MOPs.

d.) The texture coordinate indices are identical to the coordinate indices.

8.4 Spatial Scalability composition

Background composition is used in forming the background region for objects at the enhancement layer of spatial scalability when the value of both enhancement_type and background_composition is '1'. This process is useful when the enhancement VOP corresponds to the partial region of the VOP belonging to the reference layer. In this process, current enhancement VOP is composed with up-sampled VOP has same display time in the base layer as back ground image.

In object based spatial scalability, if enhancement_type is "1", use_ref_shape is "1" and background_composition is '1', reference ayer should be up-sampled in the same process described in subclause 7.9.2.6 for background composition. If reference layer has binary shape information, it shall be up-sampled in the same process described in subclause 7.5.4.4. And for the region outside shape of the enhancement VOP, the both texture and shape pixels should be placed on the up-sampled VOP in the reference layer.

In object based spatial scalability, if enhancement_type is "1", use_ref_shape is "0" and background_composition is '1', reference layer shape and texture should be up-samped in the same process described in subclause 7.5.4.4 and 7.9.2.6. And for the shape and texture outside the enhancement VOP, the shape and texture of the up-sampled VOP of the reference layer should be used. In this case, enhancement layer shape and texture are derived by composing the shapes and textures inside enhancement VOP and ouside enhancement VOP. The shape outside enhancement VOP is given by the collocated shape of the up-sampled VOP of the reference layer, and the texture outside enhancement VOP is given by the collocated texture of the up-sampled VOP of the reference layer.

9 Profiles and Levels

NOTE In this part of ISO/IEC 14496 the word "profile" is used as defined below. It should not be confused with other definitions of "profile" and in particular it does not have the meaning that is defined by ISO/IEC ISP.

Profiles and levels provide a means of defining subsets of the syntax and semantics of this part of ISO/IEC 14496 and thereby the decoder capabilities required to decode a particular bitstream. A profile is a defined sub-set of the entire bitstream syntax that is defined by this part of ISO/IEC 14496. A level is a defined set of constraints imposed on parameters in the bitstream. Conformance tests will be carried out against defined profiles at defined levels.

The purpose of defining conformance points in the form of profiles and levels is to facilitate bitstream interchange among different applications. Implementers of this part of ISO/IEC 14496 are encouraged to produce decoders and bitstreams which correspond to those defined conformance regions. The discretely defined profiles and levels are the means of bitstream interchange between applications of this part of ISO/IEC 14496.

In this clause the constrained parts of the defined profiles and levels are described. All syntactic elements and parameter values which are not explicitly constrained may take any of the possible values that are allowed by this part of ISO/IEC 14496. In general, a decoder shall be deemed to be conformant to a given profile at a given level if it is able to properly decode all allowed values of all syntactic elements as specified by that profile at that level.

9.1 Visual Object Types

The following table lists the tools included in each of the Object Types. Bitstreams that represent a particular object corresponding to an Object Type shall not use any of the tools for which the table does not have an 'X'.

Table 9-1 -- Tools for Version 1 Visual Object Types

Visual Tools	Visual Object Types								
	Simple	Core	Main	Simple Scalable	N-bit	Animated 2D Mesh	Basic Animated Texture	Still Scalable Texture	Simple Face
Basic - I-VOP - P-VOP • AC/DC Prediction • 4-MV, Unrestricted MV	X	X	X	X	X	X			
Error resilience • Slice Resynchronization • Data Partitioning • Reversible VLC	X	X	X	X	X	X			
Short Header	X	X	X		X	X			
B-VOP		X	X	X	X	X			
P-VOP with OBMC (Texture)									
Method 1/Method 2 Quantisation		X	X		X	X			
P-VOP based temporal scalability • Rectangular • Arbitrary Shape		X	X		X	X			
Binary Shape		X	X		X	X	X		
Grey Shape			X						
Interlace			X						
Sprite			X						
Temporal Scalability (Rectangular)				X					
Spatial Scalability (Rectangular)				X					
N-Bit					X				
Scalable Still Texture						X	X	X	
2D Dynamic Mesh with uniform topology						X	X		
2D Dynamic Mesh with Delaunay topology						X			
Facial Animation Parameters									X

NOTE 1: "Binary Shape Coding" includes constant alpha.

NOTE 2: The parameters are restricted as follows for the tool "P-VOP based temporal scalability":

- ref_select_code shall be either '00' or '01'.
- vop_coding_type of the enhancement layer VOP shall be either '00' or '01'.
- vop_coding_type of the reference layer VOP shall be either '00' or '01'.
- load_backward_shape shall be '0' and background composition is not performed.

NOTE 3: An 'X' for the "Method 1/Method 2 Quantization" indicates that both quantization methods are supported by the Visual Object Type. Where there is no 'X' the only quantizaton method supported is the Second Inverse Quantization Method (subclause 7.4.4.2).

Table V2 - 39 -- Tools for Version 2 Visual Object Types

Visual Tools	Visual Object Types				
	Advanced Real Time Simple	**Advanced Coding Efficiency**	**Advanced Scalable Texture**	**Core Scalable**	**Simple FBA**
Basic • I-VOP • P-VOP • AC/DC Pediction • 4-MV, Unrestricted MV	X	X		X	
Error resilience • Slice Resynchronization • Data Partitioning • Reversibe VLC	X	X		X	
Short Header	X	X		X	
B-VOP		X		X	
P-VOP with OBMC (Texture)					
Method 1/Method 2 Quantisation		X		X	
P-VOP based temporal scalability • Rectangular • Arbitrary Shape		X		X	
Binary Shape		X		X	
Grey Shape		X			
Interlace		X			
Sprite					
Temporal Scalability (Rectangular)				X	
Spatial Scalability (Rectangular)				X	
N-Bit					
Scalable Still Texture			X		
2D Dynamic Mesh with uniform topology					

2D Dynamic Mesh with Delaunaytopology					
Facial Animation Parameters					X
Body_Animation_Parameters					X
Dynamic Resolution Conversion	X				
NEWPRED	X				
Global Motion Compensation		X			
Quarter-pel Motion Compensation		X			
SA-DCT		X			
Error Reslience for Visual Texture Coding			X		
Wavelet Tiling			X		
Scalable Shape Coding for Still Texture			X		
Object Based Spatial Scalability				X	

9.2 Visual Profiles

Decoders that conform to a Profile shall be able to decode all objects that comply to the Object Types for which the table lists an 'X'.

Table 9-2 -- Version 1 Visual Profiles

	Object Types **Profiles**	**Simple**	**Core**	**Main**	**Simple Scalable**	**N-Bit**	**Animated 2D Mesh**	**Basic Animated Texture**	**Scalable Texture**	**Simple Face**
1.	Simple	X								
2.	Simple Scaleable	X			X					
3.	Core	X	X							
4.	Main	X	X	X					X	
5.	N-Bit	X	X			X				
6.	Hybrid	X	X				X	X	X	X
7.	Basic Animated Texture							X	X	X
8.	Scalable Texture								X	
9.	Simple FA									X

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/SRIDHAR LADAMANE
ANSI Store order #X37445 Downloaded 7/18/2007 3:56:21 PM ET
Single user license only. Copying and networking prohibited.

Table V2 - 40 -- Version 2 Visual Profiles

	Object Types Profiles	Simple	Simple Scalable	Core	Core Scalable	Advanced Real Time Simple	Advanced Coding Efficiency	Advanced Scable Texture	Simple FBA
V2-1.	Advanced Real Time Simple	X				X			
V2-2.	Core Scalable	X	X	X	X				
V2-3.	Advanced Coding Efficiency	X		X			X		
V2-4.	Advanced Core	X		X				X	
V2-5.	Advanced Scalable Texture							X	
V2-6.	Simple FBA								X

NOTE Object Types that are not listed in this table are not decodable by the decoders complying to the Profiles listed in this table.

9.3 Visual Profiles@Levels

9.3.1 Natural Visual

The table that describes the natural visual profiles is given n annex N.

9.3.2 Synthetic Visual

9.3.2.1 Scalable Texture Profile

Table 9-3 -- Scalable texture profile levels

Profile	Level	Default wavelet filter	Max. download filter length	Max. no. of decomposition levels	Typical visual session size^1	Max. Qp value	Max. no. of pixels/ session	VCV decoder rate (equivalent MB's)^2	Max. no. of bitplanes for DC values	Max. VCV buffer size (equivalent MB)^2
Scalable Texture	L3	Float, Integer	ON, 15	10	8192 x8192	12 bits	67108864	262144	18	262144
Scalable Texture	L2	Integer	ON, 15	8	2048 x2048	10 bits	4194304	16384	16	16384
Scalable Texture	L1	Integer	OFF	5	704 x576	8 bits	405504	1584	13	1584

Note: (1) This column is for informative use only. It provides an example configuration of the max. no. of pixels/session.

(2) This still texture VCV model is separate from the gobal video VCV model. An equivalent MB corresponds to 256 pixels.

9.3.2.2 Simple Face Animation Profile

All ISO/IEC 14496-2 facial animation decoders (for all object types) are required to generate at their output a facial model including all the feature points defined in this part of ISO/IEC 14496, even if some of the features points will not be affected by any information received from the encoder.

The Simple Face object is not required to implement the viseme_def/expression_def functionality.

Level 1:

- number of objects: 1,
- The total FAP decode frame-rate in the bitstream shall not exceed 72 Hz,
- The decoder shall be capable of a face model rendering update of at least 15 Hz, and
- Maximum bitrate 16 kbit/s.

Level 2:

- maximum number of objects: 4,
- The FAP decode frame-rate in the bitstream shall not exceed 72 Hz (this means that the FAP decode framerate is to be shared among the objects),
- The decoder shall be capable of rendering the face models with the update rate of at least 60Hz, sharable between faces, with the constraint that the update rate for each individual face is not required to exceed 30Hz, and
- Maximum bitrate 32 kbit/s.

9.3.2.3 Simple FBA Profile

All ISO/IEC 14496-2 FBAdecoders (for all object types) are required to generate at their output a humanoid model including all the feature points and joints defined in this part of ISO/IEC 14496, even if some of the features points and joints will not be affected by any information received from the encoder.

Level 1:

number of objects: 1

The total FBA decode frame-rate in the bitstream shall not exceed 72 Hz

The decoder shall be capable of a humanoid model rendering update of at least 15 Hz

Maximum bitrate 32 kbit/s

The decoder is not required to animate Spine3, Spine4 and Spine5 BAP groups

Level 2:

maximum number of objects: 4

The FBA decode frame-rate in the bitstream shall not exceed 72 Hz (this means that the FBA decode framerate is to be shared among the objects)

The decoder shall be capable of rendering the humanoid models with the update rate of at least 60Hz, sharable between humanoids, with the constraint that the update rate for each individual humanoid is not required to exceed 30Hz

Maximum bitrate 64 kbit/s

Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/[illegible]
Single user licence only. Copying and networking prohibited.

9.3.2.4 Advanced Core Profile and Advanced Scalable Texture Profile

Table V2 - 41 – Advanced core profile levels

Visual Profile	Level	Default wavelet filter	Max. download filter, length	Max. no. of decomposition levels	Typical visual session size[1]	Max. Qp value (bits)	Max. no. of pixels / session[2]	VCV decoder rate (equivalent MB/s)[3]	Max. no. of bitplains for DC values	Max. VCV buffer size (equivalent MB)	Max. STO packet length (bits)	Max. no. of pixels / tile	Max. no. of tile
Advanced Core	L2	Integer	ON, 15	8	8192x8192	10	67108864	262144	16	262144	8192	262144	2048
Advanced Core	L1	Integer	OFF	5	2048x2048	8	4194304	16384	13	16384	4096	65536	1024
Advanced Scalable Texture	L3	Float, Integer	ON, 15	10	8192x8192	12	67108864	262144	18	262144	8192	67108864	4096
Advanced Scalable Texture	L2	Integer	ON, 15	8	2048x2048	10	4194304	16384	16	16384	4096	4194304	2048
Advanced Scalable Texture	L1	Integer	OFF	5	704x576	8	405504	1584	13	1584	2048	405504(= 4xCIF)	1024

Note: (1) This column is for informative use only. It provides an example configuration of the max. no. of pixels/session.

(2) When the number of pixels/session is larger than max. no. of pixels/tile, tiling_disable shall be 0.

(3) This still texture VCV model is separate from the global video VCV model. An equivalent MB corresponds to 256 pixels.

9.3.3 Synthetic/Natural Hybrid Visual

The *Levels* of the Profiles which support both Natural Visual Object Types and Synthetic Visual Object Types are specified by giving bounds for the natural objects and for the synthetic objects. Parameters like bitrate can be combined across natural and synthetic objects.

9.3.3.1 Basic Animated Texture Profile

Level 1 = Simple Facial Animation Profile @ Level 1 + Scalable Texture @ Level 1 + the following restrictions on Basic Animated Texture object types:

Maximum number of Mesh objects (with uniform topology): 4,

Maximum total number of nodes (vertices) in Mesh objects: 480,
(= 4 x nr. Of nodes of a uniform mesh covering a QCIF image with 16x16 pixel elements),

Maximum frame-rate of a Mesh object: 30 Hz, and

Maximum bitrate of Mesh objects: 64 kbit/sec.

Level 2 = Simple Facial Animation Profile @ Level 2 + Scalable Texture @ Level 2 + the following restrictions on Basic Animated Texture object types:

Maximum number of Mesh objects (with uniform topology): 8,

Maximum total number of nodes (vertices) in Mesh objects: 1748,
(= 4 x nr. of nodes of a uniform mesh covering a CIF image with 16x16 pixel elemens),

Maximum frame-rate of a Mesh object: 60 Hz, and

Maximum bitrate of Mesh objects: 128 kbit/sec.

© ISO/IEC 2001 – All rights reserved
Licensed to UNIVERSITY OF TEXAS AT ARLINGTON/PODOLSKY, ADAM/ANE
ANSI Store order X337445 Downloaded 7/18/2007 3:58:21 PM ET
Single user license only. Copying and networking prohibited.

9.3.3.2 Hytrid Profile

Level 1 = Core Visual Profile @ Level 1 + Basic Animated Texture Profile @ Level 1 + the following restrictions on Animated 2D Mesh object types:

Maximum number of Mesh objects (with uniform or Delaunay topology): 4
(= maxinum number of objects in visual session)

Maximum total number of nodes (vertices) in Mesh objects: 480
(= 4 x nr. of nodes of a uniform mesh covering a QCIF image with 16x16 pixel elements)

Maximum frame-rate of a Mesh object: 30 Hz
(= maximum frame-rate of video object)

Maximum bitrate of Mesh objects: 64 kbit/sec.

Level 2 = Core Visual Profile @ Level 2 + Basic Animated Texture Profile @ Level 2 + the following restrictions on Animated 2D Mesh object types:

Maximum number of Mesh objects(with uniform or Delaunay topology): 8
(= maximum number of objects in visual session)

Maximum total number of nodes (vertices) in Mesh objects: 1748
(= 4 x nr. of nodes of a uniform mesh covering a CIF image with 16x16 pixel elements)

Maximum frame-rate of a Mesh object: 60 Hz
(= 2 x the maximum frame-rate of video object)

Maximum bitrate of Mesh objects: 128 kbit/sec.

Annex A

(normative)

Coding transforms

A.1 Discrete cosine transform for video texture

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 coefficients are represented in (*n*+4) bits. The dynamic range of the DCT coefficients 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, with the following modifications :

1) In item (1) of subclause 3.2 of the IEEE specification, the last sentence is replaced by: <<Data sets of 1 000 000 (one million) blocks each should be generated for (L=256, H=255), (L=H=5) and (L=384, H=383). >>

2) The text of subclause 3.3 of the IEEE specification is replaced by : <<For any pixel location, the peak error shall not exceed 2 in magnitude. There is no other accuracy requirement for this test.>>

3) Let F be the set of 4096 blocks Bi[y][x] (i=0..4095) defined as follows :

a) Bi[0][0] = i - 2048

b) Bi[7][7] = 1 if Bi[0][0] is even, Bi[7][7] = 0 if Bi[0][0] is odd

c) All other coefficients Bi[y][x] other than Bi[0][0] and Bi[7][7] are equal to 0

For each block Bi[y][x] that belongs to set F defined above, an IDCT that claims to be compliant shall output a block f[y][x] that as a peak error of 1 or less compared to the reference saturated mathematical integer-number IDCT fii(x,y). In other words, | f[y][x] - fii(x,y)| shall be <= 1 for all x and y.