examined. Moreover, the matching is not necessary on the pixels which are zero and are included in a 3x3 zero window, and they are assumed to be still. Thus, at the expense of some comparisons, the matching is performed on few pixels, and at the expense of examining more than a pair of difference, the search window is small.

Step Three: Regularization Still regions, internal to moving regions, are re-assigned to foreground. The regularization algorithm refines the imposing a priori local connectivity constraints, on both the background and the foreground.

Namely, topological constraints on the size of objects irregularities such as holes, isthmi, gulfs and isles are imposed by means of morphological filtering. Five morphological operations, with circular structuring element, are applied (Table F.1). From a topological point of view, the regularization supports multiple moving regions whose size exceeds the size of the structuring element, corresponding to different moving objects. Finally, a post-processing operation refines the results on the basis of spatial edges.

Introducing user interaction in the segmentation algorithm allows to enhance its capability to extract the shapes of moving objects with accuracy. In fact, to minimize false alarm errors, the objects of interest may be partially missed and in particular the extracted object borders usually do not match the real object borders. To increase the segmentation performance the following user-algorithm interactions are provided:

Table F.1 — Morphological operations

	Radius	Radius	Iteration	Iteration
Closing by Rec.	2	3	5	6
Opening by Rec.	7	12	20	35
Opening	4	6		
Closing	3	6		
Closing by Rec.	10	15	200	250

The user driven threshold setting in order to tune, sequence by sequence and frame by frame, the algorithm capability to select the objects of interest.

The user marked area in which the object of interest is located.



Figure F.19 — Segmentation results comparison: original, automatic, semiautomatic

In this case the application of the automatic segmentation technique can be restricted and optimized within that ROI only, and in most the cases this gives a performance enhancement because parameters critical for revealing the object are estimated on the region where the object is included and not on the entire image where regions with different characteristics are usually present.

© ISO/IEC 2004 – All rights reserved

Google
Exhibit 1042

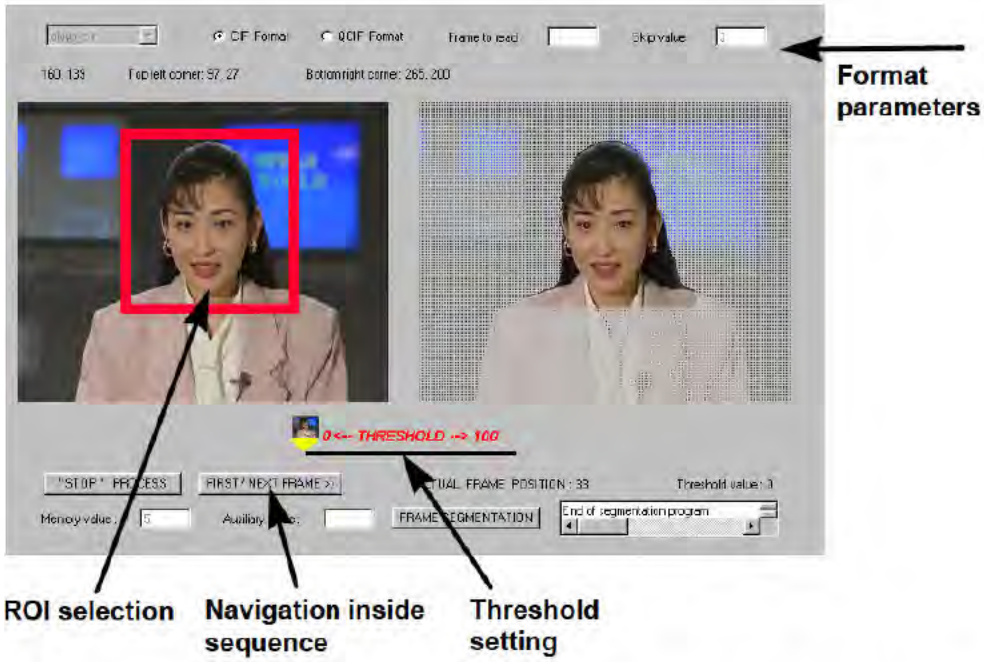


Figure F.20 — Java based Graphical User Interface for semi-automatic segmentation

An example of the obtained results is reported in Figure F.19, in comparison with the result obtained by a completely automatic technique. The enhanced algorithm capability to estimate critical parameters allows to improve its performance in revealing the object shape with accuracy.

A graphical user interface (GUI) allowing easy user interactions with the segmentation process is shown in Figure F.20. Basically, the result of the segmentation could be presented to the user frame by frame, so that he can decide to continue the process with the same parameters or to stop the process, revise the information he previously gave (such as parameters values or ROI selections), and restart the process.

F.1.3 References

[1] T. Aach, A. Kaup, R. Mester, "Statistical model-based change detection in moving video," *Signal Processing*, Vol. 31, No. 2, pp. 165-180, March 1993.

[2] T. Aach, A. Kaup, R. Mester, "Change detection in image sequences using Gibbs random fields: a Bayesian approach," *Proceedings Int. Workshop on Intelligent Signal Processing and Communication Systems*, Sendai, Japan, pp. 56-61, October 1993.

[3] J. Benois, L. Wu, "Joint Contour-based and Motion-based Image Sequences Segmentation for TV Image Coding at Low Bit-rate," *Visual Communications and Image Processing*, Chicago, Illinois, September 1994.

[4] M. Bierling, "Displacement estimation by hierarchical blockmatching," *3rd SPIE Symposium on Visual Communications and Image Processing*, Cambridge, USA, pp. 942-951, November 1988.

[5] P. Bouthemy, E. François, "Motion segmentation and qualitative dynamic scene analysis from an image sequence," *International Journal of Computer Vision*, vol.10, no.2, pp157-182, 1993.

[6] J. Dugelay, H. Sanson, "Differential methods for the identification of 2D and 3D motion models in image sequences," *Signal Processing*, vol.7, pp. 105-127, Sept. 1995.

© ISO/IEC 2004 – All rights reserved

[7] C. Gu, T. Ebrahimi, M. Kunt, "Morphological Spatio-temporal Segmentation for Content-based Video Coding," *International Workshop on Coding Techniques for Very Low Bit-rate Video*, Tokyo, Japan, November 1995.

[8] M. Hötter, R. Thoma, "Image Segmentation based on object oriented mapping parameter estimation," *Signal Processing*, vol. 15, No. 3, pp. 315-334, October 1988.

[9] R. Mech, M. Wollborn, "A Noise Robust Method for Segmentation of Moving Objects in Video Sequences," *International Conference on Acoustic, Speech and Signal*, Munich, Germany, April 1997.

[10] R. Mech, M. Wollborn, "A Noise Robust Method for 2D Shape Estimation of Moving Objects in Video Sequences Considering a Moving Camera," *Workshop on Image Analysis for Multimedia Interactive Services*, Louvain-la-Neuve, Belgium, June 1997.

[11] R. Mech, M. Wollborn, "A Noise Robust Method for 2D Shape Estimation of Moving Objects in Video Sequences Considering a Moving Camera," *Signal Processing: Special Issue on Video Sequence Segmentation for Content-based Processing and Manipulation*, Vol. 66, No. 2, pp. 203-217, April 1998.

[12] A. Neri, S. Colonnese, G. Russo, "Video Sequence Segmentation for Object-based Coders using Higher Order Statistics," *ISCAS* '97, Hongkong, June 1997.

[13] A. Neri, S. Colonnese, G. Russo, "Automatic Moving Objects and Background Segmentation by means of Higher Order Statistics," *IS&T Electronic Imaging* '97 Conference: Visual Communication and Image Processing, San Jose, 8-14 February 1997, SPIE Vol. 3024.

[14] F. Pedersini, A. Sarti, S. Tubaro, "Combined Motion and Edge Analysis for a Layer-based Representation of Image Sequences," *IEEE International Conference on Image Processing*, Lausanne, Switzerland, September 1996.

[15] P. Salembier, M. Pardàs, "Hierarchical Morphological Segmentation for Image Sequence Coding," *IEEE Transactions on Image Processing*, Vol.3, No.5, pp. 639-651, September 1994.

[16] Luc Vincent, Pierre Soille, "Watersheds in digital spaces: an efficient algorithm based on immersion simulations," *IEEE Transactions on PAMI*, Vol.13, No. 6, pp. 583-598, June 1991.

[17] N. T. Watsuji, H. Katata, T. Aono, "Morphological Segmentation with Motion Based Feature Extraction," *International Workshop on Coding Techniques for Very Low Bit-rate Video*, Tokyo, Japan, November 1995.

[18] *Core Experiments on Multifunctional and Advanced Layered Coding Aspects of MPEG-4 Video*, ISO/IEC JTC 1/SC 29/WG 11 N 2176, May 1998.

[19] Jae Gark Choi, Munchurl Kim, Myoung Ho Lee and Chieteuk Ahn, "A user-assisted segmentation method for video object plane generation," *1998 International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC'98)*, Sokcho, Korea, pp. 7 – 10, July 1998.

[20] Jinsuk Kwak, Jun Geun Jeon, Munchurl Kim, Myoung Ho Lee and Chieteuk Ahn, "A semi-automatic video segmentation method for object oriented multimedia applications," *1999 International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC'99)*, Niigata, Japan, pp. 114 – 117, July 13-15, 1999.

[21] Juergen Stauder, Roland Mech, and Joern Ostermann, "Detection of moving Cast Shadows for Object Segmentation," to be published in *IEEE Transactions on Multimedia*, vol. 1, no. 1, 1999.

F.2 Bounding Rectangle of VOP Formation

This clause describes the bounding rectangle of VOP formation process. The formation of the bounding rectangle of VOP is based on the segmented shape information. The following explains the process to achieve the bounding rectangle of a VOP in such a way that the minimum number of macroblocks containing the object will be attained in order to achieve a higher coding efficiency.

© ISO/IEC 2004 – All rights reserved

1. Generate the tightest rectangle whose vertical and horizontal positions of the top-left point are even numbered. In case of interlaced mode, the vertical position of the top-left point of the rectangle is a multiple of 4.

2. If the top left position of this rectangle is not the same as the origin of the image, the following steps have to be performed. Otherwise no further processing is necessary.

3. Form a control macroblock at the top left corner of the tightest rectangle as shown in Figure F.21.

4. Count the number of macroblocks that completely contain the VOP for all the points of which vertical and horizonal positions are even numbered (in case of interlaced mode, all the points of which horizontal positions are even numbered and vertical positions are multiple of 4) of the control macroblock using the following procedure:

   - Generate a bounding rectangle from the control point to the right bottom side of the VOP which consists of multiples of 16x16 blocks.

   - Count the number of macroblocks in this bounding rectangle, which contain at least one object pel. To do so, it would suffice to take into account the boundary pels of a macroblock only.

5. Select the control point that results in the smallest number of non transparent macroblocks for the given object.

6. Extend the top left coordinate of the tightest rectangle generated in Figure .F.21 to the selected control coordinate. This will create a rectangle that completely contains the object but with the minimum number of non transparent macroblocks in it. The VOP horizontal and vertical spatial references are taken directly from the modified top-left coordinate.

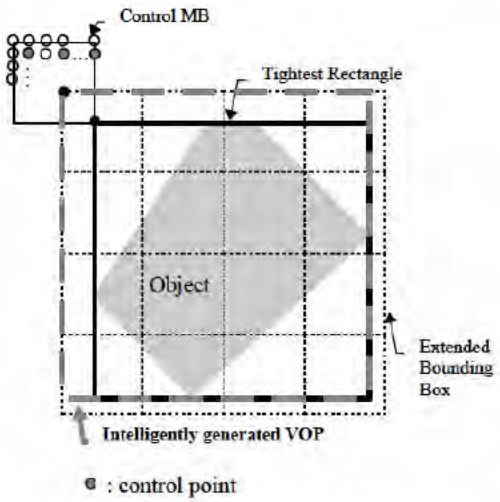


Figure F.21 — Intelligent VOP Formation

F.3 Postprocessing for Coding Noise Reduction

The post-filter consists of deblocking filter and deringing filter. Either one or both of them can be turned on as needed.

© ISO/IEC 2004 – All rights reserved

F.3.1 Deblocking filter

The filter operations are performed along the 8x8 block edges at the decoder as a post-processing operation. Luminance as well as chrominace data is filtered. Figure F.22 shows the block boundaries.

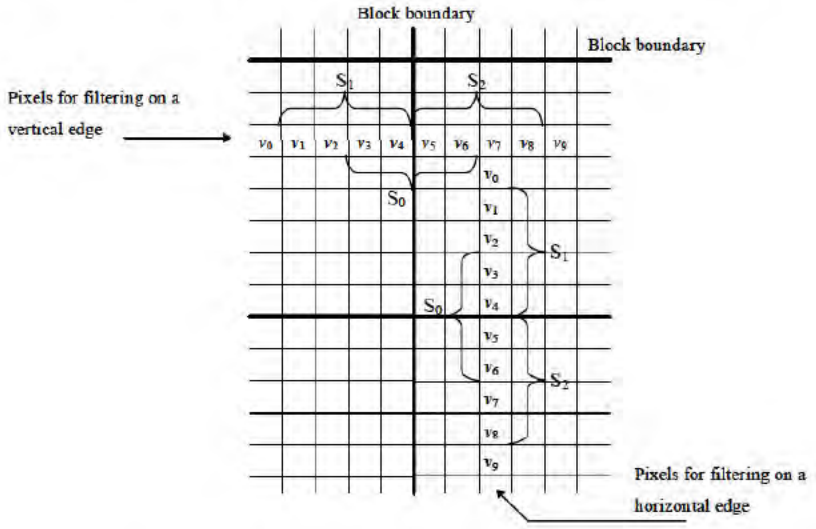


Figure F.22 — Boundary area around block of interest

In the filter operations, two modes are used separately depending on the pixel conditions around a boundary. The following procedure is used to find a very smooth region with blocking artifacts due to small DC offset and to assign it a DC offset mode. In the other case, default mode operations are applied.

eq_cnt = ϕ(v0–v1) + ϕ(v1–v2) + ϕ(v2–v3) + ϕ(v3–v4) + ϕ(v4–v5) + ϕ(v5–v6) + ϕ(v6–v7) + ϕ(v7–v8) + ϕ(v8–v9),

where $\phi(\gamma) = 1$ if $|\gamma| \leq$ THR1 and 0 otherwise.

If (eq_cnt $\geq$ THR2)
DC offset mode is applied,
else
Default mode is applied.

For the simulation, threshold values of THR1 = 2 and THR2 = 6 are used.

In the default mode, a signal adaptive smoothing scheme is applied by differentiating image details at the block discontinuities using the frequency information of neighbor pixel arrays, S_0, S_1, and S_2,. The filtering scheme in default mode is executed by replacing the boundary pixel values v_4 and v_5 with v_4' and v_5' as follows:

$$v_4' = v_4 - d,$$

$$v_5' = v_5 + d,$$

and $$d = \text{CLIP}(5 \cdot (a_{3,0}' - a_{3,0})//8,\ 0,\ (v_4 - v_5)/2) \cdot \delta(|a_{3,0}| < QP)$$

where $$a_{3,0}' = \text{SIGN}(a_{3,0}) \cdot \text{MIN}(|a_{3,0}|, |a_{3,1}|, |a_{3,2}|).$$

Frequency components $a_{3,0}$, $a_{3,1}$, and $a_{3,2}$ can be evaluated from the simple inner product of the approximated DCT kernel [2 -5 5 -2] with the pixel vectors, i.e.,

© ISO/IEC 2004 – All rights reserved

a3,0 = ([2 -5 5 -2] • [v3 v4 v5 v6]T) // 8,
a3,1 = ([2 -5 5 -2] • [v1 v2 v3 v4]T) // 8,
a3,2 = ([2 -5 5 -2] • [v5 v6 v7 v8]T) // 8.

Here CLIP(*x*,*p*,*q*) clips *x* to a value between *p* and *q*; and QP denotes the quantiser scale of the macroblock where pixel v_5 belongs. δ(*condition*)=1 if the "*condition*" is true and 0 otherwise.

In very smooth region, the filtering in the default mode is not good enough to reduce the blocking artifact due to DC offset. So we treat this case in the DC offset mode and apply a stronger smoothing filter as follows :

max = MAX (v1, v2, v3, v4, v5, v6, v7, v8),
min = MIN (v1, v2, v3, v4, v5, v6, v7, v8),
if (|max−min| < 2·QP) {

$$v'_n = \sum_{k=-4}^{4} b_k \cdot p_{n+k}, 1 \le n \le 8$$

$$p_m = \begin{cases} (|v_1 - v_0| < QP) ? v_0 : v_1, if & m < 1 \\ v_m, & if 1 \le m \le 8 \\ (|v_8 - v_9| < QP) ? v_9 : v_8, if & m > 8 \end{cases}$$

$$\{b_k : -4 \le k \le 4\} = \{1,1,2,2,4,2,2,1,1\} // 16$$

}
else
No change will be done.

The above filter operations are applied for all the block boundaries first along the horizontal edges followed by the vertical edges. If a pixel value is changed by the previous filtering operation, the updated pixel value is used for the next filtering.

F.3.2 Deringing filter

This filter comprises three subprocesses; threshold determination, index acquisition and adaptive smoothing. This filter is applied to the pixels on 8x8 block basis. More specifically 8x8 pixels are processed by referencing 10x10 pixels at each block. The following notation is used to specify the six blocks in a macroblock. For instance, block[5] corresponds to the *Cb* block whereas block[*k*] is used as a general representation in the following subclauses.

F.3.2.1 Threshold determination

Firstly, calculate maximum and minimum gray value within a block in the decoded image. Secondary, the threshold denoted by *thr[k]* and the dynamic range of gray scale denoted by *range[k]* are set:

$$thr[k] = (\max imum[k] + \min imum[k] + 1) / 2$$
$$range[k] = \max imum[k] - \min imum[k]$$

An additional process is done only for the luminance blocks. Let *max_range* be the maximum value of the dynamic range among four luminance blocks.

$$\max_range = range[k_{\max}]$$

Then apply the rearrangement as follows.

```
for( k=1 ; k<5 ; k++ ){
    if( range[k] < 32 && max_range > =64 )
        thr[k] = thr[kmax];
    if( max_range<16 )
        thr[k] = 0;
}
```

© ISO/IEC 2004 – All rights reserved

F.3.2.2 Index acquisition

Once the threshold value is determined, the remaining operations are purely 8x8 block basis. Let *rec(h,v)* and *bin(h,v)* be the gray value at coordinates *(h,v)* where *h,v=0,1,2,...,7*, and the corresponding binary index, respectively. Then *bin(h,v)* can be obtained by:

$$bin(h,v) = \begin{cases} 1 & if\ rec(h,v) \geq thr \\ 0 & otherwise \end{cases}$$

Note that *(h,v)* is use to address a pixel in a block, while *(i,j)* is for accessing a pixel in a 3x3 window.

F.3.2.3 Adaptive smoothing

F.3.2.3.1 Adaptive filtering

The figure below is the binary indices in 8x8 block level, whereas practically 10x10 binary indices are calculated to process one 8x8 block.

	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0	1	1
	1	1	0	0	0	0	0	1	1	1
	1	1	1	0	0	0	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	0
	1	1	1	1	1	1	1	1	0	0

Figure F.23 — Example of adaptive filtering and binary index

The filter is applied only if the binary indices in a 3x3 window are all the same, i.e., all "0" indices or all "1" indices. Note 10x10 binary indices are obtained with a single threshold which corresponds to the 8x8 block shown in the above figure, where the shaded region represents the pixels to be filtered.

The filter coefficients used for both intra and non-intra blocks denoted by *coef(i,j)*, where *i,j=-1,0,1*, are:

1	2	1
2	4	2
1	2	1

Figure F.24 — Filter mask for adaptive smoothing

Here the coefficient at the center pixel, i.e., *coef(0,0)*, corresponds to the pixel to be filtered. The filter output *flt'(i,j)* is obtained by:

© ISO/IEC 2004 – All rights reserved

$$flt'(h,v) = \{8 + \sum_{i=-1}^{1} \sum_{j=-1}^{1} coef(i,j) \cdot rec(h+i, v+j)\} // 16$$

F.3.2.3.2 Clipping

The maximum gray level change between the reconstructed pixel and the filtered one is limited according to the quantiser scale, i.e., QP. Let *flt(h,v)* and *flt'(h,v)* be the filtered pixel value and the pixel value before limitation, respectively.

```
if( flt'(h,v) - rec(h,v) > max_diff )
    flt(h,v) = rec(h,v) + max_diff
else if( flt'(h,v) - rec(h,v) < -max_diff )
    flt(h,v) = rec(h,v) - max_diff
else
    flt(h,v) = flt'(h,v)
```

where max_diff=QP/2 for both intra and inetr macroblocks.

F.3.3 Further issues

In order to reduce the number of computations in post-filtering, two kinds of semaphores can be defined: the blocking semaphores and the ringing semaphore. Depending on the blocking semaphores, the horizontal and the vertical deblocking filtering is applied strongly and weakly on the block boundary. If the ringing semaphore (RS) of a current block is "1", deringing filtering is applied. For extracting the semaphores in intra-frame, when only a DC component in the 8x8 inverse quantised coefficients (IQC), the DCT coefficients after inverse quantisation, has a non-zero value, both the horizontal blocking semaphore (HBS) and the vertical blocking semaphore (VBS) of the block are set to "1". When only the coefficients in the top row of the IQC have non-zero values, the VBS is set to "1". When only the coefficients in the far left column of the IQC have non-zero values, the HBS is set to "1". The RS is set to "1" if any non-zero coefficient exists in positions other than a DC component, the first horizontal AC component, and the first vertical AC comonent. Also the semaphores of the inter-frame are calculated from both the residual signal and the semaphores of the reference frame by using the motion vector.

[1] M2723 Y.L. Lee et al.

F.4 Chrominance Decimation and Interpolation Filtering for Interlaced Object Coding

The chrominance decimation filtering and the chrominance interpolation filtering are performed on the basis of the same field in case of interlaced object-based coding during the image format conversion process from 4:2:2 to 4:2:0 and from 4:2:0 to 4:2:2, respectively. In case that filter taps contain only object samples, conventional filtering methods are applied. When filter taps contain both object and background samples, the following decimation and interpolation filtering methods are applied:

Decimation filtering: The source object and background chroma samples are identified from the luminance shape information. Among the decimated chroma samples, the object chroma samples are identified by the field-based chroma subsampling method described in subclause 7.6.1.5. When filter taps contain both of object and background samples at the object boundary, only chroma sample values of object are used for obtaining its decimated chroma sample value. For example, in case of a two-tap filter, one tap is on an object sample and the other is on a background sample, the resulting chroma sample is obtained using only the object chroma sample, not counting on the background sample. In Figure F.25 (a), the source chroma sample C1 is object and C2 is background. The value of the resulting decimated chroma sample C3 is the value of C1.

Interpolation filtering: In the interpolation filtering process which is the inverse of the decimation filtering, object chroma samples are identified among the interpolated chroma samples from the luminance shape information and the values of the interpolated object chroma samples are obtained applying an interpolation filter on input decimated chroma samples. When filter taps include both of decimated object and background chroma samples, only the decimated object chroma samples are counted. For example, in case of a two-tap filter in Figure F.25 (b), two decimated chroma samples, C4 and C5, are used for obtaining the interpolated chroma samples C6 and C7.

© ISO/IEC 2004 – All rights reserved

Suppose that C6 is identified as object and C7 is identified as background from the luminance shape information and the chroma sample C4 is object and C5 is background, then the interpolated chroma sample C6 is obtained as the value of the chroma sample C4. The interpolation for C7 is not necessary because it is background.

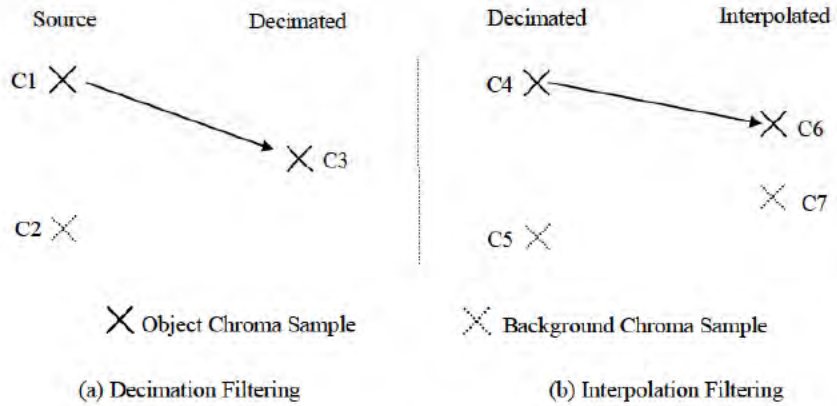


Figure F.25 — Chrominance Decimation and Interpolation Filtering at the Object Boundaries

© ISO/IEC 2004 – All rights reserved

Annex G
(normative)

Profile and level indication and restrictions

This annex specifies the syntax element restrictions and permissible layer combinations.

Table G.1 — FLC table for profile_and_level_indication

Profile/Level	Code
Reserved	00000000
Simple Profile/Level 1	00000001
Simple Profile/Level 2	00000010
Simple Profile/Level 3	00000011
Reserved	00000100 – 00000111
Simple Profile/Level 0	00001000
Reserved	00001001 - 00001111
Simple Scalable Profile/Level 0	00010000
Simple Scalable Profile/Level 1	00010001
Simple Scalable Profile/Level 2	00010010
Reserved	00010011 – 00100000
Core Profile/Level 1	00100001
Core Profile/Level 2	00100010
Reserved	00100011 – 00110001
Main Profile/Level 2	00110010
Main Profile/Level 3	00110011
Main Profile/Level 4	00110100
Reserved	00110101 – 01000001
N-bit Profile/Level 2	01000010
Reserved	01000011 – 01010000
Scalable Texture Profile/Level 1	01010001
Reserved	01010010 – 01100000
Simple Face Animation Profile/Level 1	01100001
Simple Face Animation Profile/Level 2	01100010
Simple FBA Profile/Level 1	01100011
Simple FBA Profile/Level 2	01100100
Reserved	01100101 – 01110000
Basic Animated Texture Profile/Level 1	01110001
Basic Animated Texture Profile/Level 2	01110010
Reserved	01110011 – 10000000
Hybrid Profile/Level 1	10000001
Hybrid Profile/Level 2	10000010
Reserved	10000011 – 10010000
Advanced Real Time Simple Profile/Level 1	10010001
Advanced Real Time Simple Profile/Level 2	10010010
Advanced Real Time Simple Profile/Level 3	10010011

© ISO/IEC 2004 – All rights reserved

Advanced Real Time Simple Profile/Level 4	10010100
Reserved	10010101 – 10100000
Core Scalable Profile/Level1	10100001
Core Scalable Profile/Level2	10100010
Core Scalable Profile/Level3	10100011
Reserved	10100100 – 10110000
Advanced Coding Efficiency Profile/Level 1	10110001
Advanced Coding Efficiency Profile/Level 2	10110010
Advanced Coding Efficiency Profile/Level 3	10110011
Advanced Coding Efficiency Profile/Level 4	10110100
Reserved	10110101 – 11000000
Advanced Core Profile/Level 1	11000001
Advanced Core Profile/Level 2	11000010
Reserved	11000011 – 11010000
Advanced Scalable Texture/Level1	11010001
Advanced Scalable Texture/Level2	11010010
Advanced Scalable Texture/Level3	11010011
Reserved	11010100 – 11100000
Simple Studio Profile/Level 1	11100001
Simple Studio Profile/Level 2	11100010
Simple Studio Profile/Level 3	11100011
Simple Studio Profile/Level 4	11100100
Core Studio Profile/Level 1	11100101
Core Studio Profile/Level 2	11100110
Core Studio Profile/Level 3	11100111
Core Studio Profile/Level 4	11101000
Reserved	11101001 – 11101111
Advanced Simple Profile/Level 0	11110000
Advanced Simple Profile/Level 1	11110001
Advanced Simple Profile/Level 2	11110010
Advanced Simple Profile/Level 3	11110011
Advanced Simple Profile/Level 4	11110100
Advanced Simple Profile/Level 5	11110101
Reserved	11110110
Advanced Simple Profile/Level 3b	11110111
Fine Granularity Scalable Profile/Level 0	11111000
Fine Granularity Scalable Profile/Level 1	11111001
Fine Granularity Scalable Profile/Level 2	11111010
Fine Granularity Scalable Profile/Level 3	11111011
Fine Granularity Scalable Profile/Level 4	11111100
Fine Granularity Scalable Profile/Level 5	11111101
Reserved	11111110
Reserved for Escape	11111111

Ed. note (JO): 255=ESCAPE needs to be re-defined by corrigendum due to mismatch with previous systems definition: 255=No Visual Capability required (critical in context of File Format)

© ISO/IEC 2004 – All rights reserved

Table G.2 — possible combination of two tools

Tools		BA	B-VOP	OBMC	Q	ER	SH	BS	GS	IN	NB	TS(B)	TS(E)	SS(B)	SS(E)	SP	ST	RRV	NP	GMC	QMC	SADCT	ERST	WT	SSST	OSS(B)	OSS(E)	MAC	FGS	FGST
Basic • I-VOP • P-VOP • AC/DC Prediction • 4-MV, Unrestricted MV	BA			✓	✓	✓	✓ a)	✓	✓	✓	✓	✓	✓		✓	✓ b)	x	✓	✓	✓	✓	✓	x	x	x	✓	✓	✓	x	x
B-VOP	BV			✓	✓	✓	x	✓	✓	✓	✓	✓	✓	✓	✓	x	x	x	x	✓	✓	✓	x	x	x	✓	✓	✓	x	x
P-VOP with OBMC (Texture)	OBMC				✓	✓	x	✓	✓	✓	✓	✓	✓	✓	✓	D	x	✓	x	✓	✓	✓	x	x	x	✓	✓	✓	x	x
Method 1/Method 2 Quantisation	Q					✓	x	✓	✓	✓	✓	✓	✓	✓	✓	✓	x	✓	✓	✓	✓	✓	x	x	x	✓	✓	✓	x	x
Error resilience	ER						x	✓	✓ d)	✓ e)	✓	✓	x	✓	x	x	x	✓	✓	✓	✓	✓	x	x	x	✓	x	✓ d)	x	x
Short Header	SH							x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Binary Shape (progressive)	BS								x	✓	✓	✓	✓	D	D	✓	H	x	x	✓	✓	✓	x	x	x	✓	✓	✓	x	x
Greyscale Shape	GS									✓	D	D	D	D	D	✓ f)	H	x	x	✓	✓	✓	x	x	x	D	D	✓ g)	x	x
Interlace	IN										✓	D	D	D	D	x	x	x	x	✓	✓	✓	x	x	x	D	D	✓	✓	✓
N-Bit	NB											✓ c)	✓ c)	D	D	x	x	x	x	D	D	D	x	x	x	D	D	D	x	x
Temporal Scalability (Base)	TS(B)															D	x	x	x	✓	✓	✓	x	x	x	D	D	D	x	x
Temporal Scalability (Enhancement)	TS(E)															x	x	x	x	x	D	D	x	x	x	D	D	D	x	x
Spatial Scalability (Base)	SS(B)															D	x	x	x	✓	✓	✓	x	x	x	✓	✓	D	x	x
Spatial Scalability (Enhancement)	SS(E)															x	x	x	x	x	D	D	x	x	x	✓	✓	D	x	x
Sprite	SP																x	x	x	x	x	x	x	x	x	x	x	x	x	x
Still Texture	ST																	x	x	x	x	✓	✓	✓	x	x	x	x	x
Reduced Resolution VOP	DRC																		✓	x	x	x	x	x	x	x	x	x	x	x
NEWPRED	NP																			x	x	x	x	x	x	x	x	x	x	x
Global Motion Compensation	GMC																				✓	✓	x	x	x	x	x	D	x	x
Quarter-pel Motion Compensation	QMC																					✓	x	x	x	x	x	D	x	x
SA-DCT	SADCT																						x	x	x	x	x	D	x	x
Error Resilience for Still Texture Coding	ERST																							✓	✓	x	x	x	x	x
Wavelet Tiling	WT																								✓	x	x	x	x	x
Scalable Arbitrary Shape for Still Texture Coding	SSST																									x	x	x	x	x

© ISO/IEC 2004 – All rights reserved

Object Based Spatial Scalability (Base)	OSS(B)																											D	×	×
Object Based Spatial Scalability (Enhancement)	OSS(E)																											D	×	×
Multiple Auxiliary Component	MAC																												×	×
Fine Granularity Scalability	FGS																													✓
Fine Granularity Scalability Temporal	FGST																													

✓ - This combination is supported by the syntax. (Subject to further restrictions imposed by the definition of visual object types).
× – This combination is not supported by the syntax.

D – This combination is supported by the syntax but not allowed within one Visual Object(). This combination may be supported in the future, if requirements arise for the corresponding combination.

H - supported at a higher syntactic level.

a) Short Header does not support DC/AC Prediction, 4MV nor Unrestricted MV
b) Sprite does not support 4MV nor Unrestricted MV.
c) P-VOP Temporal Scalability only.
d) Grey Scale does not support Data Partitioning nor RVLC
e) Interlace does not support Data Partitioning nor RVLC
f) Grey Scale does not support Low Latency Sprites.
g) MAC is extension of grayscale shape, no real combination

© ISO/IEC 2004 – All rights reserved

Annex H
(informative)

Patent statements

The International Organization for Standardization and the International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this part of ISO/IEC 14496 may involve the use of patents.

ISO and IEC take no position concerning the evidence, validity and scope of these patent rights.

The holders of these patent rights have assured the ISO and IEC that they are willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statements of the holders of these patents right are registered with ISO and IEC. Information may be obtained from the companies listed below.

Attention is drawn to the possibility that some of the elements of this part of ISO/IEC 14496 may be the subject of patent rights other than those identified in this annex. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

H.1 Patent statements for ISO/IEC 14496 Version 1

The table summarises the formal patent statements received and indicates the parts of ISO/IEC 14496 Version 1 to which the statement applies (S: ISO/IEC 14496-1:2001, V: tools listed in Table 9-1, A: ISO/IEC 14496-3:2001, R: ISO/IEC 14496-5:2000, D: ISO/IEC 14496-6:1999). The list includes all organisations that have submitted informal patent statements. However, if no "X" is present, no formal patent statement has yet been received from that organisation.

		S	V	A	R	D
1.	Alcatel	x	x	x	x	x
2.	AT&T					
3.	BBC	x	x	x	x	x
4.	Bosch		x	x	x	
5.	British Telecommunications	x	x	x	x	x
6.	Canon	x	x	x	x	x
7.	CCETT	x	x	x	x	x
8.	Columbia University	x	x	x	x	x
9.	Creative	x		x	x	
10.	CSELT			x		
11.	DEmoGraFX		x		x	
12.	DirecTV	x	x	x		
13.	Dolby	x	x	x	x	x
14.	EPFL	x	x	x		
15.	ETRI	x	x	x	x	x
16.	FhG	x	x	x	x	x
17.	France Telecom	x	x	x	x	x
18.	Fujitsu Limited	x	x	x	x	x
19.	GC Technology Corporation	x	x	x		
20.	General Instrument		x		x	

© ISO/IEC 2004 – All rights reserved

		S	V	A	R	D
21.	Hitachi	x	x	x	x	x
22.	Hyundai	x	x	x	x	x
23.	IBM	x	x		x	
24.	Institut für Rundfunktechnik	x	x	x		x
25.	InterTrust					
26.	JVC	x	x	x	x	x
27.	KDD Corporation	x	x			
28.	KPN	x	x	x	x	x
29.	LG Semicon					
30.	Lucent					
31.	Matsushita	x	x	x	x	x
32.	Microsoft	x	x	x	x	x
33.	MIT					
34.	Mitsubishi	x	x	x	x	
35.	Motorola		x		x	
36.	NEC Corporation	x	x	x	x	x
37.	NHK	x	x	x	x	x
38.	Nokia		x	x	x	
39.	NTT	x	x	x	x	x
40.	OKI	x	x	x	x	x
41.	Philips	x	x	x	x	x
42.	PictureTel Corporation		x		x	
43.	Rockwell	x	x	x	x	x
44.	Samsung	x	x	x		
45.	Sarnoff	x	x	x	x	x
46.	Scientific Atlanta	x	x	x	x	x
47.	Sharp	x	x	x	x	x
48.	Siemens	x	x	x		
49.	Sony	x	x	x	x	x
50.	Telenor	x	x	x	x	x
51.	Teltec DCU		x		x	
52.	Texas Instruments					
53.	Thomson	x	x	x		
54.	Toshiba		x			
55.	Unisearch Ltd.		x		x	
56.	Vector Vision		x			

H.2 Patent statements for the extensions provided in ISO/IEC 14496 Version 2

The table summarises the formal patent statements received and indicates the parts of the extensions provided in ISO/IEC 14496 Version 2 to which the statement applies (S: ISO/IEC 14496-1:2001, V: tools listed in Figure 7-83 and not listed in Table 9-1, A: ISO/IEC 14496-3:2001, R: ISO/IEC 14496-5:2000/Amd.1:2000, D: ISO/IEC 14496-6:1999/Amd.1:2000). The list includes all organisations that have submitted informal patent statements. However, if no "X" is present, no formal patent statement has yet been received from that organisation.

© ISO/IEC 2004 – All rights reserved

		S	V	A	R	D
1.	Apple	x			x	
2.	AT&T	x	x	x		
3.	British Telecommunications					
4.	Bosch	x	x	x	x	x
5.	CCETT	x	x	x	x	x
6.	Columbia Innovation Enterprise	x	x	x	x	x
7.	DemoGraFX	x	x	x	x	x
8.	DirecTV	x	x	x		
9.	Dolby	x	x	x	x	x
10.	EPFL	x	x		x	
11.	France Telecom	x	x	x	x	x
12.	Fraunhofer			x	x	
13.	Fujitsu		x		x	
14.	Hitachi	x	x	x	x	x
15.	Hyundai	x	x	x	x	x
16.	IBM	x	x	x	x	x
17.	Intertrust					
18.	JVC	x	x	x	x	x
19.	KPN		x			
20.	Lucent					
21.	Matsushita Electric Industrial Co., Ltd.	x	x	x	x	x
22.	Microsoft	x	x	x	x	x
23.	Mitsubishi	x	x	x	x	x
24.	NEC	x	x	x	x	x
25.	NHK	x	x	x	x	x
26.	Nokia	x		x	x	
27.	NTT		x			
28.	NTT Mobile Communication Networks			x		
29.	OKI	x	x	x	x	x
30.	Optibase	x			x	
31.	Philips					
32.	Samsung	x	x	x	x	
33.	Sarnoff	x	x	x	x	x
34.	Sharp	x	x	x	x	x
35.	Siemens	x	x	x	x	x
36.	Sony	x	x	x	x	x
37.	Sun	x				
38.	Thomson	x	x	x	x	x
39.	Toshiba		x			

H.3 Patent statements for the extensions provided in ISO/IEC 14496-2:2001/AMD1

The table summarises the formal patent statements received and indicates the extensions provided in ISO/IEC 14496-2:2001/AMD1 to which the statement applies (V: tools listed in Table 9-3 and not listed in Table 9-2 and not listed in Table 9-1, R: ISO/IEC 14496-5:2000/AMD2). The list includes all organisations that have submitted informal patent statements. However, if no "X" is present, no formal patent statement has yet been received from that organisation.

© ISO/IEC 2004 – All rights reserved

			V		R	
40.	Bosch		x		x	
41.	NHK		x		x	
42.	Sony		x		x	

H.4 Patent statements for the extensions provided in ISO/IEC 14496-2/AMD2

The table summarises the formal patent statements received and indicates the extensions provided in ISO/IEC 14496-2:2001/AMD2 to which the statement applies (V: tools listed in Table 9-4 and not listed in Table 7-36 and not listed in Table 9-2 and not listed in Table 9-1, R: ISO/IEC 14496-5:2000/AMD1). The list includes all organisations that have submitted informal patent statements. However, if no "X" is present, no formal patent statement has yet been received from that organisation.

			V		R	
1.	Bosch		x		x	
2.	Microsoft		x		x	
3.	Optivision		x		x	
4.	Philips		x		x	
5.	Vector Vision		x		x	
6.	WebCast Technologies		x		x	

H.5 Patent statements for ISO/IEC 14496-2:2001/AMD3

The table summarises the formal patent statements received and indicates the parts of ISO/IEC 14496-2:2001/AMD3 to which the statement applies (V: visual tools in this amendment, R: ISO/IEC 14496-5:2001). The list includes all organisations that have submitted informal patent statements. However, if no "X" is present, no formal patent statement has yet been received from that organisation.

		V	R
57.	PacketVideo	x	x
58.	Envivio	x	x
59.	Philips	x	x

© ISO/IEC 2004 – All rights reserved

Annex I
(informative)

Encoder Complexity Reduction Based on Intelligent Pre-Quantisation

I.1 Introduction

This subclause describes an encoder complexity reduction scheme. The proposed algorithm is aimed at reducing computations by a significant amount at the encoder. It is based on extracting a low cost feature (SAE) from the motion compensated residual blocks, and using this feature and an intelligently derived threshold value to detect "zero blocks" (blocks have all zero quantised DCT coefficients). Once a "zero block" is detected, we can skip DCT, quantisation, and corresponding IDCT and inverse quantisation for this block to reduce the encoder computation.

Most of the existing video coding standards (MPEG-1, MPEG-2, MPEG-4, H.261 and H.263) are based on motion estimation, DCT, quantisation, and entropy coding. The core of these algorithms is a differential loop where a displaced frame difference is calculated based on the motion of the current frame from the immediate past frame. This displaced frame difference is then quantised and entropy coded via a variable length code. In the end these variable length codes, along with the codes which contain the header information conforming to the syntax of a particular algorithm, are transmitted over the channel. Motion estimation is the process by which we calculate the displacement of one frame in a sequence with respect to the previous frame. By calculating the displacement on a block basis, we can compute a displaced frame difference that is easier to code, thereby reducing the temporal redundancy. In the next step, DCT of individual blocks (8x8 pixels) helps in removing the spatial redundancy by bringing down the most relevant information into the lower most coefficients in the DCT domain. After having obtained a block of DCT coefficients that contain the energy of the displaced blocks, quantisation of these blocks is done. Quantisation is a uniform quantisation with step size varying from 2 to 62. It is implemented as a division, or as a table look up operation for a fixed-point implementation. Lastly, having obtained the quantised coefficients, entropy coding is done to further reduce the overall coding rate. Traditionally most video coding standards use Huffman coding for entropy coding.

In terms of computations motion estimation is the costliest module. Typically, it accounts for more than half of the MIPs at the encoder. For this reason, most of the video encoding solutions prefers to do motion estimation either by some dedicated hardware, or by some fast sub-optimal software scheme. DCT and IDCT, which is needed in the encoder for reconstructing the quantised frame for motion estimation, comprise the next most costly part of the encoder. Then follows quantisation and Huffman encoding. In very low bit rate applications, after the quantisation of the motion compensated DCT blocks, many of the blocks have all of their coefficients as zero. It is observed that a majority of the blocks are zero after quantisation. This means that if we can detect a priori which blocks are being zeroed out after quantisation, we can forego DCT, quantisation, inverse quantisation and IDCT for those blocks. This would translate to a great amount of savings in computations at the encoder. In the following, we outline a method to achieve this goal with no loss in the perceived quality of the coded sequence.

I.2 Feature Selection and Pre-quantisation

We begin with identifying some key features of the motion compensated blocks that will help in classifying them into zero and non-zero blocks. A good feature should allow us to classify the blocks very accurately and at the same time it should be relatively easy to compute too. Using a complicated classification scheme defeats our main goal of reducing the overall computations. We consider several features and rank them in the order of their performance.

Most of the features we select are easy to compute and are based on first or second ordered moments of the motion compensated blocks. We consider the following features:

© ISO/IEC 2004 – All rights reserved

SAE: mean absolute error

$$SAE = \sum_{i=1}^{8}\sum_{j=1}^{8}|f(i,j)|\ ,\ \text{where } f(i,j) \text{ is the pel value at } (i,j)\text{ - th location}$$

MSE: mean square error

$$MSE = \sum_{i=1}^{8}\sum_{j=1}^{8}(f(i,j))^2\ ,\ \text{where } f(i,j) \text{ is the pel value at } (i,j)\text{ - th location}$$

MAD: mean absolute difference

$$MAD = \sum_{i=1}^{8}\sum_{j=1}^{8}|f(i,j) - \mu|\ \text{where } f(i,j) \text{ is the pel value at } (i,j) \text{ - th location } \&\mu \text{ is the mean of the block}$$

VAR: block variance

$$VAR = \sum_{i=1}^{8}\sum_{j=1}^{8}(f(i,j)-\mu)^2\ ,\ \text{where } f(i,j) \text{is the pel value at } (i,j)\text{ - th location } \&\mu \text{ is the mean of the block}$$

MAX: maximum value

$$MAX = \max(f(i,j))\ \ i,j \in (1,8)$$

We use each of the above features in classifying the motion compensated blocks as *zero or one*. A *zero block is* defined as a block with all zero quantised DCT coefficients and a one block as a block with at least one non-zero quantised DCT coefficient. For doing the classification, we select one of the above features and use a corresponding (experimentally determined) threshold to decide whether or not this block should be classified as zero. Each time we classify a block as zero, we would be zeroing out the block to begin with and would not be doing DCT and quantisation, and consequently IDCT and inverse quantisation. This translates to a lot of savings in computations. Next we define some terminology which would help us in ranking the performance of different features and in quantifying the degradation in performance while using any such feature based classification scheme.

As is the case with any multiple hypotheses problem, there are two types of errors associated with this feature based classification scheme. Each time we use a feature to do the above classification, there are instances where we classify a block as zero, when it should have been coded instead (one). We call this error as false alarm (FA). The second type of error we encounter in such a classification scheme is when we classify a block to be one when in reality it should have been zero. We call this as misdetection (MD). We formally define a quantitative measure of false alarm and misdetection as follows:

$$FA_{rate} = \frac{FA}{N1}\ ,\ MD_{rate} = \frac{MD}{N0}\ ,$$

where FA is the number of blocks classified as zero when they actually should have been one, MD is the number of blocks classified as one when they actually should have been zero, N1 is the number of true zero blocks, and N0 is the number of true one blocks. We compare the effectiveness of different features by plotting false alarm (FA) vs. misdetection (MD) curves resulting from using the above features in coding a video sequence. Ideally we would want a feature to give us a low false alarm and a low misdetection.

From our experiments we conclude that SAE and MSE have the best performance (lowest false alaram rate and lowest misdetection rate), MAX has the worst performance (highest curve), and MAD, VAR are in between. Since SAE is computationally less expensive to calculate than MSE, we will use SAE as our major feature in classifying the blocks. Also, since MAX is very easy to compute, and it also promises to catch some edge information in a block, we will also investigate using it further.

© ISO/IEC 2004 – All rights reserved

I.3 Model Verification and Threshold Setting

We apply the classification schemes based on above features on a wide range of video sequences. Since motion estimation acts like a normalization process, the statistics of residue blocks is expected to be similar. We verify this fact by encoding several typical sequences. We observe that the curves for misdetection rate and false alarm rate are similar for all sequences. In other words, the feature classification scheme is quite robust to the variations in the input sequence.

Based on experiments, we find SAE gives best result in terms of false alarm and misdetection rate. Here we will find an optimal threshold that results in negligible number of false alarm blocks and at the same time small number of misdetection blocks. Negligible false alarm rate means only very few true one blocks will be misclassified as zero blocks, therefore, the quality degradation will be totally insignificant. Small misdetection rate means most true zero blocks will be classified as zero blocks, and these blocks will skip DCT and quantisation, therefore, computational complexity will be reduced. Here we first calculate the upper bound of SAE that will guarantee zero false alarm rate.

The forward DCT is:

$$F(u,v) = \frac{1}{4} C(u)\, C(v) \sum_{i=0}^{7} \sum_{j=0}^{7} \cos((2i+1)u\frac{\pi}{16}) \cos((2j+1)v\frac{\pi}{16}) f(i,j)$$

$$\Rightarrow |F(u,v)| \le \frac{1}{4} \sum_{i=0}^{7} \sum_{j=0}^{7} |f(i,j)| = 1/4\, SAE, \text{ for } u, v \neq 0$$

$$= 1/8\, SAE, \text{ for } u, v = 0$$

I.3.1 H.263 Quantiser Scale

When using the H.263 quantiser scale, the quantisation for INTER mode blocks AC/DC coefficients can be expressed as:

$$coeff(u,v) = sign(F(u,v)) * \frac{|F(u,v)| - QP/2}{2*QP}$$

In this case, to guarantee all coeff(u, v)=0, we have

$$|F(u,v)| - QP/2 < 2*QP$$
$$|F(u,v)| < 5/2 * QP$$

From above equations we can guarantee that the DC componet of the block would be zero if the following condition is satisfied:

$$\frac{1}{8} SAE < 5/2 * QP$$
$$SAE < 20 * QP$$

I.3.2 MPEG-4 Quantiser Scale

When using the MPEG-4 quantiser scale with the MPEG-4 non-intra quantisation matrix, the quantisation for INTER mode blocks AC/DC coefficients can be expressed as:

$$coeff(u,v) = sign(F(u,v)) * \frac{(16 * |F(u,v)|)//w[u][v]}{2*QP}.$$

© ISO/IEC 2004 – All rights reserved

where w is the non-intra quantisation matrix. In this case, to guarantee all coeff(u, v)=0, we have

$$|F(u,v)| < 2*QP$$

From above equations we can guarantee that the DC componet of the block would be zero if the following condition is satisfied:

$$\frac{1}{8}SAE < 2*QP$$
$$SAE < 16*QP$$

Since the DC component is the most important and more likely to have non-zero value if a block has any non-zero components, this threshold value guarantees no quality degradation when applying the pre-quantisation scheme and receiving coding complexity reduction.

To conclude, the threshold is set to 20*QP when H263 quantiser scale is used, and 16*QP when MPEG-4 quantiser scale with default quantisation matrix is used. These thresholds will guarantee accurate prediction of zero coefficients both for AC as well as the DC coefficients.

© ISO/IEC 2004 – All rights reserved

Annex J
(normative)

View dependent object scalability

J.1 Introduction

Coding of View-Dependent Scalability (VDS) parameters for texture can provide for efficient incremental decoding of 3D images (e.g. 2D texture mapped onto a gridded 3D mesh such as terrain). Corresponding tools from ISO/IEC 14496-1:2001 and this part of ISO/IEC 14496 are used in conjunction with downstream and upstream channels of a decoding terminal. The combined capabilities provide the means for an encoder to react to a stream of viewpoint information received from a terminal. The encoder transmits a series of coded textures optimized for the viewing conditions which can be applied in the rendering of textured 3D meshes by the receiving terminal. Each encoded view-dependent texture (initial texture and incremental updates) typically corresponds to a specific 3D view in the user's viewpoint that is first transmitted from the receiving terminal.

A tool defined in ISO/IEC 14496-1 transmits 3D viewpoint parameters in the upstream channel back to the encoder. The encoder's response is a frequency-selective, view-dependent update of DCT coefficients for the 2D texture (based upon view-dependent projection of the 2D texture in 3D) back to the receiving terminal, along the downstream channel, for decoding by a Visual DCT tool at the receiving terminal. This bilateral communication supports interactive server-based refinement of texture for low-bandwidth transmissions to a decoding terminal that renders the texture in 3D for a user controlling the viewpoint movement. A gain in texture transmission efficiency is traded for longer closed-loop latency in the rendering of the textures in 3D. The terminal coordinates inbound texture updates with local 3D renderings, accounting for network delays so that texture cached in the terminal matches each rendered 3D view.

A method to obtain an optimal coding of 3D data is to take into account the viewing position in order to transmit only the most visible information. This approach reduces greatly the transmission delay, in comparison to transmitting all scene texture that might be viewable in 3D from the encoding database server to the decoder. At a given time, only the most important information is sent, depending on object geometry and viewpoint displacement. This technique allows the data to be streamed across a network, given that a upstream channel is available for sending the new viewing conditions to the remote database. This principle is applied to the texture data to be mapped on a 3D grid mesh. The mesh is first downloaded into the memory of the decoder using the appropriate BIFS node, and then the DCT coefficients of the texture image are updated by taking into account the viewing parameters, i.e. the field of view, the distance and the direction to the viewpoint.

J.2 Decoding Process of a View-Dependent Object

This clause explains the process for decoding the texture data using the VDS parameters. In order to determine which of the DCT coefficients are to be updated, a "mask", which is a simple binary image, shall be computed. The first step is to determine the viewing parameters obtained from the texture-mesh composition procedure that drives 3D rendering in the user's decoding terminal. These parameters are used to construct the DCT mask corresponding to the first viewpoint of the session (VD mask). This mask is then updated with differential masks, built with the new viewing parameters that allow the texture image to be streamed. The bitstream syntax for view parameters and incremental transmission of DCT coefficients is given elsewhere in ISO/IEC 14496-1:2001 and this part of ISO/IEC 14496.

J.2.1 General Decoding Scheme

The following subclauses outline the overall process for the decoder and encoder to accomplish the VDS functionalities.

© ISO/IEC 2004 – All rights reserved

J.2.1.1 View-dependent parameters computation

The VDS parameters (α and β angles, distance d for each cell) shall be computed using the geometrical parameters (Mesh, Viewpoint, Aimpoint, Rendering window). These parameters shall be computed for each cell of the grid mesh.

J.2.1.2 VD mask computation

For each 8x8 block of texture elements within a 3D mesh cell, the locations of the visible DCT coefficients inside the DCT block shall be computed using α and β angles, and the distance d defined for each cell relative to the viewpoint. The result shall be put in a binary mask image.

J.2.1.3 Differential mask computation

With the knowledge of which DCT coefficients have already been received (Binary mask buffered image) and which DCT coefficients are necessary for the current viewing conditions (Binary VD mask image), the new DCT coefficients shall be determined (Binary Differential mask image) as described in subclause J.2.4.

J.2.1.4 DCT coefficients decoding

The Video Intra bitstream, in the downstream channel, shall be decoded by the receiver terminal to obtain the DCT coefficients (DCT image). The decoding procedure is described in subclause J.2.5.

J.2.1.5 Texture update

The current DCT buffer in the receiver terminal shall be updated according to the Differential mask, using the received DCT image. The new received DCT coefficients shall be added to the buffered DCT image.

J.2.1.6 IDCT

The Inverse DCT of the updated DCT image shall computed, as specified in subclause J.2.7, to obtain the final texture.

J.2.1.7 Rendering

The texture is mapped onto the 3D mesh and the rendering of the scene is done, taking into account the mesh and the viewing conditions. This part of the procedure is outside the scope of this part of ISO/IEC 14496.

© ISO/IEC 2004 – All rights reserved

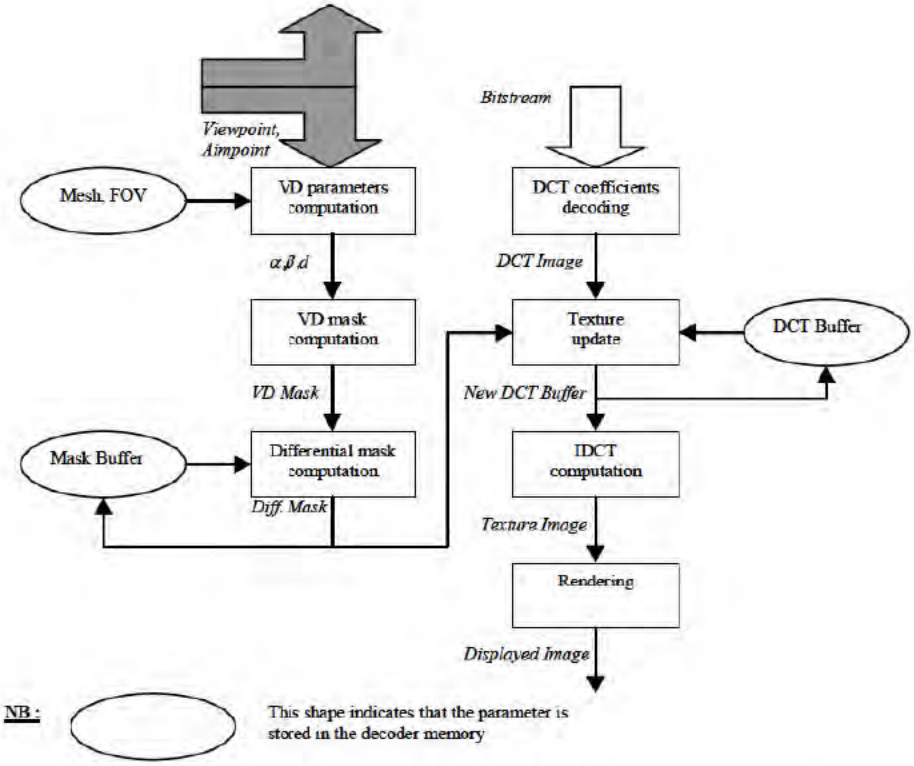


Figure J.1— General Decoding Scheme of a View-Dependent Object

J.2.2 Computation of the View-Dependent Scalability parameters

The VDS parameters shall be computed for each cell of the grid mesh. The mesh may either be a quadrilateral or a triangular mesh. The number of cells in each dimension shall be equal to the texture size divided by 8.

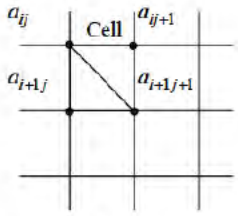


Figure J.2 — Mesh cell

J.2.2.1 Distance criterion:

$$Rd = \frac{1}{u}$$

u is the distance between viewpoint and Cell center: $u = \|\vec{v} - \vec{c}\|$ with $\vec{c} = \frac{1}{4}(\vec{a}_{ij} + \vec{a}_{i+1j} + \vec{a}_{ij+1} + \vec{a}_{i+1j+1})$ and $\vec{v}$ is the viewpoint vector.

© ISO/IEC 2004 – All rights reserved

J.2.2.2 Rendering criterion:

$$R_r = \frac{p}{q}$$

p is the distance between viewpoint and projection plane normalized to window width. p may be computed using:

$$p = \frac{1}{2\tan(FOV/2)}$$

$$p = \frac{1}{2\tan(FOV/2)}, \text{FOV is the Field Of View}$$

where FOV is the Field of View specified in radians, and q = <TextureWidth>/<WindowWidth> where the texture width is the width of the full texture (1024 for instance) and the WindowWidth is the width of the rendering window.

J.2.2.3 Orientation criteria:

$$Ra = \cos(\alpha)$$
$$Rb = \cos(\beta)$$

The angle between the aiming direction and the normal of the current cell center shall be projected into two planes. These two planes are spans of normal vector $\vec{n}$ of the cell and the cell edges in x and y directions, respectively. Then the angles (α, β) between projected vectors and the normal $\vec{n}$ shall be calculated, respectively.

The angle β is specified as the projection of the angle between $\vec{n}$, the normal of the quad cell, and $\vec{u}$, the aiming direction, onto the plane Π_x that passes through $\vec{g}_x$ and is parallel to $\vec{n}$. Similarly, the angle α is specified as the projection of the same angle onto the plane Π_y that passes through $\vec{g}_y$ and its parallel to $\vec{n}$.

This is illustrated in Figure J.3

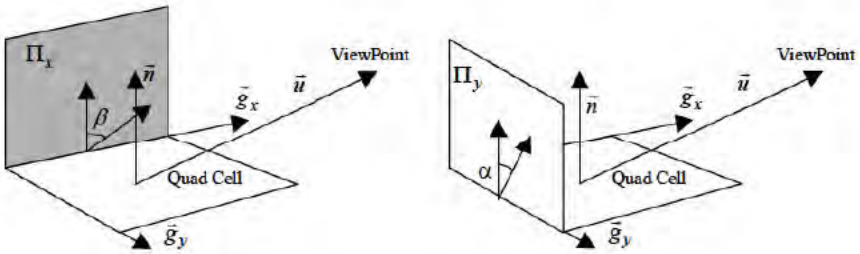


Figure J.3 — Definition of α and β angles

J.2.2.4 Cropping criterion:

Cells that are out of the field of view shall not be transmitted/received: that is, at least one of the 4 vertices which define the cell should all be inside the horizontal and vertical Field Of View (FOV).

The horizontal FOV shall be deduced from the vertical FOV using the screen geometry. The vertical FOV is equal to the FOV. Then the following shall be calculated

$HFOV = A\tan(\frac{w}{h} \cdot \tan(FOV/2))$ where w and h are the width and height, respectively, of the rendered image.

© ISO/IEC 2004 – All rights reserved

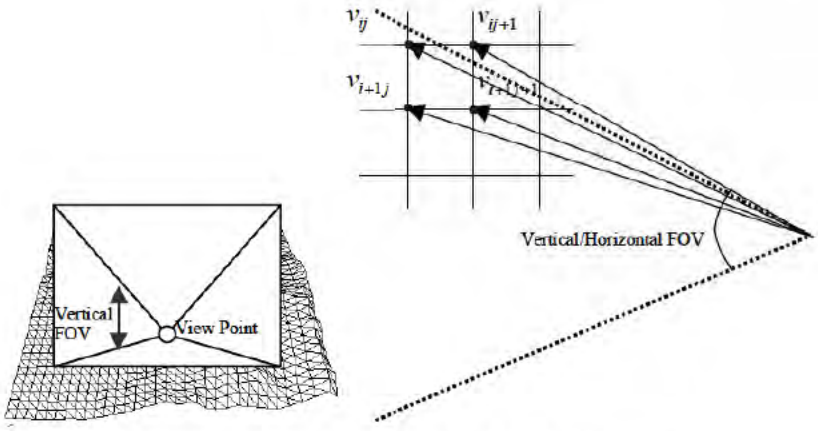


Figure J.4 — Definition of Out of Field of View cells

J.2.3 VD mask computation

The VD mask is a binary image of the same size as the texture image. Each value in the mask shall indicate if the corresponding DCT coefficient is needed (1) or not (0), given the VDS parameters.

For each cell, the following rules shall be applied to fill the corresponding 8x8 block of the VD mask:

- **Use of cropping criterion**: If all the vertices of the cell are out of the field of view, the corresponding 8x8 block of the mask image shall be set to 0.
- **Use of rendering, distance, tilting and rotation criteria**: For each 8x8 block of the mask (corresponding to a quad cell), the 4 criteria mentioned above shall be computed. Two values of the rotation and tilting criteria shall be obtained for a quad cell, but only the higher value of each criterion shall be kept.

Two thresholds, T_X and T_Y, shall be calculated as the product of the three VDS parameters *Rr, Rd, Rb,* and *Rr, Rd, Ra*, respectively, and the value 8. The results shall be bounded to 8. This procedure may be indicated symbolically as follows

$$Tx = Min(8.8 \cdot R_r \cdot R_d \cdot R_b)$$
$$Ty = Min(8.8 \cdot R_r \cdot R_d \cdot R_a)$$

The flag (i,j) of the 8x8 block corresponding to the current cell shall be set to 1 if i < T_X and j < T_Y. The flag shall be set to 0 in all other cases, as illustrated in the figure below.

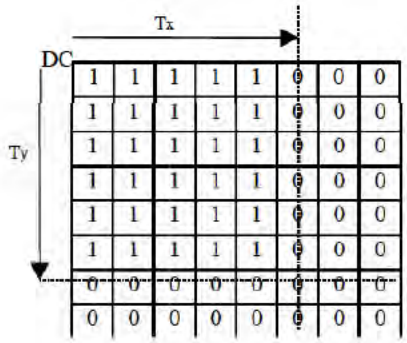


Figure J.5 — VD mask of an 8x8 block using VD parameters

© ISO/IEC 2004 – All rights reserved

J.2.4 Differential mask computation

Once the first image has been received using the previously described filter, less data is necessary to update the texture data for the following frames. (assuming there is a correlation between viewpoint positions). Since this computation is exactly the same for each flag of each cell, it shall be performed directly on the full mask images and not on a cell by cell basis.

If the coefficient has not been already transmitted (buffer mask set to 0) and is needed according to VDS visibility criteria (VD mask set to 1), then the corresponding pixel of the differential mask shall be set to 1. This implies that the texture shall be updated, according to the procedure described in subclause J.2.6.

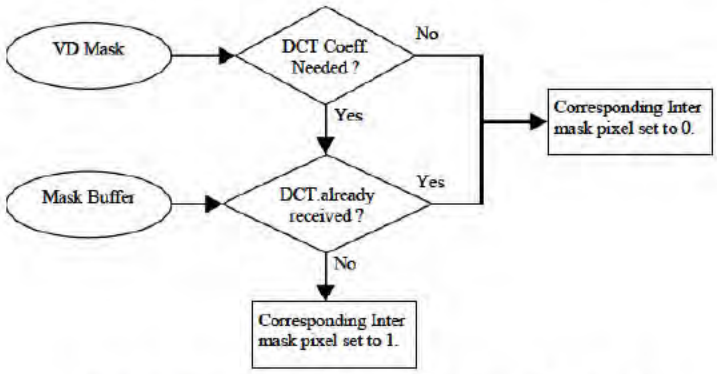


Figure J.6 — Differential mask computation scheme

J.2.5 DCT coefficients decoding

The DCT coefficients shall be decoded using the Video Intra mode, Separated Texture/Motion mode as described in the subclause 7.4.

J.2.6 Texture update

The Differential mask image shall be used to select which DCT coefficients of the buffered texture should be updated using the decoded DCT coefficients.

- Y component

If the Differential mask is set to 0, the corresponding DCT value of the buffer shall be left unchanged, otherwise the value shall be updated with the previously decoded DCT coefficient.

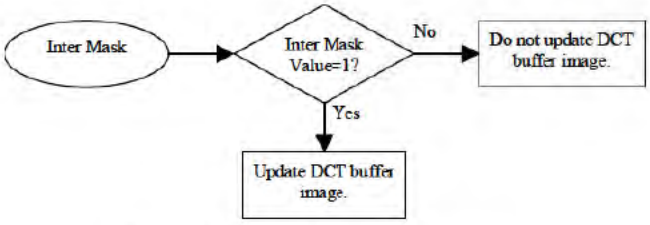


Figure J.7 — Texture update scheme

- U and V component

The texture is coded in 4:2:0 format, as specified in subclause 6.1.3.6, which shall imply that for each chrominance DCT coefficient, 4 Differential mask flags shall be available. The chrominance coefficients shall be received/transmitted if at least 1 of these 4 flags is set to 1.

© ISO/IEC 2004 – All rights reserved

J.2.7 IDCT

The IDCT and de-quantisation shall be performed using the same process as in the Video Intra mode as described in subclause 7.4.

© ISO/IEC 2004 – All rights reserved

Annex K
(normative)

Decoder Configuration Information

K.1 Introduction

This annex identifies the syntax elements defined in the main body of this part of ISO/IEC 14496 that describe the configuration of the visual decoder. This Annex amplifies the information provided in subclause 6.2.1. These elements shall be processed differently if this part of the specification is used jointly with the Systems part, ISO/IEC 14496-1:2001. Instead of conveying the configuration at the beginning of a visual elementary bitstream, it shall be conveyed as part of an Elementary Stream Descriptor that is itself included in an Object Descriptor describing the visual object. This descriptor framework is specified in ISO/IEC 14496-1:2001.

K.2 Description of the set up of a visual decoder (informative)

The process of accessing ISO/IEC 14496 content is specified in ISO/IEC 14496-1:2001. It is summarized here and visualized in Figure L.1 to outline the processing of decoder configuration information in an ISO/IEC 14496 terminal. This description assumes that all elementary streams are accessible and solely the problem of identifying them is to be solved.

The content access procedure starts from an Initial Object Descriptor that may be made available through means that are not defined in this part of ISO/IEC 14496, like a http or ftp link on an HTML page, a descriptor in an ISO/IEC 13818-1 Transport Stream, or some H.245 signaling, etc. Note that this may need standardisation by the responsible group.

This Initial Object Descriptor contains pointers at least to a scene description stream and an object descriptor stream. The scene description stream conveys the time variant spatiotemporal layout of the scene. For each streaming media object incorporated in the scene, an Object Descriptor exists that describes the set of streams associated to this media object. The set of Object Descriptors is conveyed in a separate stream, in order to distinguish scene description from the description of the streaming resources.

Both the scene description stream and the object descriptor stream allow for time stamped updates of the scene description and the object descriptors, respectively. Due to the time stamps it is always known at which point in time, or from which point in time onwards a data item, called Access Unit, is valid.

The Object Descriptor associated to a given visual object is identified by its ObjectDescriptor_ID. Each visual object may require more than one elementary stream (ES) that convey its coded representation, especially if any form of scaleability is used. Each of these streams is described by an ES_Descriptor. This description contains a unique label for the stream, the ES_Id, and, among others, the DecoderSpecificInfo structure that is of concern for the purpose of this Annex.

© ISO/IEC 2004 – All rights reserved

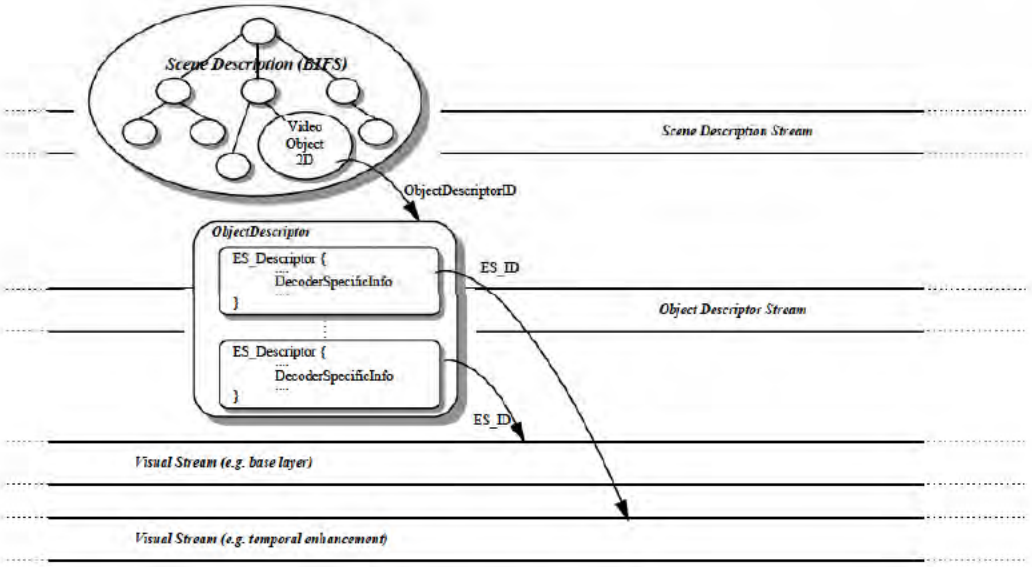


Figure K.1 — Visual decoder setup

K.2.1 Processing of decoder configuration information

After the retrieval of the Object Descriptor for a media object a decoder for the visual stream(s) is instantiated, connected to the stream(s) and initialised with the data found in ES_Descriptor.DecConfigDescr.DecSpecificInfo for each stream. Subsequently, in a random access scenario, the decoder is expected to search forward to the next random access point, while in a client-server or local storage scenario, data in the visual stream may already be aligned in a way that the first data arriving in the visual stream corresponds to a random access point.

The difference between a visual-only scenario, as specified in the main body of this part of ISO/IEC 14496, and an integrated application using both ISO/IEC 14496-1:2001 and this part of ISO/IEC 14496 is visualized in a figure. Figure K.2 shows the Visual-only approach with configuration information at the beginning of a bitstream and optionally repeated thereafter to enable random access. Figure K.3 with the integrated Systems and Visual approach shows the plain bitstreams with the configuration information extracted into the object descriptors. In this case the object descriptors will be repeated if random access is desired. The Access Units shown in the figure correspond to VOPs in the case of visual media streams.

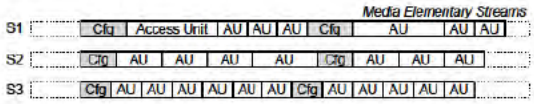


Figure K.2 — Visual-only scenario

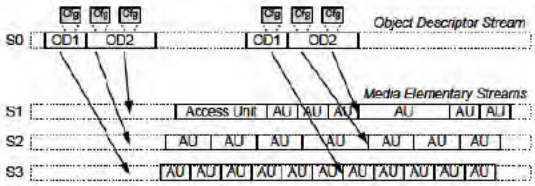


Figure K.3 — Integrated Systems and Visual approach

© ISO/IEC 2004 – All rights reserved

K.3 Specification of decoder configuration information

The decoder configuration information for a visual elementary stream is given by a concatenation of those syntax elements that precede the actual encoded data, i. e., that form the ‚stream header' according to the syntax specification in subclause 6.2. Those syntax elements are identified separately for each type of visual object in the subsequent subclauses. The syntax elements that are conveyed as decoder configuration information shall not be present in the visual elementary stream itself. Furthermore, the syntax definition in subclause 6.2 is constrained to clarify that an elementary stream may not contain a concatenation of, for example, multiple VisualObject() structures or multiple VideoObjectLayer() structures.

The decoder configuration information shall be conveyed in the DecConfigDescr.DecSpecificInfo in field of the respective ES_Descriptor and passed to the decoder before any data of the visual elementary stream itself.

VisualObjectSequence() is transmitted explicitly in each DecConfigDescr.DecSpecificInfo of an Object Descriptor.

Identification and priority of objects is signaled for all kinds of audiovisual objects in the ES_Descriptor. This signalling overrides the value of VisualObject.is_visual_object_identifier and VideoObjectLayer.is_object_layer_identifier.

VisualObject() carries information describing all the elementary streams associated with an object and is carried in a container associated with all the layers of this object.

VideoObjectLayer() carries information describing a single elementary stream, i.e. a single-layer object or one layer of a multi-layer object. This information is carried in a container associated with the corresponding elementary stream.

K.3.1 VideoObject

The decoder configuration information for a visual object of type VideoObject consists of the following elements

- All syntax elements of VisualObjectSequence()

including all syntax elements of one VisualObject()

- including all syntax elements of one VideoObject()
- including all syntax elements of one VideoObjectLayer() excluding the trailing Group_of_VideoObjectPlane() and VideoObjectPlane() that convey the coded data.

The elementary stream consists of the Group_of_VideoObjectPlane(), VideoObjectPlane(), video_plane_with_short_header and all lower layers.

K.3.2 StillTextureObject

The decoder configuration information for a visual object of type StillTextureObject consists of the following elements

- All syntax elements of VisualObjectSequence()
  - including all syntax elements of one VisualObject()
  - including all syntax elements of one StillTextureObject() up to but not including the first call to wavelet_dc_decode() in case of visual_object_verid == "0001".
  - including all syntax elements of one StillTextureObject() up to but not including the first call to shape_object_decoding() in case of visual_object_verid != "0001" and tiling_disable=="1" and texture_object_layer_shape == "01"
  - Including all syntax elements of one StillTextureObject() up to but not including the first texture_tile_start_code in case of visual_object_verid != "0001" and tiling_disable=="0".

© ISO/IEC 2004 – All rights reserved

K.3.3 MeshObject

The decoder configuration information for a visual object of type MeshObject consists of the following elements

- All syntax elements of VisualObjectSequence()

including all syntax elements of one VisualObject()

K.3.4 FaceObject

The decoder configuration information for a visual object of type FaceObject consists of the following elements

- All syntax elements of VisualObjectSequence()

including all syntax elements of one VisualObject()

K.3.5 3DMeshObject

The decoder configuration information for a visual object of type 3DMeshObject consists of the following elements

- All syntax elements of VisualObjectSequence()

including all syntax elements of one VisualObject()

© ISO/IEC 2004 – All rights reserved

Annex L
(informative)

Rate control

L.1 Frame Rate Control

L.1.1 Introduction

Rate control and buffer regulation is an important issue for both VBR and CBR applications. In the case of VBR encoding, the rate controller attempts to achieve optimum quality for a given target rate. In the case of CBR encoding and real-time application, the rate control scheme has to satisfy the low-latency and VBV buffer constraints. In addition, the rate control scheme has to be applicable to a wide variety of sequences and bit rates.

The scalable rate control (SRC) scheme is designed to meet both VBR without delay constraints and CBR with low-latency and buffer constraints. The SRC scheme is scalable for various bit rates (e.g. 10kbps to 1Mbps), various spatial resolutions (e.g. qcif to cif) and various temporal resolutions (e.g. 7.5fps to 30fps) and various coders (e.g. DCT and wavelet). The technique can handle I, P, and B pictures. The SRC is for single VO, and extensions to M-VOs and other improvements are being addressed in Clause L.2 The current description in Clause L.1 only handles I and P pictures.

L.1.2 Description

The SRC scheme assumes that the encoder rate distortion function can be modeled as

$$R=X1*S*Q^{**}(-1)+X2*S*Q^{**}(-2)$$

The encoding bit count is denoted as R. The encoding complexity which is mean absolute difference (MAD) is denoted as S. The quantiser scale is denoted as Q. The modeling parameters are denoted as X1 and X2. Because of the generality of the assumption, the scalable rate control scheme is applicable to a variety of bit rates, spatial resolutions, temporal resolutions, buffer constraints and types of coders.

There are four steps in the SRC scheme:

1) Initialization
   - X1 and X2 are the first and second order coefficients.
2) Computation of the target bit rate before encoding.
   - The target bit rate is computed based on the bits available and the last encoded frame bits. If the last frame is complex and uses excessive bits, more bits should be assigned to this frame. However, there are fewer bits left for encoding. Thus, fewer bits can be assigned to this frame. A weighted average reflects a compromise of these two factors.
   - A lower bound of target rate (R/30) is used so that minimal quality is guaranteed.
   - The target rate is adjusted according to the buffer status to prevent both overflow and underflow.
3) Computation of the quantiser scales (QP) before encoding.
   - QP is solved based on the model parameters X1 and X2.
   - QP is clipped between 1 and 31.
   - QP is limited to vary within 25 percent of the previous QP to maintain a VBR quality.

© ISO/IEC 2004 – All rights reserved

4) After encoding model parameters is updated based on the encoding results of the current frame.

- The rate distortion model is updated based on the encoding results of the current frame. The bits used for the header and the motion vectors are deducted since they are not related to QP.
- The data points are selected using a window whose size depends on the change in complexity. If the complexity changes significantly, a smaller window with more recent data points is used.
- The model is calibrated again by rejecting the outlier data points. The rejection criteria are that data point is discarded when the prediction error is more than one standard deviation.
- The next frame is skipped if the current buffer status is above 80 percents.

The algorithm is described as follows:

Step 1: Initialization after the first frame is coded

Rs: bit rate for the sequence (or segment). /* e.g., 24000 bits/sec */
Rf: bits used for the first frame. /* e.g., 10000 bits */
Rc: bits used for the current frame. It is the bit count obtained after encoding.
Rp: bits to be removed from the buffer per picture.
Ts: number of seconds for the sequence (or segment). /* e.g., 10 sec */
Ec: mean absolute difference for the current frame after motion compensation.
If the macroblock is intra coded, the original spatial pixel values are summed.
Qc: quantisation level used for the current frame.
Nr: number of P frames remaining for encoding.
Ns: distance between encoded frames. /* e.g., 4 for 7.5 fps */
Rr: number of bits remaining for encoding this sequence (or segment).
T: target bit to be used for the current frame.
S: number of bits used for encoding the previous frame.
Hc: header and motion vector bits used in the current frame. It includes all the information except to the residual information.
Hp: header and motion vector bits used in the previous frame. It includes all the information except to the residual information.
Ql: quantisation level used in the previous frame
Bs; buffer size e.g., R/2
B; current buffer level e.g., R/4 - start from the middle of the buffer

```
Rr = Ts*Rs-Rf;          /* total number of bits available for this segment */
Rp = Rr/Nr;             /* the average bits to be removed from the buffer */
if(first frame){
    Qc=15;
    goto step 4
}
```

Step 2: Target Bit Calculation (Before Encoding the Current Frame)

```
T=Max(Rs/30,Rr/Nr*0.95+S*0.05);/*each frame is assigned a minimum of R/30 */
T=T*(B+2*(Bs-B))/(2*B+(Bs-B)); /* increase if less than half */
                    /* decrease if more than half, don't change if half */
if (B+T > 0.9*Bs)
    T = Max(Rs/30, 0.9*Bs-B);   /* to avoid overflow*/
else if (B-Rp+T < 0.1*Bs)
    T = Rp-B+0.1*Bs;        /* to avoid underflow*/
```

© ISO/IEC 2004 – All rights reserved

Step 3: Quantisation Level Calculation (Before Encoding the Current Frame)

```
T= Max(Rp/3+Hp, T);

if ((X2==0) || (((X1*Ec)**2+4*X2*Ec*(T-Hp))<0))
    Qc = X11*Ec/(T-Hp);          /* fall back 1st order mode */
else Qc= (2*X2*Ec)/(sqrt((X1*Ec)**2+4*X2*Ec*(T-Hp))-X1*Ec)
                    /* 2nd order mode */

Qc = Min (ceil(Ql*1.25), Qc, 31); /* clipping*/
Qc = Max (ceil(Ql*0.75), Qc, 1);  /* clipping */
Quantisation();
Encoding();
```

Step 4: After encoding the current frame

```
B += Rc - Rp;          /* update buffer fullness */
Rr -= Rc;              /* update the remaining bit counts */
S = Rc;            /* update the previous bit counts */
Hp = Hc;               /* update the previous header and motion bit counts */
Ql = Qc;               /* update the previous quantisation level */
Nr--;                  /* update the frame counter */
UpdateRDModel(Qc,Rc,Ec,Hc,X1,X2);
                /* estimation of a new model */
while (B > 0.8 * Bs) {
    skip_next_frame();
    Nr--;
    B -= Rp;
}
```

If the buffer has reached 80% of the buffer size, the encoder will skip the upcoming frame to be encoded. Thus the buffer is reduced to prevent from the buffer overflow.

```
UpdateRDModel (Qc, Rc, Ec, Hc, X1, X2) {
    Qall[n]: quantisation levels for the past frames
    Rall[n]: scaled encoding complexity used for the past frames
    n: number of encoded past frames;
    x: matrix contains Q;
    y: matrix contains Q*(R-H)/E;
    Ep: mean absolute difference for the previous frame. This is computed after motion compensation for
    the Y component only.

    Rall[n] = (Rc-Hc)/Ec;
    Qall[n] = Qc;

    w = Min(total_data_number, 20) /* Maximum data number set to 20 */
    if(Ep>Ec)
       w = ceil(Ec/Ep*w);          /* sliding window for scene change */
    else
       w = ceil(Ep/Ec*w);          /* sliding window for scene change */
    Ep = Ec;                    /* update mad */

    for(i=n; i>n-w; i++){
        Qp[i]=Qall[i]
        Rp[i]=Rall[i]
    }
    Estimator(Qp,Rp,w,b);
    RemoveOutlier(X1,X2,Qp,Rp,Ec,w);
    Estimator(Qp,Rp,w,b);
}
```


© ISO/IEC 2004 – All rights reserved

```
Estimator(Qp, Rp,w,b) {
    X11: First order coefficient.
    x = [1, Qp[i]**(-1)( i=1,2,...w)]          /* Dimension wx2 */
    y = [Qp[i]*Rp[i] (i=1,2,.....w)]      /* Dimension wx1 */

    for (i=1; i<=w; i++)   X11 += y[i]/w;

    if (all Qp[i] are not all the same) {
        b = (x_Transpose*x)**(-1)*x_Transpose*y;
        /* Dimension of the matrix */
        /* 2x1 = (2xw * wx2)**(-1)*(2xw)*(w*1) */
        X1 = b(1,1);
        X2 = b(2,1);
    }
}

RemoveOutlier(X1,X2,Qp,Rp,Ec,w) {
    error[w];     Estimation error
    new_w;

    for (i=1; i<=w; i++) {
        std += ((X1*Ec*Qp[i]**(-1)+X2*Ec*Qp[i]**(-2)-Rp[i]*Ec))**2;
        error[i] = X1*Ec*Qp[i]**(-1)+X2*Ec*Qp[i]**(-2)-Rp[i]*Ec;
    }

    /* When w is 2 the threshold should be 0, some implementation may resolve a small number due to
    precision problem. */
    threshold = sqrt(std/w);              /* Setup rejection threshold */

    new_w = 0
    for (i=1; i<=w; i++){
        if(abs(error[i]) <= threshold){
            new_w++;
            Qp[new_w] = Qp[i];
            Rp[new_w] = Rp[i];
        }
    }
    w = new_w;
}
```

L.1.3 Summary

The scalable rate control scheme achieves frame level rate control for both VBR and CBR cases. It assumes a simple quadratic rate distortion function of the video encoder. In the case of CBR encoding, a variable frame rate approach is used to achieve the target rate. If a tighter rate control is desired, the same technique is applicable at either slice layer or macroblock layer. Because of the generality of this scheme, extension of such a scheme to shape coding and Multiple-VO coding can be easily made on the basis of the existing rate control framework.

L.2 Multiple Video Object Rate Control

This clause describes an algorithm which achieves a constant bit rate when coding multiple video objects. The scheme assumes the following relationship between the bits and quantiser scale for the i-th object:

$$R[i]=X1[i]*S[i]*Q[i]**(-1)+X2[i]*S[i]*Q[i]**(-2)$$

The above is a simple extension of the relationship used for S-VOP rate control when sprite_enable == 'static'.

© ISO/IEC 2004 – All rights reserved

As shown in Figure L.1, the algorithm for performing the joint rate control can be decomposed into a number of steps, which may be grouped into a pre-encoding and post-encoding stage. Each block is briefly described below.

L.2.1 Initialization

The initialization process is very similar to the S-VOP process when sprite_enable == 'static'. Since a single buffer is used, the buffer drain rate is the same. The major difference is that most parameters are extended to vector quantities.

L.2.2 Quantisation Level Calculation for I-frame and first P-frame

To calculate the quantiser scales which are used for the I-frame and first P-frame, a bit allocation table based on Human Visual Sensitivity (HVS) of color tolerance is used. This scheme assigns a target bit rate which is predicted by histogram of variance and HVS classification.

Color/Variance classification: A macro block is classified into one of 32 macro block classes based on HVS and one of 16 classes based on the variance. A reference quantisation step size is assigned to the macro block according to its macro block class. A macro block is classified into a sensitive category if the number of pixels in the macro block which satisfy LUT (Look-Up Table) is larger than a threshold value. Otherwise the macro block is classified into insensitive category.

$$texture_class = H_n + V_n$$

$$H_n = \sum_{y=1}^{16}\sum_{x=1}^{16}\left(\left|P_{x,y} - P_{x-1,y}\right| > th\right)$$

$$V_n = \sum_{y=1}^{16}\sum_{x=1}^{16}\left(\left|P_{x,y} - P_{x,y-1}\right| > th\right)$$

$$th = f\left(\sum_{y=1}^{16}\sum_{x=1}^{16} P_{x,y}\right)$$

$$color_class = \min(color_tolerance)$$

$$color_tolerance = LUT(color_{mean})$$

$$color_{mean} = \frac{1}{64}\sum_{y=1}^{16}\sum_{x=1}^{16}(P_{x,y})$$

The block variance classification allows us to estimate the number of coded bits in a block. Generally, the number of coded bits generated in a block is proportional to the block variance when the quantisation step size is assumed to be constant. We divide the magnitude of block variance into 16 intervals. Threshold values in each interval can be found in Table L.1. Table L.2 illustrates the reference quantisation step size based on HVS for each macro block class and picture type.

Table L.1 — Block variance classes

Bv VOP	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
I	16	32	64	128	256	384	512	768	1024	1280	1792	2048	3072	4096	6154	¥
P	8	16	32	48	64	96	128	160	192	256	320	448	640	896	1536	¥
Bv : Block Variance Class Number, I : INTRA FRAME TYPE, P : INTER FRAME TYPE																

Table L.2 — Reference quantisation step size

Mc	0	1	2	3	4	5	6	7
Qss	-4	-2	-1	0	1	2	3	4

© ISO/IEC 2004 – All rights reserved

Initial QP setting method

```
SetInitialQP()
{
    PreDiff = 1000000;
    Classification();
    if(Flat Image)        TargetBit = AverageBitPerFrame * 9;
    else                  TargetBit = AverageBitPerFrame * 4;

    /* Find Optimal Qp value */
    for(i=1; i<31; i++){
      SetMB_QP(i); /* Allocate HVS optimal QP to each MB */
      EstimatedBit = Bitestimation(); /* Pre-bit-estimation */
      Diff = abs(TargetBit - EstimatedBit);
      if(Diff<PreDiff){
         PreDiff = Diff;
         MinErrorQ = i;
      }
    } /*  MinErrorQ is the final average Qp value */
    SetMB_QP(MinErrorQ); /* Allocate HVS Optimal Qp to each MB */
}
```

Frame target bit allocation

$$\text{FrameTargetBit} = \begin{Bmatrix} \frac{bitrate}{fps} * (4 \sim 9) ---- \text{Intra frame} \\ \frac{bitrate}{fps} ----- \text{First Inter Fr.} \end{Bmatrix}$$

Frame coded-bit estimation: There are 512 classes which are combinations of 32 macro block color classes(Mc) and 16 block variance classes(Bv). We can estimate the total number of coded bits in a frame with the histogram and the bit model table. **FrameEstimatedBit** is the estimated number of coded bit in the current frame

$$\text{FrameEstimateBit} = \sum_{Mc=0}^{31} \sum_{Bv=0}^{15} \{if(Coded)B[R[Mc]][Bv] * H[Mc][Bv]\} + \text{ShapeBit}$$

In the above equation, B[Mc][Bv] is the number of bits in the bit table and H[Mc][Bv] is the computation result of the histogram when a block is classified into Bv. The histogram can be computed with the information of macro block classes and block classes of the previous frame because the current frame to be coded and the previous frame are very highly correlated. Therefore the real-time coding process is possible.

Bit Table

The number of macro block classes and block classes in a frame depends on the statistical characteristics of the frame. However, the target number of bits which is assigned to the frame by bit allocation is not related with statistical characteristics of the frame. Hence the target number of bits for frame has much difference from the total number of coded bits in the frame when all macro blocks in the frame are quantised with their reference quantisation step sizes. So reference quantisation step sizes should be updated in order that the total number of coded bit in the frame becomes almost the same as the number of target bits for the frame. To determine the ratio of change of reference quantisation step size, we have to estimate the total number of coded bits in a frame when all macro blocks in the frame are quantised with their reference quantisation step sizes. The bit table shows the average value of the estimated number of bits in a block according to macro block classification and block classification in Intra/Inter frame. It is possible to estimate the total number of coded bits in a frame with the bit table.

© ISO/IEC 2004 – All rights reserved

```
(INTRA FRAME Bit Table)
int BitTableI[32][16] =
{
65,145,174,208,242,278,298,317,341,352,365,386,400,419,435,438,
24, 79,102,127,153,179,194,209,232,243,255,266,272,279,286,292,
16, 52, 70, 93,114,136,148,161,180,190,201,214,220,226,231,234,
12, 38, 53, 73, 91,110,122,132,150,159,166,179,185,188,193,195,
11, 29, 42, 59, 75, 92,104,112,131,138,144,154,158,160,163,168,
10, 24, 35, 49, 64, 79, 91, 98,116,122,127,135,137,142,148,159,
 9, 19, 29, 42, 55, 69, 80, 87,103,109,116,122,126,130,137,147,
 9, 17, 25, 37, 48, 61, 72, 79, 95,100,105,111,114,119,125,135,
 9, 15, 22, 33, 43, 55, 65, 71, 86, 91, 97,103,106,110,115,123,
 8, 13, 19, 29, 39, 50, 59, 65, 79, 84, 89, 95, 98,100,103,114,
 8, 12, 18, 26, 35, 45, 54, 60, 73, 78, 84, 88, 89, 95, 99,104,
 8, 11, 16, 24, 32, 42, 51, 55, 68, 74, 79, 83, 85, 91, 94, 98,
 8, 11, 15, 22, 30, 38, 47, 51, 63, 69, 74, 78, 81, 86, 88, 90,
 8, 10, 14, 21, 28, 35, 44, 48, 59, 64, 69, 75, 77, 79, 80, 83,
 8, 10, 13, 19, 26, 33, 41, 45, 55, 61, 65, 71, 71, 74, 76, 78,
 8,  9, 12, 18, 24, 31, 39, 42, 52, 58, 61, 66, 66, 70, 72, 75,
 8,  9, 12, 17, 23, 29, 37, 40, 48, 55, 57, 62, 63, 65, 69, 73,
 8,  9, 11, 16, 21, 27, 35, 37, 46, 51, 55, 60, 60, 64, 66, 69,
 8,  9, 11, 14, 20, 26, 33, 36, 43, 48, 52, 56, 59, 63, 65, 66,
 8,  9, 10, 14, 19, 25, 31, 34, 41, 45, 50, 55, 57, 61, 64, 66,
 8,  9, 10, 13, 18, 23, 30, 33, 40, 44, 48, 53, 54, 59, 62, 63,
 8,  8, 10, 12, 17, 22, 28, 31, 38, 42, 46, 50, 52, 57, 60, 62,
 8,  8,  9, 12, 16, 21, 27, 30, 36, 40, 44, 49, 51, 55, 57, 60,
 8,  8,  9, 11, 16, 21, 26, 29, 35, 39, 43, 46, 48, 54, 55, 58,
 8,  8,  9, 11, 15, 20, 25, 28, 33, 38, 41, 44, 45, 51, 54, 57,
 8,  8,  9, 11, 14, 19, 24, 27, 32, 36, 40, 43, 44, 48, 50, 55,
 8,  8,  9, 10, 13, 18, 24, 26, 31, 35, 38, 42, 43, 46, 48, 55,
 8,  8,  8, 10, 13, 17, 23, 25, 29, 33, 37, 41, 41, 43, 46, 55,
 8,  8,  8, 10, 13, 17, 22, 24, 29, 32, 36, 39, 40, 42, 41, 53,
 8,  8,  8,  9, 12, 16, 22, 23, 28, 31, 35, 38, 39, 40, 43, 52,
 8,  8,  8,  9, 12, 16, 21, 22, 27, 30, 34, 36, 37, 40, 43, 52,
};
```

L.2.3 Update Rate-Distortion Model

After the encoding stage, the parameters for the RD model are updated. Given the number of texture bits, R[i], the quantisation level, Q[i], and the mean absolute difference, S[i], for the current and a specified number of past frames, the model parameters, X1[i] and X2[i] can be calculated using a least-squares estimation.

L.2.4 Post-Frameskip Control

The post-frameskip control determines the value of *N_skip_post*. It is invoked after the buffer has been updated, i.e., the sum of bits used to code every VO is added to the current buffer level and the number of bits to be removed per picture is subtracted. Essentially, if the "virtual buffer" has exceeded a specified level (e.g., 80% of the buffer size), the encoder will skip the upcoming frames so as to reduce the current buffer level. The virtual buffer level (VB) is given by:

$$VB = B + Bp - (N_skip_post+1)*Rp$$

where the new variable, Bp, denotes the number of bits used in the previous frame. As before, B is the current buffer level and Rp is the number of bits removed from the buffer per picture. After the value of *N_skip_post* has been found, the value of *N_skip_pre* is added to it, where *N_skip_pre* is calculated in the pre-frameskip control. The time instant is updated based on the total frames to be skipped, *N_skip* = *N_skip_pre* + *N_skip_post*.

© ISO/IEC 2004 – All rights reserved

Mode of Operation: Since coding decisions may change according to the environment, it is desirable to have a mechanism to detect and keep an update of such changes.

```
if(N_skip > SkipTH)
    Operate in LowMode
else
    Operate in HighMode
```

The mode of operation can be used to impose constraints on the coding parameters, assist in setting the weights for target distribution, and influence the shape rate control mechanism.

Target Estimation: The initial target bit rate can be calculated based on the number of available bits for the segment and the number of bits used in the previous corresponding VO. A similar lower bound to the S-VOP simulation can be used to target minimum quality.

Joint Buffer Control: The joint buffer control symbolizes the same operation as buffer control in S-VOP with sprite_enable == 'static', however, the target is a sum of individual targets, i.e., T = sum of T[i].

Pre-FrameSkip Control: Often, in low bit rate coding conditions, the target which emerges from the joint buffer control may not be enough to even code the motion, shape, and header information, let alone the texture. In that case, there should be a mechanism to alert other parts of the system (e.g., target distribution, QP calculation, time instant update) that there is a deficiency in the number of allocated bits. Let s = T - Hp be the difference between the target and the amount of bits used in the previous frame for shape, motion, and header, and Beta be a bit threshold greater than zero. N_skip_pre is determined by:

```
while(s < Beta)
    N_skip_pre = N_skip_pre + 1
    s = s + Rd
```

Target Distribution: The joint buffer control provides a new total target to be distributed among each VO. The distribution can be based on the size, the motion, and the variance (MAD^2) of each object:

$$T[i] = (MOTION[i]*wm + SIZE[i]*ws + MAD2[i]*wv)*T$$

This type of distribution is only executed when the target is valid. The target is valid when it is greater than the sum of header bits used for coding the previous frame. If it is not valid, all the targets are made negative. The negative targets can serve as a flag for clipping the quantiser scales to lie in a specified range [Qc 31].

Shape Rate Control: The value of AlphaTH has considerable effect over the number of bits which will be spent on the shape information. Using the current mode of operation, an appropriate value can be determined.

- If the mode of operation is LowMode, then increment the current AlphaTH by AlphaINC;
- If the mode of operation is HighMode, then decrement the current AlphaTH by AlphaDEC.

The minimum value of AlphaTH is 0. This implies lossless shape encoding. The largest value that AlphaTH may take is 255, however, it is recommended that an AlphaMAX in the range [32, 64] be specified.

© ISO/IEC 2004 – All rights reserved

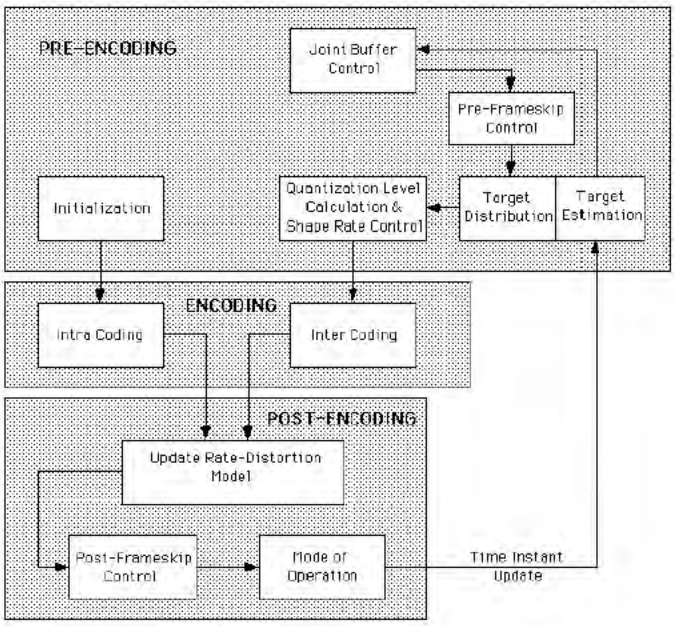


Figure L.1 -- Block diagram for multi-VO rate control

L.3 Macroblock Rate Control

The rate control method described in Clause L.1 used a fixed quantiser scale QP for all the macroblocks in a VOP, where the value of QP was chosen to encode the VOP with a target number of bits. In this clause we describe a method for adapting the value of QP at each macroblock within the VOP. This technique achieves the VOP target more accurately, which is useful for low-delay video applications with small buffer constraints, and allows adapting QP with the macroblock energy and other perceptually relevant measures. The method attempts to combine the strengths of the following contributions:

- A model of the human visual system as described in Clause L.2 can be used to compute W_j.
- The model with two parameters maintains a near-constant value of QP for all the macroblocks in the VOP. This approach is more appropriate at lower bit rates, since it requires a small number of bits for encoding the QP's (using DQUANT).
- The model with single parameter adapts QP with the energy of the block by using finer quantisation (smaller QP) for macroblocks of flatter image regions. It requires a larger QP bit overhead and hence is more appropriate at higher bit rates.

L.3.1 Rate-Distortion Model

We model the number of bits B_i produced for the i-th macroblock as follows:

$$B_i = \begin{cases} \frac{A_1}{QP_i^2} MAD_i^2; & Bitrate > R \\ \frac{A_2}{QP_i^2} MAD_i + \frac{A_3}{QP_i} MAD_i; & Bitrate \le R \end{cases} \quad (1)$$

- A1 , A2 and A3 are the model parameters.

© ISO/IEC 2004 – All rights reserved

- Bitrate is the rate available (in bits per pixel) for the texture of the VOP.
- R is a threshold. By default, we set R=0.085.

L.3.2 Target Number of Bits for Each Macroblock

Let N be the number of macroblocks in a VOP. The target number of bits, T_i, for the i-th macroblock is:

$$T_i = \frac{W_i \, MAD_i}{\sum_{j=1}^{N} W_j \, MAD_j} T_{TEX}, \qquad (2)$$

where:

- W_i is a weight that indicates the perceptual importance of the i-th macroblock. By default, we set $W_1 = W_2 = \ldots = W_N = 1$, but they could be obtained from similar techniques as described in Clause L.2.
- T_{TEX} is the target number of bits for the texture or DCT coefficients (luminance and chrominance components) of the VOP, which is the target frame bits T from frame-layer rate control [see Clause L.1] minus the bits for motion, shape, and syntax overhead. If the latter number of bits is not available for the current VOP, one can subtract those from the previous VOP. Bitrate in (1) is T_{TEX} divided by the number of pixels in the VOP.

Observe that if we combine (1) and (2), we obtain the following expression:

$$QP_i^2 = \begin{cases} MAD_i \, C_1; & Bitrate > R \\ C_2; & Bitrate \le R \end{cases} \qquad (3)$$

where the values of C_1 and C_2 depend on $\{ A_1, MAD_1, \ldots, MAD_N, T \}$ and $\{A_2, A_3, MAD_1, \ldots, MAD_N, T\}$, respectively. The model parameters A_1, A_2, and A_3 are estimated while encoding and in practice they are nearly constant within a VOP, and hence C_1 and C_2 are also approximately constant. As a result, observe that for large Bitrate, the quantiser scale QP will increase with the MAD energy of the macroblock, and for low Bitrate, QP will be nearly constant for all the macroblocks in the VOP.

L.3.3 Macroblock Rate Control

Step 1 Initialization.

N - number of macroblocks in the VOP,
T_{TEX} - number of bits available for encoding the VOP texture

If first frame, let A1= A1_prev= 100, A2= A2_prev= 400, A3= A3_prev= 0
Otherwise, let A1= A1_prev , A2= A2_prev, and A3= A3_prev.

If Bitrate > R = 0.085, let K=1, otherwise let K=2.
Set macroblock index i=1 and skip= 0.
Finally, let $S_1 = \sum_{j=1}^{N} W_j \, MAD_j$. (By default, set W1= W2= ...=WN = 1.)

© ISO/IEC 2004 – All rights reserved

Step 2 Compute QP for i_th macroblock

Let $T_i = \frac{W_i\, MAD_i}{S_i} T_{TEX}$, the target number of bits per macroblock and:

$$QP = \begin{cases} MAD_i \sqrt{\frac{A_1}{T_i}}; & K = 1 \\ Q: \frac{A_2}{Q^2} MAD_i + \frac{A_3}{Q} MAD_i = T_i; & K = 2 \end{cases}$$

If $(A_3 MAD_i)^2 + 4\, T_i A_2 MAD_i$ is negative, there is no value of Q that solves the second-order equation for K=2. In that case, the term with A_2 is dropped and the following is used instead:

$$QP = \frac{MAD_i A_{3_X11}}{T_i},$$

where A_{3_X11} is similar to the X11 parameter in the frame-based rate control in subclause L.1.2.

Round QP to value in 1, ..., 31, and indicate the difference between the current value of QP and that of the previous macroblock in DQUANT. (Recall that DQUANT can only take values within {-2, -1, 0, 1, 2 }, so clipping may be needed.)

Step 3 Encode Macroblock with QP

Step 4 Update Counters

Let B_i' be the actual number of texture (DCT) bits produced when encoding the i_th macroblock. Compute:

$$T_{TEX} = T_{TEX} - B_i', \text{ and } S_{i+1} = S_i - W_i\, MAD_i$$

Step 5 Update A_1, or A_2, A_3, A_{3_X11}

If B_i' is zero (the DCT produced no bits), then skip= skip +1, and go to Step 6.

Otherwise, do the updating:

If K = 1, $\hat{A}_1 = \frac{B_i'}{MAD_i^2} QP^2$.

Let $A_1' = \hat{A}_1 \frac{1}{i - skip} + A_1' \frac{i - skip - 1}{i}$, $A_1 = A_1' \frac{i}{N} + A_{1_prev} \frac{N - i}{N}$.

If K = 2, use linear regression to estimate A2, A3, and A3_X11, similarly as in subclause L.1.2:
We order the values of B'i, QP, and MADi, for the last W encoded macroblocks (whose B'i > 0) into three sets: { b1, b2, b3, ..., bW }, { q1, q2, ..., qW }, and { m1, m2, ..., mW }, where the bj's, qj's and mj's correspond to the values of B'i, QP, and MADi, respectively, for the j-th macroblock in the set.

Let NC be the number of macroblocks (whose B'i > 0) and set W= min{20, NC }. Define:

P1= [1, 1, ..., 1]T,
P2= [1/q1, 1/q2, ..., 1/qW]T,
P3= [b1q1/m1, b2q2/m2, ..., bWqW/mW]T,
P4= [P1 P2],

© ISO/IEC 2004 – All rights reserved

where the superscript "T" denotes the transpose of a vector or matrix. P1, P2 and P3 are vectors of dimension W x 1, and P4 is a matrix of W x 2 elements.

Compute:

$$\begin{bmatrix} A_2 \\ A_3 \end{bmatrix} = (P4^T \cdot P4)^{-1} P4^T \cdot P3,$$

and $$A_{3_X11} = \frac{P3^T \cdot P2}{W},$$

where the superscript "-1" denotes the inverse of the matrix.

Step 6. If i = N, Stop (all macroblocks are encoded), let $A_f_prev = A_f$ for f =1,2,3.

Otherwise, i = i+1, and go to Step 2.

© ISO/IEC 2004 – All rights reserved

Annex M
(informative)

Binary shape coding

M.1 Introduction

This part of ISO/IEC 14496 will be the first international standard allowing the transmission of arbitrarily shaped video objects (VO). A time instant of a VO is called Video Object Plane (VOP). A VOP is a rectangular video frame or a part thereof. Following an object-based approach, the ISO/IEC 14496-2 video encoder applies the motion, texture and shape coding tools to the VOP using I, P, and B modes similar to the modes of MPEG-2. The encoder transmits texture, motion, and shape information of one VO within one bit stream. The bit streams of several VOs and accompanying composition information can be multiplexed such that the decoder receives all the information to decode the VOs and arrange them into a video scene (Figure M.1).

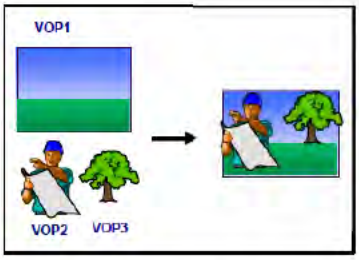


Figure M.1 — Object-based coding requires the decoder to compose different Video Object Planes (VOP) into a scene.

An arbitrarily shaped VO can have binary shape information only or binary shape and texture information. Several shape coding algorithms were evaluated [1][2]. Binary shape information is transmitted as a bitmap using a context-based arithmetic coder described in Clause M.2. Coding of texture for arbitrarily shaped VOs is described in Clause M.3. This part of ISO/IEC 14496 leaves it up mainly to the encoder to achieve efficient coding of texture at object boundaries by employing an appropriate texture extrapolation algorithm. The efficient coding of a VO with lossily encoded shapes requires an encoder architecture as described in Clause M.4. In Clause M.5, guidelines for coding arbitrarily shaped VOs will be presented. Different shape encoding modes are investigated in order to achieve a bit efficient representation of a VO or in order to allow for fast encoding that is important for real-time applications.

M.2 Context-Based Arithmetic Shape Coding

In order to enable content based access to video objects, this part of ISO/IEC 14496 codes the shape of video objects. This part of ISO/IEC 14496 chose to code the shape as a bitmap [2]. For binary shape coding, a rectangular bounding rectangle enclosing the arbitrarily shaped VOP is formed such that its horizontal and vertical dimensions are multiples of 16 pels (macroblock size).

Each block of size 16x16 pels within this bounding rectangle is called binary alpha block (bab). Each bab is associated with the co-located macroblock. Three types of babs are distinguished and signaled to the decoder: Transparent blocks do not contain information about the object, opaque blocks are located entirely inside the object and boundary blocks cover part of the object as well as part of the background. For boundary blocks a context-based shape coder was developed. This coder exploits the spatial redundancy of the binary shape information to be coded. Pels are coded in scan-line order and row by row. In the following paragraphs, shape encoding in intra mode is described first. Then, this technique is extended to include an inter mode.

© ISO/IEC 2004 – All rights reserved

M.2.1 Intra Mode

In intra mode, three different types of macroblocks are distinguished: Transparent and opaque blocks are signaled as macroblock type. The macroblocks on the object boundary containing transparent as well as opaque pels belong to the third type. For these boundary macroblocks, a template of 10 pels is used to define the causal context for predicting the shape value of the current pel (Figure M.2a). For encoding the state transition, a context-based arithmetic encoder corresponding to the arithmetic decoder as defined in subclause 7.5.3 is used. The probability table of the arithmetic encoder for the 1024 contexts was derived from sequences that are outside of the test set used for comparing different shape coders. With two bytes allocated to describe the symbol probability for each context, the table size is 2048 bytes. In order to avoid emulation of start codes like VOP start code, the arithmetic coder stuffs one '1' into the bitstream whenever a long sequence of '0' is sent.

The template extends up to 2 pels to the left, to the right and to the top of the pel to be coded (Figure M.2a). Hence, for encoding the pels in the 2 top and left rows of a bab, parts of the template are defined by the shape information of the already transmitted babs on the top and on the left side of the current bab. For the 2 right-most columns, each undefined pel of the context is set to the value of its closest neighbor inside the bab. In case the top and left babs are transparent, the top and left 2 rows of the current bab are duplicated in order to define the context for the template for intra mode.

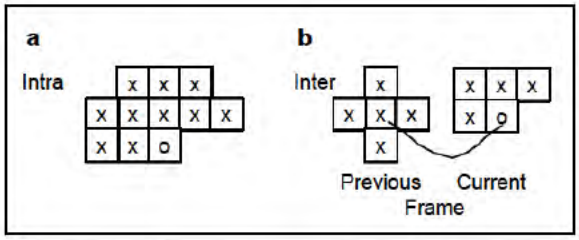


Figure M.2 — Templates for defining the context of the pel to be coded (o),
a) defines the intra mode context,
b) the context when coding in inter mode.
The alignment is done after motion compensating the previous VOP [1].

In order to increase coding efficiency as well as to allow lossy shape coding, a bab can be subsampled by a factor of 2 or 4 resulting in a sub-block of size 8*8 pels or 4*4 pels, respectively. The sub-block is encoded using the encoder as described above. The encoder transmits to the decoder the subsampling factor such that the decoder decodes the shape data and then upsamples the decoded sub-block to macroblock size. Obviously, encoding the shape using a high subsampling factor is more efficient, but the decoded shape after upsampling may or may not be the same as the original shape. Hence, this subsampling is mostly used for lossy shape coding. In order to achieve smooth object boundaries after upsampling, an adaptive non-linear upsampling filter is used. The context of this upsampling filter is shown in Figure M.3.

The efficiency of the shape coder differs depending on the orientation of the shape data. Therefore the encoder can choose to code the bab using horizontal scanning as described above or transpose the macroblock using vertical scanning prior to arithmetic coding.

M.2.2 Inter Mode

In order to exploit temporal redundancy in the shape information, the coder described above is extended by an inter mode requiring motion compensation and a different template for defining the context.

For motion compensation, a 2D integer pel motion vector is estimated using full search for each bab in order to minimize the prediction error between the previous coded VOP shape M'k-1 and the current shape Mk. The shape motion vectors are predictively encoded with respect to the shape motion vectors of neighboring macroblocks. If no shape motion vector is available, texture motion vectors are used as predictors. The shape motion vector of the current block is used to align a new template designed for coding shape in inter mode (Figure M.2b). The template defines a context of 9 pels resulting in 512 contexts. The probability for one symbol is described by 2 bytes giving a probability table size of 1024 bytes. Four pels of the context are neighbors of the pel to be coded, 5 pels are located at the motion compensated location in the previous VOP. Assuming that the motion vector (dx,dy)T points

© ISO/IEC 2004 – All rights reserved

from the current vopk to the previous coded vop'k-1, the part of the template located in the previously coded shape is centered at m'(x-dx,y-dy) with (x,y)T being the location of the current pel to be coded.

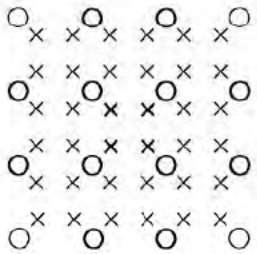


Figure M.3 — For shape upsampling, the upsampled pels (x) lie between the location of the subsampled pels (o). Neighboring pels (bold o) defining the values (transparent or opaque) of the pels to be upsampled (bold x).

In inter mode, the same options as in intra mode like subsampling and transposing are available. For lossy shape coding, the encoder may also decide that the shape representation achieved by just carrying out motion compensation is sufficient thus saving bits by avoiding the coding of the prediction error. The encoder can select one of 7 modes for the shape information of each bab: transparent, opaque, intra, inter with and without shape motion vectors, and inter with/without shape motion vectors and prediction error coding. These different options with optional subsampling and transposition allow for encoder implementations of different coding efficiency and implementation complexity.

M.3 Texture Coding of Boundary Blocks

For motion compensated prediction of the texture of the current VOP, the reference VOP is motion compensated using block motion compensation. In order to guarantee that every pel of the current VOP has a value to be predicted from, all of the boundary blocks and some of the transparent blocks of the reference VOP have to be padded using motion compensation (MC) padding. Boundary blocks are padded using repetitive padding: First boundary pels are replicated in horizontal direction, then in vertical direction making sure that if a value can be assigned to a pel by both padding directions an average value is assigned to the pel. Since this repetitive padding puts a significant computational burden on the decoder, a simpler mean padding is used in a second step. Transparent macroblocks bordering boundary blocks are assigned to an average value determined by the pels of its neighboring padded blocks.

In order to encode the texture of a boundary block, this part of ISO/IEC 14496 treats the macroblock as a regular macroblock and encodes each block using an 8*8 DCT. The decoder decodes the texture and discards all information that falls outside of the decoded shape. In order to increase coding efficiency, the encoder can choose the texture of pels outside of the object such that the bitrate is minimized. This non-normative process is called texture padding. For intra mode, a lowpass extrapolation filter increasing the PSNR of boundary blocks by an average of 2dB was developed [4]. For inter mode setting these pels to 0 gives good efficiency.

VOP texture may be encoded using B-frames. The related shape information of the B-frame is coded using the inter mode as described in below using the closest P or I frame as a reference VOP.

M.4 Encoder Architecture

Figure M.4a shows the block diagram of this object-based video coder. This diagram focuses on the object-based mode in order to allow a better understanding of how shape coding influences the encoder and decoder. Image analysis creates the bounding rectangle for the current VOP S_k and estimates texture and shape motion of the current VOP S_k with respect to the reference VOP S'_{k-1}. Parameter coding encodes the parameters predictively. The parameters get transmitted, decoded and the new reference VOP is stored in the VOP memory and also handed to the compositor of the decoder for display. The increased complexity due to the coding of arbitrarily shaped video objects becomes evident in Figure M.4b that shows a detailed view of the parameter coding.

© ISO/IEC 2004 – All rights reserved

Since this part of ISO/IEC 14496 defines motion vectors for motion compensation of texture and for motion compensation of shape, we refer to them as texture motion vectors and shape motion vectors, respectively. The parameter coder encodes first the shape of each boundary block using shape motion vectors and texture motion vectors for prediction. Lossy shape coding may change a boundary block to a transparent block. If the boundary block is not transparent, its shape motion vector is coded. The shape motion coder knows which motion vectors to code by analyzing the possibly lossily encoded shape parameters.

For texture prediction, the reference VOP is padded as described above. The prediction error is then padded using the original shape parameters to determine the area to be padded. Using the original shape as a reference for padding is again an encoder choice. Using the original or the coded shape for defining the texture of a macroblock makes obviously only a difference if the shape is coded lossily. Assuming that the texture between object and background in the original VOP differs significantly, using the original shape to define the area to be padded in the macroblock prevents the background texture from influencing the padding process. Texture padding will use the object texture for padding resulting in an efficiently to code block. If on the other hand the coded shape is used to define the area to be padded, background texture may affect the padding result potentially increasing the bitrate for texture coding drastically. Finally, the texture of each macroblock is encoded using DCT.

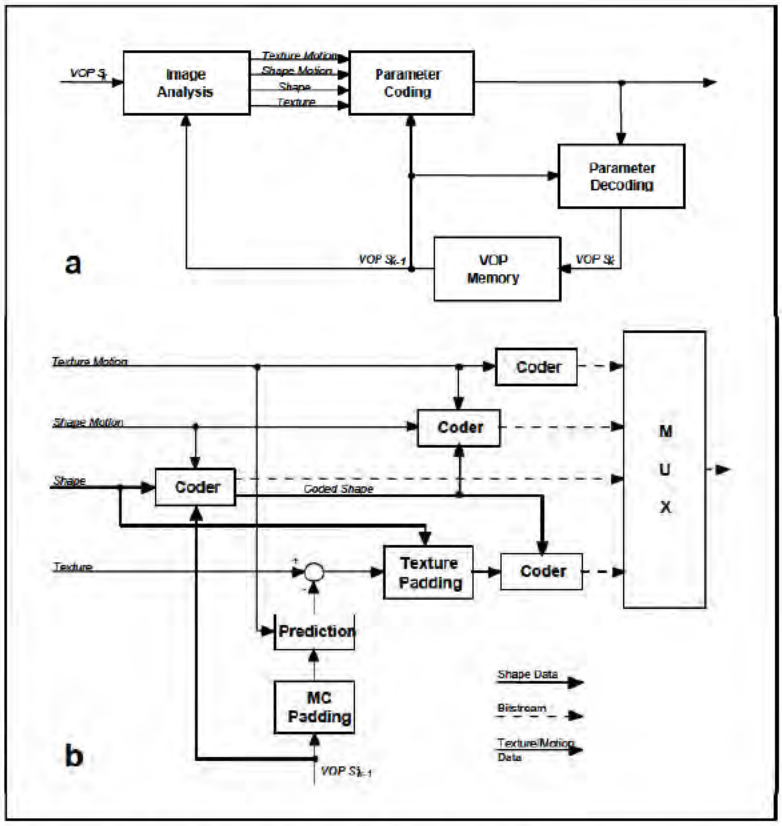


Figure M.4 — Block diagram of the video encoder (a) and the parameter coder (b) for coding of arbitrarily shaped video objects.

M.5 Encoding Guidelines

When encoding arbitrarily shaped VOs several new encoding options become available when compared to coding of rectangular video sequences. Here we provide some hints on lossy shape coding, shape coding mode selection.

© ISO/IEC 2004 – All rights reserved

M.5.1 Lossy Shape Coding

The decoder has no possibility to find out whether the encoder uses lossless or lossy shape coding, what shape coding strategy the encoder uses or texture padding algorithm is used. Therefore, post-processing filter for the binary shape information may not be used. In its reference implementation of a video coder, this part of ISO/IEC 14496 chose to control lossy shape coding by using an alpha-threshold. The alpha threshold defines the maximum number of incorrectly coded pels within a boundary block. A fixed or adaptive alpha-threshold can be used for each bab. Using an alpha threshold may change the topology of the shape. Often, isolated pels at the object boundary are coded as part of the object. Using morphological filters at the encoder to smooth object boundaries or to filter the prediction error with respect to the original shape provides a much more predictable quality of a lossily encoded shape. Furthermore, the encoder can choose to neglect the prediction error signal at the object boundary if it does not change the topology of the object and to update the prediction error in those parts of the block where the prediction error leads to a change of the object topology.

M.5.2 Coding Mode Selection

As far as shape coding is concerned, this part of ISO/IEC 14496 allows for 18 coding modes of each BAB: (Intra/inter/inter with MC)*(horizontal/vertical scanning)*(Subsampling factor 0/1/2). The influence of different shape coding modes on the performance of the coder in terms of coding efficiency but also in terms of computational complexity is of interest, since the shape coder including padding can account for up to 50% of the encoder operations [3]. When encoding a binary shape, checking every mode achieves the highest compression. However, the gain achieved by this checking should be evaluated in relation to the overall bitrate required for coding a VOP. Table M.1 shows the main usage for each coding mode.

Table M.1 — Shape coding modes and their main usage

	Mode	Main Usage
1	intra	I frames, arbitrarily shaped still texture object, error resilience
2	inter, inter MC	P frames
3	horizontal/vertical scanning	Low-bitrate shape coding
4	subsampling to a block size 8x8 or 4x4	Low-bitrate lossy shape coding

The intra mode is used for I frames and for coding the shape of the arbitrarily shaped still texture object. Similarly, the use of the intra mode as well as the inter mode with and without motion compensation should be investigated for coding of P frames. For low-bitrate applications, the shape usually requires a significant percentage of the overall bitrate. Then it makes sense for the encoder to investigate the gain of adapting the scanning direction of each BAB reducing the shape bit rate by up to 8%. In case of lossy shape coding at low bit rates, the subsampling filter reduces the bit rate for shape coding by up to 20% for an alpha threshold of 16. The use of the subsampling filter is most effective for I babs. In case of lossless shape coding, the coder should signal at the beginning of each VOP that the subsampling feature would not be used. This avoids the overhead of signaling the subsampling for each BAB.

M.6 Conclusions

This part of ISO/IEC 14496 standardizes an object-based decoder. Several choices regarding texture extrapolation at object boundaries and shape encoding are available to the encoder. It was found that for texture extrapolation a simple filter setting the texture outside of the *original* object shape to gray is sufficient. In the case of lossy shape coding, not using the *original* object shape for defining the support region of the texture extrapolation may increase the overall bitrate by more than 100%. Therefore, lossy shape coding makes only sense if the encoder architecture presented in this paper is used.

Fixing the bab scanning to horizontal instead of adaptively switching between horizontal and vertical scanning increases the overall bitrate by just 1% but saves significant compute time since the compute intensive arithmetic coding with context determination has to be used only half as often.

© ISO/IEC 2004 – All rights reserved

The downsampling of the shape information increases the computational complexity of the encoder. In the case of lossy shape coding in intra mode, the subsampling increases the subjective quality of the coded object shape due to a non-linear shape upsampling filter. For low bitrate video coding using lossily coded shapes, this filter reduces the overall bitrate.

As the experiments show, the ISO/IEC 14496-2 binary shape coder can operate in a very bit efficient manner. In order to save computational complexity for high bit rate applications, the encoder may choose to not explore all of the encoding options and choose a less efficient shape encoding. The increase of the overall bitrate may be less than 1% at high bit rates but the computational complexity of the encoder decreases by 15% to 50%.

M.7 References

[1] N. BRADY, F. BOSSEN, N. MURPHY, "Context-based arithmetic encoding of 2D shape sequences", Special session on shape coding, ICIP 97, Santa Barbara, 1997

[2] A. KATSAGGELOS, L. KONDI, F. MEIER, J. OSTERMANN, G. SCHUSTER, "MPEG-4 and rate-distortion based shape coding techniques", Proceedings of the IEEE, pp. 1126-1154, June 1998

[3] J. OSTERMANN, "Coding of arbitrarily shaped video objects with binary and greyscale alpha maps: What can MPEG-4 do for you?". ISCAS 98, Monterey, CA, TPA 2-1, June 1998

[4] A. KAUP, "Adaptive low-pass extrapolation for object-based texture coding of moving video", Proc. Visual Communications and Image Processing, SPIE vol. 3024, pp. 731-741, Feb. 1997

© ISO/IEC 2004 – All rights reserved

Annex N
(normative)

Visual profiles@levels

The table that describes the visual profiles and levels is given below, with the following notes:

1 Enhancement layers are not counted as separate objects.

2 The maximum VMV buffer size is the bound on the memory (in macroblock units) which can be used by the VMV algorithm. This algorithm (see Clause D.5) models the pixel memory needed by the entire visual decoding process. This includes memory needed for reference VOPs in the prediction of P, B, and S(GMC)-VOPs and the storage of the reconstructed VOPs until such time that they are released by the decoder, plus the memory required to queue B-VOPs until composition occurs. In profiles that contain more than one layer, the memory requirements include all base and enhancement layers. When belonging to different, overlapping objects, some of these macroblocks may overlay on the display, however separate memory is required (prior to composition) in the VMV.

3 The conformance point for the base layer of Simple Scalable Visual Profile is the Simple Profile @L1 when Simple Scalable Profile @L1 is used and the Simple Profile @ L2 when Simple Scalable Profile @ L2 is used.

4 The VCV decoder rate is the vcv_decoder_rate (H) referred to in Clause D.4; it is the number of macroblocks/second based on the typical spatial and temporal resolutions, as follows:
- 1485 MBs/s corresponds to QCIF at 15Hz.
- 5940 MBs/s corresponds to CIF at 15 Hz and also twice QCIF at 30 Hz.
- 11880 MB/s corresponds to CIF at 30 Hz.
- 7425 MB/s corresponds to 1.25 times CIF at 15 Hz.
- 23760 MB/s corresponds to twice CIF at 30 Hz.
- 97200 MB/s corresponds to twice ITU-R 601 at 30 Hz.
- 489600 MB/s corresponds to twice 1920x1088 at 30 Hz.

5 The total (aggregated) vbv_buffer_size is the sum of the individual VBV buffer occupancies at any given time (in units of 16384 bits) for all VOLs of all VOs. This total VBV size is limited according to the profile and level.

6 The maximum video packet length is defined as the maximum number of bits of data_partitioned_motion_shape_texture() in one video packet. The constraint applies only when the data-partitioning tool is enabled in the bitstream. When data partitioning is disabled, there is no limit on the size of video packet length.

7 N. A. means Not Applicable.

8 The maximum VCV buffer size (cumulative over all layers of all VOs) is twice the maximum number of macroblocks per VOP in the profile and level combination except for the Simple Visual Profile, Simple Scalable profile (Level 1) and Advanced Real Time Simple Profile. For the Simple Visual Profile and the Advanced Real Time Simple Profile, this value is maximum number of macroblocks per VOP. For the Simple Scalable profile (Level 1), it is 1.25 times of the maximum number of macroblocks per VOP. The limit applies to both VCV buffer and the boundary MB VCV buffer.

9 The VCV boundary MB decoder rate column bounds the number of macroblocks containing non trivial shape information. The VCV boundary MB decoder rate constrains the total number of boundary MBs in all VOLs concurrently. Note that the boundary macroblocks are added to both the VCV and boundary VCV buffers.

10 Max unique Quant Tables is for the total number of Quant Tables. It is the sum of intra, non-intra, grey-scale intra, and grey-scale non-intra Quant Tables used for all objects. The four types of tables can be on or off independently in the syntax definition.

© ISO/IEC 2004 – All rights reserved

11 For Simple Profile @ Level L0, the following restrictions apply:

a) The maximum frame rate shall be 15 frames per second;

b) The maximum f_code shall be 1;

c) The intra_dc_vlc_threshold shall be 0;

d) The maximum horizontal luminance pixel resolution shall be 176 pels/line;

e) The maximum vertical luminance pixel resolution shall be 144 pels/VOP;

f) If AC prediction is used, the following restriction applies : QP value shall not be changed within a VOP (or within a video packet if video packets are used in a VOP). If AC prediction is not used, there are no restrictions to changing QP value.

12 For Simple Scalable Profile @ Level L0, the following restrictions apply:

a) Base layer compliant to Simple Profile @ Level L0;

b) The enhancement layer maximum f_code shall be 1;

c) The enhancement layer intra_dc_vlc_threshold shall be 0;

d) For enhancement layer, if AC prediction is used, the following restriction applies: QP value shall not be changed within a VOP (or within a video packet if video packets are used in a VOP). If AC prediction is not used, there are no restrictions to changing QP value;

e) Spatial scalability shall be limited to a 1:1 scaling ratio.

© ISO/IEC 2004 – All rights reserved

Table N.1 — Definition of Version 1 Natural Visual Profiles@Levels

Visual Profile	Level	Typical Visual Session Size	Max ob-jects [1]	Maximum number per type	Max unique Quant Tables	Max. VMV buffer size (MB units)[2]	Max VCV buffer size (MB)[3]	VCV decoder rate (MB/s)[4]	VCV Bound-ary MB decoder rate (MB/s)[5]	Max total VBV buffer size (units of 16384 bits)[6]	Max vbv buffer size (units of 16384 bits)	Max. video packet length (bits)[6]	Max sprite size (MB units)	Wavelet restrictions	Max bitrate (kbit/s)	Max. enhance-ment layers per object
N-Bit	L2	CIF	16	16 x Core or Simple or N-Bit	4	2376	792	23760	11880	80	80	8192	N. A.[7]	N. A.	2000	1
Main	L4	1920 x 1088	32	32 x Main or Core or Simple	4	65344	16320	489600	244800	760	760	16384	65280	Scalable Texture Profile@L2	38400	1
Main	L3	ITU-R 601	32	32 x Main or Core or Simple	4	11304	3240	97200	48600	320	320	16384	6480	Scalable Texture Profile@L1	15000	1
Main	L2	CIF	16	16 x Main or Core or Simple	4	3960	1188	23760	11880	80	80	8192	1584	Scalable Texture Profile@L1	2000	1
Core	L2	CIF	16	16 x Core or Simple	4	2376	792	23760	11880	80	80	8192	N. A.	N. A.	2000	1
Core	L1	QCIF	4	4 x Core or Simple	4	594	198	5940	2970	16	16	4096	N. A.	N. A.	384	1
Simple Scalable[3]	L2	CIF	4	4 x Simple or Simple Scalable	1	3168	792	23760	N.A.	40	40	4096	N. A.	N. A.	256	1 spatial or temporal enhance ment layer
Simple Scalable	L1	CIF	4	4 x Simple or Simple Scalable	1	1782	495	7425	N. A.	40	40	2048	N. A.	N. A.	128	1 spatial or temporal enhance ment layer
Simple Scalable	L0	QCIF	1	1 x Simple or Simple Scalable	1	446	124	1865	N. A.	10	10	2048	N. A.	N. A.	128	1 spatial or temporal enhance ment layer

© ISO/IEC 2004 – All rights reserved

Visual Profile	Level	Typical Visual Session Size	Max ob-jects [1]	Maximum number per type	Max unique Quant Tables	Max. VMV buffer size (MB units)[2]	Max VCV buffer size (MB)[3]	VCV decoder rate (MB/s)[4]	VCV Bound-ary MB decoder rate (MB/s)[5]	Max total VBV buffer size (units of 16384 bits)[6]	Max vbv buffer size (units of 16384 bits)	Max. video packet length (bits)[6]	Max sprite size (MB units)	Wavelet restrictions	Max bitrate (kbit/s)	Max. enhance-ment layers per object
Simple	L3	CIF	4	4 x Simple	1	792	396	11880	N. A.	40	40	8192	N. A.	N. A.	384	N. A.
Simple	L2	CIF	4	4 x Simple	1	792	396	5940	N. A.	40	40	4096	N. A.	N. A.	128	N. A.
Simple	L1	QCIF	4	4 x Simple	1	198	99	1485	N.A.	10	10	2048	N. A.	N. A.	64	N. A.
Simple	L0	QCIF	1	1 x Simple	1	198	99	1485	N.A.	10	10	2048	N. A.	N. A.	64	N. A.

© ISO/IEC 2004 – All rights reserved

Table N.2 — Definition of Version 2 Natural Visual Profiles@Levels

Visual Profile	Level	Typical Visual Session Size	Max ob-jects [1]	Maximum number per type	Max unique Quant Tables	Max. VMV buffer size (MB units)[2]	Max VCV buffer size (MB)[3]	VCV decoder rate (MB/s)[4]	VCV Bound-ary MB decoder rate (MB/s)[5]	Max total VBV buffer size (units of 16384 bits)[6]	Max vbv buffer size (units of 16384 bits)	Max. video packet length (bits)[6]	Max sprite size (MB units)	Wavelet restrictions	Max bitrate (kbit/s)	Max. enhance-ment layers per object
Advanced Coding Efficiency	L4	1920 x 1088	32	32 x Adv. Coding Efficiency or Core or Simple	4	48960	16320	489600	244800	760	760	16384	N. A.	N. A.	38400	1
Advanced Coding Efficiency	L3	ITU-R 601	32	32 x Adv. Coding Efficiency or Core or Simple	4	9720	3240	97200	48600	320	320	16384	N. A.	N. A.	15000	1
Advanced Coding Efficiency	L2	CIF	16	16 x Adv. Coding Efficiency or Core or Simple	4	2376	1188	23760	11880	80	80	8192	N. A.	N. A.	2000	1
Advanced Coding Efficiency	L1	CIF	4	4 x Adv. Coding Efficiency or Core or Simple	4	1188	792	11880	5940	40	40	8192	N. A.	N. A.	384	1
Core Scalable	L3	ITU-R 601	16	16 x Core or Simple or Core scalable or Simple Scalable	4	12906	4032	120960	60480	80	80	16384	N. A.	N. A.	4000	2
Core Scalable	L2	CIF	8	8 x Core or Simple or Core scalable or Simple	4	2970	990	29700	14850	80	80	4096	N.A.	N.A.	1500	1
Core Scalable	L1	CIF	4	4 x Core or Simple or Core scalable or Simple Scalable	4	2376	792	14850	7425	64	64	4096	N.A.	N.A.	768	1
Advanced Core	L2	CIF	16	16 x Core or Simple or Adv. scalable Texture	4	2376	792	23760	11880	80	40	8192	N. A.	see Table 9-10	2000	1
Advanced Core	L1	QCIF	4	4 x Core or Simple or Adv. Scalable Texture	4	594	198	5940	2970	16	8	4096	N. A.	see Table 9-10	384	1

© ISO/IEC 2004 – All rights reserved

Visual Profile	Level	Typical Visual Session Size	Max ob-jects [1]	Maximum number per type	Max unique Quant Tables	Max. VMV buffer size (MB units)[2]	Max VCV buffer size (MB)[3]	VCV decoder rate (MB/s) [4]	VCV Bound-ary MB decoder rate (MB/s)[5]	Max total VBV buffer size (units of 16384 bits)[6]	Max vbv buffer size (units of 16384 bits)	Max. video packet length (bits)[6]	Max sprite size (MB units)	Wavelet restrictions	Max bitrate (kbit/s)	Max. enhance-ment layers per object
Advanced Real Time Simple	L4	CIF	16	16 x Simple or Adv. Real Time Simple	1	792	396	11880	N. A.	80	80	16384	N. A.	N. A.	2000	N. A.
Advanced Real Time Simple	L3	CIF	4	4 x Simple or Adv. Real Time Simple	1	792	396	11880	N. A.	40	40	16384	N. A.	N. A.	384	N. A.
Advanced Real Time Simple	L2	CIF	4	4 x Simple or Adv. Real Time Simple	1	792	396	5940	N. A.	40	40	16384	N. A.	N. A.	128	N. A.
Advanced Real Time Simple	L1	QCIF	4	4 x Simple or Adv. Real Time Simple	1	198	99	1485	N.A.	10	10	8192	N. A.	N. A.	64	N. A.

© ISO/IEC 2004 – All rights reserved

Table N.3 — Definition of Studio Visual Profiles@Levels

Visual Profile	Level	Typical Visual Session formats[1]	Max. Pixel depth	Max. objects	Maximum number per type	Max. VMV buffer size (sample)[2]	Max. VCV buffer size (sample)[3]	VCV decoder rate (sample/s)	VCV Boundary MB decoder rate (sample/s)	Max. total VBV buffer size	Max. vbv buffer size	Max. video packet length (bits)	Max. sprite size (sample)[4]	Wavelet restrictions	Max. bitrate (Mbit/s)	Max. enhancement layers per object
Simple Studio	L4	ITU - R709.60P:444 ITU - R709.60I:444444 2Kx2Kx30P:444	12	1	1 x Simple Studio	12582912	12582912	377487360	377487360	4320	4320	N.A.	N.A.	N.A.	1800	N.A.
Simple Studio	L3	ITU - R709.60I:444 ITU - R709.60I:4224	12	1	1 x Simple Studio	6291456	6291456	188,743680	188,743680	2880	2880	N.A.	N.A.	N.A.	900	N.A.
Simple Studio	L2	ITU - R709.60I:422 ITU-R601:444444	10	1	1 x Simple Studio	4194304	4194304	125,829120	125,829120	1920	1920	N.A.	N.A.	N.A.	600	N.A.
Simple Studio	L1	ITU-R601:4224 ITU-R601:444	10	1	1 x Simple Studio	1313280	1313280	33177600	33177600	576	576	N.A.	N.A.	N.A.	180	N.A.
Core Studio	L4	ITU - R709.60P:444 ITU - R709.60I:444444 2Kx2Kx30P:444	10	16	16 x Core Studio or Simple Studio	50331648	25165824	754974720	754974720	4320	4320	N.A.	150994944	N.A.	900	N.A.
Core Studio	L3	ITU - R709.60I:444 ITU - R709.60I:4224	10	8	8 x Core Studio or Simple Studio	2516582 4	12582912	377487360	377487360	2880	2880	N.A.	75202560	N.A.	450	N.A.
Core Studio	L2	ITU - R709.60I:422 ITU-R601:444444	10	4	4 x Core Studio or Simple Studio	16777216	8388608	251658240	251658240	1920	1920	N.A.	50135040	N.A.	300	N.A.
Core Studio	L1	ITU-R601:4224 ITU-R601:444	10	4	4 x Core Studio or Simple Studio	5253120	2626560	66355200	66355200	576	576	N.A.	8294400	N.A.	90	N.A.

© ISO/IEC 2004 – All rights reserved

[1] ITU - R709 is ITU-R BT. 709 and ITU - R601 is ITU-R BT. 601
444444 means 444(RGB)+ 3 Auxiliary Channels
4224 means 422(YUV)+ 1 Auxiliary Channels

[2] VMV is defined by number of samples which belong to the bounding box of texture regardless shape information. VMV also includes auxiliary channel samples.

[3] VCV is defined by number of samples which belong to the bounding box of texture regardless shape information. VCV also includes auxiliary channel samples.

[4] Max sprite size is defined by number of samples for Sprite memory.

Within Advanced Simple Profile and Fine Granularity Scalable Profile, five levels are defined in each profile as specified in Table N.4.

© ISO/IEC 2004 – All rights reserved

Table N.4 — Definition of Levels in Advanced Simple Profile and Fine Granularity Scalable Profile

Visual Profile	Level	Typical Visual Session Size	Max. objects	Maximum number per type	Max. unique Quant Tables	Max. VMV buffer size (MB units)	Max VCV buffer size (MB)	VCV decoder rate (MB/s)	Max. Percentage of Intra MBs with AC prediction in VCV buffer	Max total VBV buffer size (units of 16384 bits)	Maximum VBV Buffer Size (units of 16384 bits)	Max. video packet length (bits)	Maximum Bitrate (kbits/s) (Note 2)	Maximum number of coded vop-bps (Note 3)
AS	L0	176x144	1	1x AS or Simple	1	297	99	2970	100	10	10	2048	128	N.A.
AS	L1	176x144	4	4x AS or Simple	1	297	99	2970	100	10	10	2048	128	N.A.
AS	L2	352x288	4	4x AS or Simple	1	1188	396	5940	100	40	40	4096	384	N.A.
AS	L3	352x288	4	4x AS or Simple	1	1188	396	11880	100	40	40	4096	768	N.A.
AS	L3 b	352x288	4	4x AS or Simple	1	1188	396	11880	100	65	65	4096	1500	N.A.
AS	L4	352x576	4	4x AS or Simple	1	2376	792	23760	50	80	80	8192	3000	N.A.
AS	L5	720x576	4	4x AS or Simple	1	4860	1620	48600	25	112	112	16384	8000	N.A.
FGS	L0	176x144	1	1x AS or FGS or Simple	1	297	99	2970	100	10	10	2048	128	4
FGS	L1	176x144	4	4x AS or FGS or Simple	1	297	99	2970	100	10	10	2048	128	4
FGS	L2	352x288	4	4x AS or Simple	1	1188	396	5940	100	40	40	4096	384	4
FGS	L3	352x288	4	4x AS or FGS or Simple	1	1188	396	11880	100	40	40	4096	768	4
FGS	L4	352x576	4	4x AS or FGS or Simple	1	2376	792	23760	50	80	80	8192	3000	4
FGS	L5	720x576	4	4x AS or FGS or Simple	1	4860	1620	48600	25	112	112	16384	8000	4

© ISO/IEC 2004 – All rights reserved

Note1: The following restriction applies to L0 of AS Profile and FGS Profile:

- If AC prediction is used, QP value shall not be changed within a VOP (or within a video packet if video packets are used in a VOP). If AC prediction is not used, there are no restrictions to changing QP value.

Note 2: For FGS Profile, this column is the maximum base-layer bitrate.

Note 3: The maximum number of coded vop-bps takes into consideration the shifted bits after applying frequency weighting and/or selective enhancement.

Note 4: The number of FGS, FGST, or FGS-FGST layer is always one. If FGS layer and FGST layer are separated, the number of total enhancement layers is two.

Note 5: The interlace tools are not used for levels L0, L1, L2, and L3 of AS Profile and FGS Profile.

Note 6: It is inherent in the FGS profile that the base and enhancement layers are tightly coupled to one another. To avoid unnecessary memory storage, the following constraints apply to the decoding time relationship of the enhancement layer and the base layer:

- Decoding and composition (or presentation in a no-compositor decoder) of each FGS or FGST VOP shall be performed in the same time unit.
- Decoding of each FGS and FGST VOP shall be performed immediately after the reference base layer VOP(s) are decoded without violating the above constraint.

© ISO/IEC 2004 – All rights reserved

Annex O
(informative)

3D Mesh Coding

O.1 Introduction

Representation of a virtual world (including 3-D models) is not something new in this part of ISO/IEC 14496. For example, ISO/IEC 14772-1:1997 describes an efficient way to represent virtual world. It, however, remains challenging if entities of the virtual world such as 3-D models are to be transmitted or stored due to the large size of data. 3D Mesh Coding in this part of ISO/IEC 14496 is a way to efficiently compress a 3-D model with enhanced network capabilities: computational graceful degradation control, incremental representation, support for non-manifold model, error resilience, and quality scalability via hierarchical transmission of levels of details.

The Topological Surgery (TS) representation is the key element in coding of a 3-D mesh. A 3-D mesh can be thought of as IndexedFaceSet in ISO/IEC 14772-1:1997. And it consists of connectivity, geometry, and properties (e.g. colour, normal, texture coordinate) information. The connectivity information is encoded losslessly, whereas the other information could be quantised before compression. In order to maintain the congruence of the system, geometry and properties information are encoded in a similar fashion. The output of the decoder can be directly used as an IndexedFaceSet node represented in ISO/IEC 14496-1:2001.

O.2 Topological Surgery Representation

The connectivity of a 3D mesh is the relationship among the vertices, edges, and faces of the 3D mesh. In the Topological Surgery representation (TS) the connectivity of a connected component of a manifold 3D mesh is decomposed into *a Simple Polygon* and a *Vertex Graph*.

A non-manifold or non-orientable 3D mesh is first converted to an oriented manifold 3D mesh by cutting through some vertices and edges. One such method is described in reference [10]. The information necessary to reconstruct the original connectivity from the resulting orientable manifold connectivity can be encoded using *Stitches*.

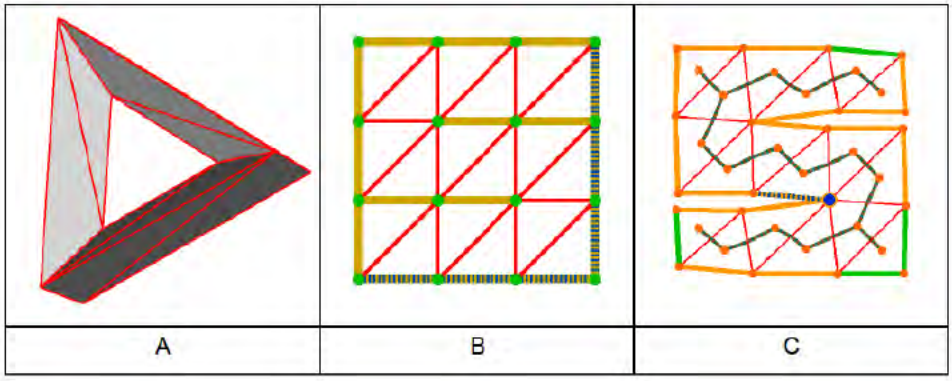


Figure O.1 — Example of Topological Surgery on a torus.
The torus (A) can be obtained from the quadrilateral (B) by identifying boundary vertices. The simple polygon is obtained by constructing a spanning tree in the dual graph of the mesh (C). The boundary of the simple polygon is the polygon loop. The vertex spanning tree is constructed as a spanning tree in the vertex spanning graph (B).

© ISO/IEC 2004 – All rights reserved

The first step of the encoding process, which generates this representation, is to construct a *face forest* spanning the dual graph of the 3D mesh. Two sets of the edges of the 3D mesh can be determined by the face forest: the *face forest edges* and the *vertex graph edges*. The vertex graph edges define the vertex graph spanning the graph of the 3D mesh. All the faces of each connected component of the 3D mesh are connected by face forest edges, resulting in one-to-one correspondence between connected components and trees of the face forest. The simple polygons are defined as the connected components of the 3D mesh resulting from cutting the original 3D mesh through the vertex graph edges. The decoder uses the vertex gr aph to identify how to stitch the 3D mesh back together. Figure O.1 shows an example of this decomposition.

The simple polygons are divided into triangle tree and triangle data pairs. This enables the decoder to incrementally render the model as data for each triangle is received from triangle data bitstream.

A non-manifold 3D mesh is represented as a manifold 3D mesh and a vertex clustering array. The vertex clustering array is represented as a series of stitches operations, with one command per vertex. The encoding process encodes the stitches after the vertex graph is encoded using the vertex clustering array, and optionally using a v_father array. The v_father array provides for each vertex in the vertex graph the index of its predecessor (father) in the directed vertex graph without jumps (potentially the vertex itself if it is a root of the graph without jumps). A pair (vertex, predecessor) is an edge of the 3D mesh.

In the actual representation to be described below, we force the face tree to be binary. We do this, by triangulating polygonal faces, effectively, adding additional edges to the mesh. Since this operation adds extra bits, we avoid unnecessary overheads in the vertex graph, by computing it previously to the triangulation.

O.2.1 Simple Polygon Representation

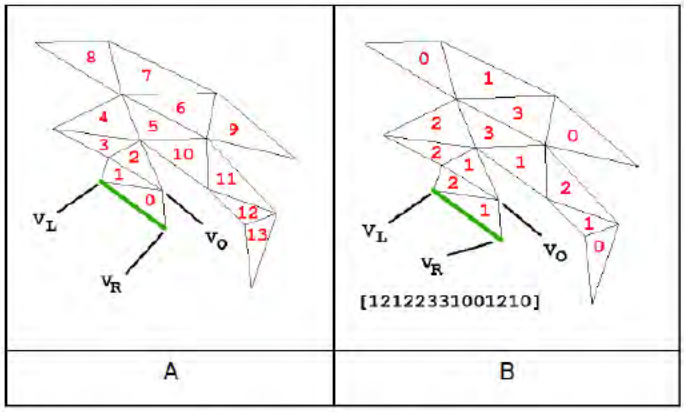


Figure O.2 — Constant-length encoding of a simple polygon. A: Triangles labels according to their order of traversal. B: Triangles labels according to their two bit code. The encoding of the polygon is the sequence between the brackets.

The simple polygon can be represented as a string of two-bit codes by recursively traversing the triangle tree as illustrated in Figure O.2. The traversal starts by entering a first triangle called *root triangle* through a polygon loop edge called the *root edge* (green edge in Figure O.2). One of the two endpoints of the root edge is called the *root vertex* of the simple polygon. The traversal algorithm uses three vertex pointers, the *left vertex* vL, the *right vertex* vR, and the *opposite vertex* vO. The right vertex is initialized to the root vertex, the left vertex to the other endpoint of the root edge, and the opposite vertex to the third vertex of the root triangle. The edge eL=(vL,vO) is called *left edge*, and the edge eR=(vR,vO) is called *right edge*. Each of the two edges is either a boundary edge of the polygon, or internal to the polygon. One bit of the encoding is used to indicate which case, with the bit associated with the left edge first followed by the bit associated with the right edge, and with values 1 for internal, and 0 for boundary. If only the left edge is internal, the traversal algorithm sets vR=vO and continues with the other triangle incident to eL. If only the right edge is internal, the traversal algorithm sets vL=vO and continues with the other triangle incident to eR. If both edges are internal, the traversal algorithm pushes vO and vR onto a traversal stack, it sets vR=vO and continues with the other triangle incident to eL. If both edges are boundary, and the traversal

© ISO/IEC 2004 – All rights reserved

stack is not empty, the traversal algorithm pops vR and vL from the traversal stack, and continues with the other triangle incident to the edge (vL,vR). If both edges are boundary, and the traversal stack is empty, all the triangles of the simple polygon have been visited. For example, in Figure O.2-A, the triangles are labeled with their order of traversal, and in Figure O.2-B, with their corresponding two-bit code, as a number in the range 0,...,3. The encoding of this polygon is [12122331001210].

The simple polygon can also be represented as a *table of triangle runs* in the same way as the vertex tree, except that each triangle run record is composed of only two fields *(tt_run_length, tt_leaf)*. A *Tlast* field is not necessary because the triangle tree is a binary tree, and each triangle run which ends in a branching node must be followed by exactly two runs. The structure of the triangle tree does not completely describe the triangulation of the simple polygon, though. To complete the description, an extra bit per triangle associated with a regular node of the triangle tree must be included. This *sequence of marching patterns* determines how to triangulate the triangle runs by advancing either on the left or on the right on the boundary of the simple polygon.

Note that there is a very simple transformation from the previous representation of the simple polygon to this one. The 3s and 0s in the sequence in the previous representation mark the end of the triangle runs. In the example of Figure O.2, the triangle-runs are defined by the subsequences [121223], [3], [10], [0], and [1210], which are in the same order defined by the table of triangle runs. The tt_run_length field value is the number of two-bit codes in the corresponding subsequence (6,1,2,1, and 4 in this case). The tt_leaf field value is determined by the last code in the subsequence ($3 \rightarrow 0$, $0 \rightarrow 1$), and the marching patterns by the other codes in the sequence, skipping the 3s and 0s ($1 \rightarrow 0$, $2 \rightarrow 1$). In this example the marching edge is the following sequence of bits [010110010]. Clearly, the opposite transformation is equally simple.

O.2.2 Vertex Graph representation

For a connected manifold without boundary, each edge of the vertex graph corresponds to two polygon loop edges (i.e., boundary edges of the simple polygon), but for manifolds with boundary, some edges of the vertex graph (those which correspond to boundary edges of the mesh) correspond to only one polygon loop edge. In the case of a *simple mesh* (manifold without boundary with Euler number 2, i.e., topologically equivalent to a sphere), the vertex graph is a tree. When the manifold mesh is not simple (i.e, when the mesh has boundary edges, or when the Euler number is different from 2), the vertex graph contains loops. If the vertex tree is constructed as a spanning tree by depth first traversing the vertex graph, then one back-edge, or *jump* edge, connects a leaf node of the vertex tree with a previously visited node, forming a loop.

The vertex graph is represented as data structure VG composed of sequence of *runs*. Even though each jump corresponds to a single edge in the vertex graph, we represent a jump by two runs in the VG, one of them having zero length. The first, and true, run starts at the first node the edge gets visited. The second run is generated when we reach the previously visited node, which, then, needs to be considered as a branching node, and the run generated has zero length. Each run record of VG is composed of a variable number of fields. A vertex run connects a leaf or branching node to another leaf or branching node through a path of zero or more regular nodes. The order of traversal of the tree defines a sequential order for the vertex runs, and a first and last node for each run. Each vertex run is represented as a vertex run record composed of the following fields. The vg_last field is a bit that determines if the run shares the first node with the next run or not. The vg_run_length field is an integer with a value equal to the number of edges in the run, not counting the jump edges. The vg_leaf field is a bit which determines if the run ends in a leaf or branching node. Each extended vertex run ending in a leaf node (vg_leaf=1) has an additional vg_loop field, indicating if the last node of the vertex run is connected to another node through a jump edge. Note that each extended vertex runs of length zero, which is always associated with a jump edge, should be considered as ending in a leaf. For these extended vertex runs, first and last node are the same. If vg_loop=1, the record also has a vg_forward_run field indicating whether this run corresponds to the start or end of the run. Each jump edge connects the last node of a run to a previously visited node. The node visited first is the *start* of the jump (vg_forward_run =1), and the other one is the *end* of the jump (vg_forward_run =0). If vg_forward_run =0, the record also has a vg_loop_index field. Every time the start of a jump is encountered, the compression and decompression algorithms push the reference to the jump edge onto an auxiliary *jump stack*. The value of vg_loop_index is the depth of the corresponding jump edge in the jump stack at the time the compression or decompression programs encounter the end of the jump. The reference is immediately removed from the stack, moving subsequent references up, and reducing the total depth of the stack by one (and so reducing the number of bits required to represent the depth of subsequent jump edges). Figure O.3 illustrates some of these concepts.

© ISO/IEC 2004 – All rights reserved

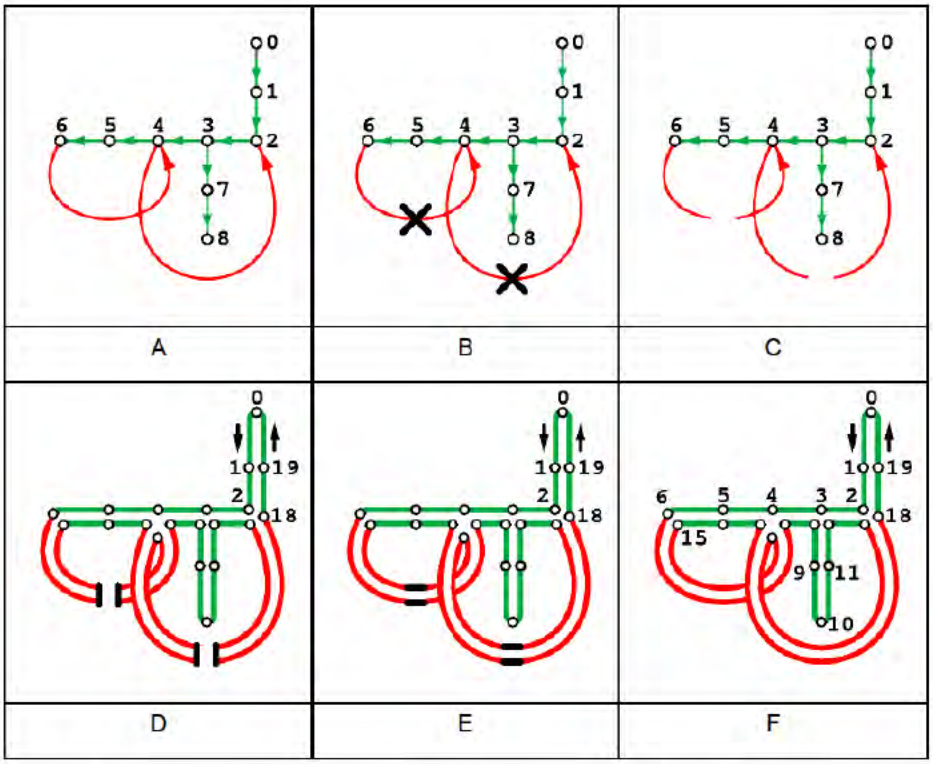


Figure O.3 — Steps to build the bounding loop from the vertex graph.
A : vertex graph decomposed into vertex tree (green) and jump edges (red).
B : extended vertex tree is created by cutting jump edges in half.
C : the extended vertex tree has two leaves for each jump edge.
D : build the extended vertex tree loop.
E : connect the start and end of each jump edge.
F : resulting boundary loop.

O.3 Encoding guidelines for 3D Mesh Coding

O.3.1 Topological Surgery Encoding

A manifold 3D mesh is represented in compressed form as the concatenation of its connected components represented in compressed form. The algorithm to compress a connected component of the connectivity of a manifold 3D mesh can be decomposed into the following steps:

1. Construct the face tree as a spanning tree in the dual graph of the 3D mesh.
2. Define the vertex graph as the graph resulting from deleting the face tree edges from the graph of the mesh.
3. Construct the vertex tree as a spanning tree in the vertex graph.
4. Label each edge of the 3D mesh as a polygon tree edge, a vertex tree edge, or a jump edge.
5. Triangulate the polygonal faces by adding internal non-polygonal edges to the faces.
6. The simple polygon is the mesh resulting from cutting the triangular mesh through the edges of the vertex graph. The triangle tree is the dual graph of the simple polygon.

© ISO/IEC 2004 – All rights reserved

7. Label each triangle tree edge as a polygon tree edge or non-polygon tree edge.
8. Choose an edge of the vertex tree and call it root edge of the vertex tree.
9. Choose an endpoint of the root edge of the vertex tree and call it root vertex of the vertex tree.
10. Of the faces incident to the root edge of the vertex tree, choose an incident face and call it root face of the face tree.
11. Call the edge of the root face of the face tree corresponding to the root edge of the vertex tree, root edge of the face tree.
12. Call the vertex of the root face of the face tree corresponding to the root vertex of the vertex tree, root vertex of the face tree.
13. Encode the simple polygon by depth first traversal starting at the root edge of the simple polygon, with the root vertex of the simple polygon as right vertex, and the other endpoint of the root edge as left vertex. Upon entering a branching triangle, traverse the left branch before the right branch.
14. Encode the vertex graph as a table of extended vertex runs by depth first traversing the extended vertex tree according to the orientation of the simple polygon induced by the order of traversal of the simple polygon.

This encoding procedure is described in further detail in [17].

O.3.2 Support for non-manifolds and Non-orientable manifolds

A non-manifold 3D mesh is represented as an orientable manifold mesh and a vertex clustering array. A non-orientable manifold 3D mesh is considered in the same way as a non-manifold with the inconsistent edges as singular. A first step in the encoding process consists of converting a non-manifold mesh to a manifold mesh and a vertex clustering array. There are several possible methods for performing this operation, one of which being described by Gueziec *et al.* in reference [10].

The Stitches Encoder operates in a manner symmetric to the Stitches Decoder. Its operation is summarized in the pseudo-code of Figure O.4 below.

```
encode(anchor_stack) {
 if (has_stitches == true) {
  encode has_stitches;
  for (i = nV0; i < nV1; i++) {
  // nV0 is the first vertex of the current component, and nV1 –1 is the last vertex
   encode stitch_cmd;
    if (stitch_cmd) {
      encode stitch_pop_or_get;
      if (stitch_pop_or_get) {
        encode stitch_pop;
        encode stitch_stack_index;
         retrieve stitching_anchor from anchor_stack;
        if (stitch_pop) {
          remove stitching_anchor from anchor_stack;
        } // end if
```


© ISO/IEC 2004 – All rights reserved

```
        encode stitch_incr_length;
        if (stitch_incr_length != 0) {
          encode stitch_incr_length_sign;
        } // end if
        decode stitch_push;
        if (stitch_push)
          push i to the back of anchor_stack
        retrieve stitch_length at stitching_anchor;
        total_length =  stitch_length + stitch_incr_length;
        if (total_length > 0)
          encode stitch_reverse;
        save total_length at stitching_anchor;
      } // end if (stitch_pop_or_get)
      encode stitch_length;
      push i to the back of anchor_stack;
    } // end if (stitch_cmd)
  } // end for
}
```

Figure O.4 — Pseudo-code for the Stitches encoder

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 encoder encodes a stitching command from the bitstream. The boolean value **stitch_cmd** is encoded. If the boolean value **stitch_cmd** is true, then the boolean value **stitch_pop_or_get** is encoded; if the boolean value **stitch_pop_or_get** is false, an unsigned integer **stitch_length** is encoded. The current vertex i is then pushed to the back of the anchor_stack.

if **stitch_pop_or_get** is true, then the boolean value **stitch_pop** is encoded, followed with the unsigned integer value **stitch_stack_index**. Using **stitch_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 **stitch_pop** boolean variable is true, then the stitching_anchor is removed from the anchor_stack. Then,an integer **stitch_incr_length** is encoded as an unsigned integer. If it is different from zero, its sign (boolean **stitch_incr_length_sign**) is encoded. A **stitch_push** boolean value is encoded. If **stitch_push** 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 **stitch_incr_length**; if total_length is greater than zero, then a **stitch_reverse** boolean value is encoded. Finally, the value total length is saved at (associated with) the stitching_anchor.

O.3.3 Support for Error Resilience

To support error resilience, data partitioning is very important. Component based data partitioning (CODAP) provides a new functionality to the TS method. CODAP provides a way to implicitly divide a large 3D mesh into multiple partitions. A partition may include part of a connected component or multiple connected components. Since vertex graph is relatively small compared to simple polygon, the division of connected component is only applied to simple polygon. However, the correspondence between vertex graph (vg) and simple polygon is kept through the whole coding process. There are three different partition types, which reflect different combinations of vertex graphs and simple polygons. In front of each partition, a start code is provided for resynchronization.

© ISO/IEC 2004 – All rights reserved

In order to make a partition into a given size, traversal order determination and partitioning methods are used. The traversal order information is coded with particular branching triangles. Virtual leaf triangle information is inserted at the end of each divided triangle tree to make a virtually complete triangle tree.

If channel errors hit the bit stream, some partitions will be corrupted or lost. However, other partitions will be decoded with its simple polygon and corresponding vg. Forward error correction may be used to ensure the vg is protected.

O.3.3.1 Traversal Order Determination

In TS, the reconstruction procedure is as follows: 1) building the bounding loop table, 2) computing the boundary offset for each branch, and 3) reconstructing triangles and polygons.

In basic TS coding, the traversal direction is always clockwise or left most search. In the case that the left branch is much larger than the right branch of the triangle tree, the decoder has to wait until it decodes the vertices all the way down the left branch before decoding and rendering the smaller right branch. See Figure O.5. However in the CODAP technique, in order to avoid unnecessary delay, the encoder will choose the shorter (left or right) branch. A one-bit flag will be sufficient to inform the decoder of the traversal order of the triangle tree.

In Figure O.5, the information of the traversal order is given at a branch on a main stem of the partitioned tree. The green branching triangle has the right first traversal order and the red branching triangle has the left first traversal order. Within a branch, the traversal order is always clockwise, as shown with the purple line.

me_1	me_2	...	td_orientation $_1$	...	me_i	td_orientation$_N$	...	me_n

The triangle data contains triangle marching edge (me) information where there is a triangle run. However, the triangle data contains the **td_orientation** fields where there is a branch in the main stem of the binary triangle tree.

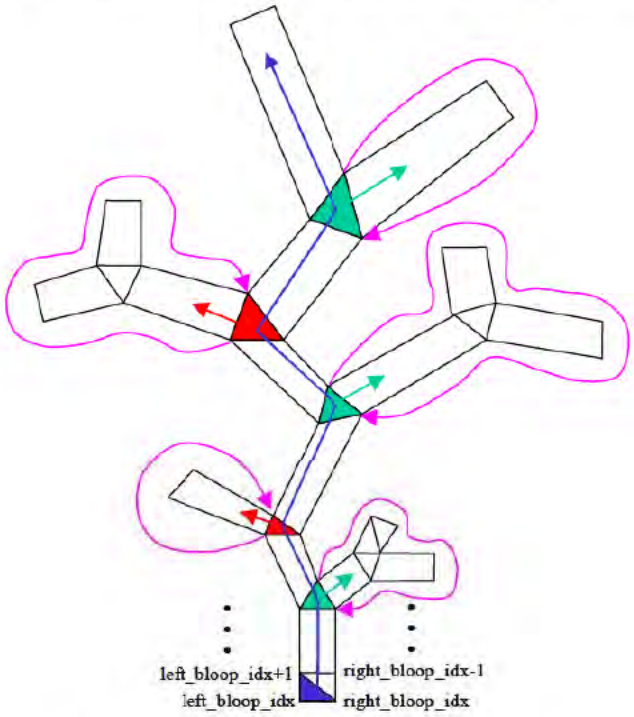


Figure O.5 — Determination of traversal order

© ISO/IEC 2004 – All rights reserved

O.3.3.2 Partitioning

One way to handle the Y-vertex while maintaining independence between partitioned simple polygon data is to make the partitioned data as a virtual connected component by inserting virtual bits, as shown in Figure O.6. There are two main cases for defining the virtual bits. When the partition ends with a run or a leaf triangle, a pair of virtual bits (1,1) is added to the (tt_run_length, tt_leaf) pair. When the partition ends on a branching triangle, two pairs of virtual bits (1,1) are added to the (tt_run_length, tt_leaf) pair. Also, if the partition ends on a branching triangle, the length of the branch (codap_branch_len) is coded so that Y-vertex index can be calculated.

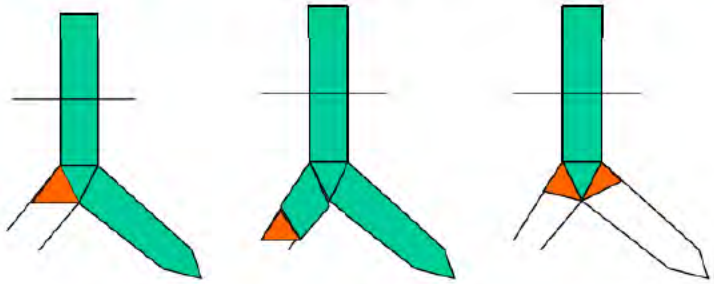


Figure O.6 — Examples of forming virtual connected component by inserting virtual bits to make virtual triangles (marked in red). The partitioning was done at : (a) a leaf (b) a run (c) a branch.

When the simple polygon data is partitioned and formed as virtual connected components, the decoder can find out if the virtual bits exist by checking the following condition:

codap_right_bloop_idx – codap_left_bloop_idx – 1 > total number of triangles decoded in tt.

If this condition is true, then virtual bits exist. To find out if there are one or two pairs of virtual triangles, the third triangle from the end is checked to see whether it is a branch or not. If it is a branch, then the partition ended on a branch and two virtual triangles are added. Otherwise, the partition ended on a leaf or a run and only one virtual triangle is added at the end. However, if the leaf is the only triangle in the subtree in Figure O.6 (a), encoders shall not make a partition at this point. This is to avoid a conflict with Figure O.6 (c).

Also the extended boundary prediction mode shall be used for the vertices in the partition of the subtree unless every vertex in the partition of the subtree is not used in the partitions after the subtree.

In topological surgery, to code the mesh information containing polygons, the polygons are first divided into triangles. Figure O.7 shows an example for this case.

© ISO/IEC 2004 – All rights reserved

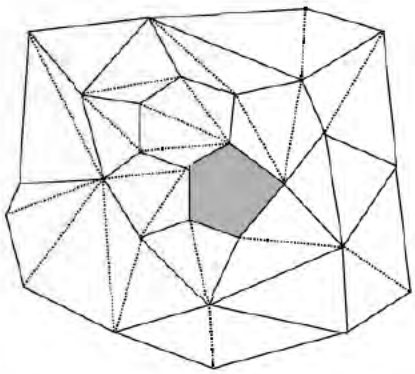


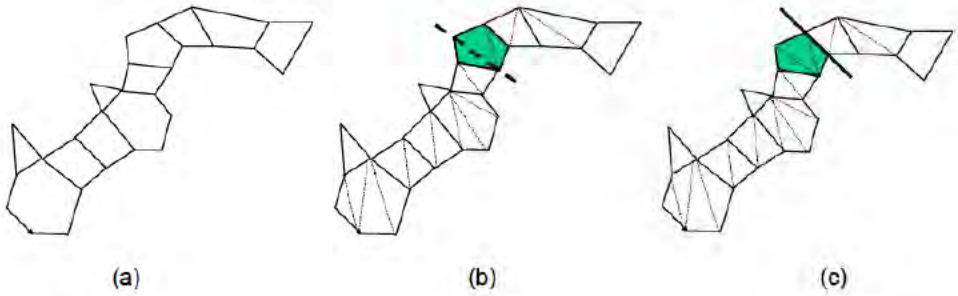
Figure O.7 — Example of dividing 3D mesh into triangle mesh : the solid lines are the original 3D mesh edge, and the dotted lines are the imaginary edges added for the division of polygon into triangles.

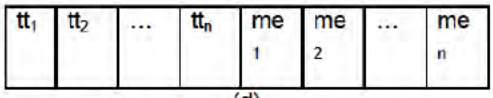
When polygons are divided into triangles, imaginary edges are added. Therefore, this information must be sent to the decoder to reconstruct the polygons. This information is called polygon edge. So when a polygon edge information is sent for each triangle, value 1 is sent for a real edge, and 0 is sent for an imaginary edge. However, if 3D mesh does not exist (only triangle meshes : triangulated==1), then polygon edge information is not needed.

In the topological surgery method, every polygon in the mesh is divided into triangles and polygon edge information is produced by the number of triangles. All of this information is coded, transported and stored. However, since the first triangle must start with a real edge, the corresponding polygon information will always be 1, and so it is not coded. In this case, the decoder assumes that the polygon edge information for the first edge is 1.

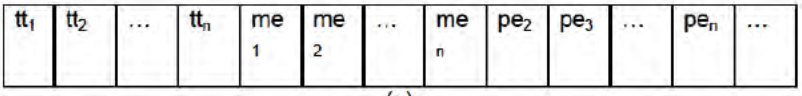
When data partitioning is performed, the partitioning may be done at the imaginary edge inside the polygon, as in Figure O.8 (b). In this case, it cannot be assumed that the polygon edge of a data partition starts with 1. Therefore the polygon edge must be coded by the number of triangles, except for the simple polygon data in partition type 0, which is described in the next section.

© ISO/IEC 2004 – All rights reserved

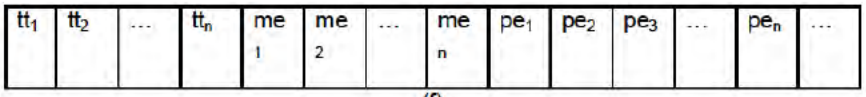




(d)



(e)



(f)

Figure O.8 — Example of partitioning on 3D mesh and the related syntax : tt is the triangle run information, me is the marching edge information, pe is the polygon edge information, annex at the bottom shows the order of tm and pe corresponding to tt, n is the number of triangles. (a) the original 3D mesh, (b) partitioning inside a polygon, (c) partitioning at the boundary edge(real edge) of two polygons, (d) when triangulated == 1(no 3D mesh) (e) when 3D mesh exists and partitioning is only allowed at the real edge, (f) when 3D mesh exists and partitioning is allowed at the imaginary edge.

On the other hand, if there exists a polygon with many sides (edges) but it is partitioned only at a real edge, the size of a partition may not come into the range of target size. So when the partition inside the polygon is allowed, it is possible to achieve the size that within a small range of target size. Also, allowing the partition inside the polygon makes the method more generalized.

O.3.3.3 Boundary Prediction

Once a connected component is partitioned into pieces, prediction of geometry coding (as well as other attributes that use predictive coding) should allow independent decoding between partitions.

There are two modes of coding vertices on the boundary between partitions: restricted boundary prediction mode and extended boundary prediction mode.

Confer the details of both modes in subclause 7.13.5.3

When partitioning is done at a branching triangle, it is recommended to use extended boundary prediction mode for the first branch from the branching triangle.

© ISO/IEC 2004 – All rights reserved

O.4 Encoder considerations for efficient compression of Vertex Properties

Given a geometry sample (x, y, z), the quantised geometry sample (qx,qy,qz) is obtained as:

xq = floor((x – **coord_xmin**)/**coord_size***((1<<**coord_quant**)-1)+0.5)

yq = floor((y – **coord_ymin**)/**coord_size***((1<<**coord_quant**)-1)+0.5)

zq = floor((z – **coord_zmin**)/**coord_size***((1<<**coord_quant**)-1)+0.5)

The procedure for colors and texture coordinates is the same.

A higher-order geometrical prediction scheme is employed in coding of vertex positions, color properties, texture positions, and normal vectors. The encoder transmits **pred_type,** the information of prediction type, to the decoder, as shown in Table O.1.

Table O.1 — Meaning of pred_type

pred_type	Prediction type
00	No_prediction
01	Tree_prediction
10	Parallelogram_prediction
11	reserved

For example, if the **pred_type** indicates the prediction type is tree-prediction or parallelogram, the predicted value($\tilde{p}$) can be expressed as shown in Eq. O.x.1.

$$\tilde{p} = \sum_{i=1}^{N} \lambda_i p_i \qquad \text{(Eq. O.x.1)}$$

The p_i's in Eq. O.x.1 represent the vertex properties of the adjacent vertices. The encoder transmits the each λ_i and N to the decoder.

However, the vertex property values such as colors, positions, normal vectors, and texture positions along the vertex traversal sequences has low correlation between successive vertices, it degrades the prediction accuracy. In the case of low correlation sequences, the prediction scheme can be limited to the simple 1st-order prediction. The simple 1st-order prediction scheme can be easily implemented to the encoder and decoder by fixing the λ_i's, as shown in Eq. O.x.2.

$$\lambda_i = \begin{cases} 1 & \text{, where } \ i = 1 \\ 0 & \text{, otherwise} \end{cases} \qquad \text{(Eq. O.x.2)}$$

Table O.2 shows the weighting factors of the prediction schemes. The higher-order prediction mode is the general parallelogram predictor, while the simple 1st-order prediction mode is the modified encoding scheme. The simple 1st-order prediction mode shows better performance for color property coding.

Table O.2 — weighting factors of predictor

Prediction scheme	N	λ_i	Coding efficiency
parallelogram	3	1, 1, -1	-
Simple 1st-order prediction	L (> 0)	1, 0, … , 0	better for color encoding

© ISO/IEC 2004 – All rights reserved

O.5 Progressive Forest Split Representation

In the progressive forest split (PFS) compression scheme, a 3D model is represented as a low resolution model 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 and property displacements. 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. The representation, encoding scheme, simplification algorithm, encoding and decoding methods are described in [8].

The general structure of a simplification scheme to generate the progressive representation, and the associated level of detail hierarchy, from a single resolution 3D mesh, is described as well. This method may require additional compression parameters, which may be tuned interactively for each transition from one level of detail to the next one.

O.5.1 Encoding the Forest

A simple encoding of the forest requires one bit per edge, for example a value 1 for the edges belonging to the forest and 0 for the rest. These bits are placed in the compressed data stream in the edge index order defined above. However, since any subset of edges of a forest form a forest, the edges with bit value 1 read up to a certain point may determine that certain edges with higher edge index should have bit value 0, otherwise they would create loops in the forest defined so far by the bits with bit value 1. These edges can be skipped in the compressed data stream. Arithmetic encoding of the bit-stream results in fewer bits. In the experiments that we have run so far, where the number of triangles increase by 50-70% with each forest split operation, the skipping of predictable bits described above trims the length of the forest bit-stream by about 1-8%, and the simple encoding of the resulting bit-stream is usually shorter than the run-length encoded stream. A pseudo-code implementation of the method to determine which edges to encode is as follows:

```
/ forest encoder
// 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  = (e.isInForest)?true:false;
 if(p.find (i)!=p.find(j)) {
  if(edge_bit==true)
   p.join(i,j);
 } else {
  // forest => edge_bit==false
  skip_edge = true;
```

© ISO/IEC 2004 – All rights reserved

```
  }
  if(skip_edge==false)
    encode_bit(edge_bit);
}
```

Figure O.9 — forest encoder pseudo-code

O.5.2 Support for meshes with polygonal faces

The decoder process supports polygonal faces for the base mesh, as well for the simple polygons used to fill the tree loops.

O.5.3 Method for generating of a PFS Representation of a Triangular 3D Mesh

The method developed in this section takes as input a 3D mesh, where all polygons are triangles and generates a series of levels of detail that can be encoded as a series of PFS operations. The implementation included in the reference software takes as an input a restricted VRML single-resolution file format (described in the Core Experiments document) and returns a Clustered multi-resolution file format (extension ".cmr" as described in the Core Experiments document). Although a Clustered multi-resolution file format can represent any hierarchy of levels of detail, the output of our method is a particular hierarchy corresponding to a base mesh and a succession of PFS refinements. The method is based on applying a succession of edge collapses to the original mesh. The edge collapse operation is described in the Core Experiments document, in the section dedicated to the Progressive Vertex Split method: it consists of merging the two endpoints of an edge, and thereby eliminating two triangles from the mesh (one triangle if the edge belongs to the mesh boundary). The method of the reference implementation consists of entering all the edges of the mesh in a priority queue that is keyed with the length of the edge. Then, in succession the edges are removed from the queue, and collapsed provided that they pass a series of tests. The tests are of two types: "topological" tests and "geometric" tests. Topological tests are performed first, as they are expected to be less costly than the geometric ones. If the tests are all passed, the edge is removed from the queue, and all neighboring edges are removed and re-entered in the queue that should reflect their new lengths. Before the geometric tests are performed a decision is made on how to position the vertex resulting from the edge collapse. Depending on the user's choice, this position can be either edge endpoint or the midpoint of the edge. In the sequel, the vertex resulting from the edge collapse is referred to as "simplified vertex". Figure O.10 is a diagram illustrating the algorithm.

```
while queue not empty
    take shortest edge
    if edge can be collapsed, e.g.,
        (valence does not exceed maximum &&
        mesh is still manifold after operation &&
        all collapses correspond to pfs &&
        triangle rotation is acceptable &&
        triangle compactness is within bounds)
      remove all edges in the star from the queue
      insert new edges
    else /* collapse of edge is not possible */
      simply remove edge from queue
```

Figure O.10 — Algorithm for generating PFS representation by collapsing edges of a 3D mesh

© ISO/IEC 2004 – All rights reserved

O.5.4 Topological Tests

The purpose of the topological tests is to ensure that the mesh is still a manifold after collapsing a given edge and that the collection of triangles that were removed are such that the dual graph of this collection is a forest. The first test verifies that the valence of the vertex resulting from the edge collapse, or simplified vertex, is at east 3 if it is not a boundary vertex. The second test ensures that all the vertices incident to the simplified vertex are different (otherwise we would have a non-manifold mesh). The third test verifies that the collection of triangles that were removed by collapsing edges form a forest as well as a manifold mesh. Both properties are checked by means of a comprehensive test. To keep track of the triangles that are eliminated, we introduce a color coding of the triangles as follows: blues triangles are the ones eliminated during edge collapses; red triangles are the ones remaining, purple triangles are the triangles that would be eliminated (would become blue) if the edge collapse under current testing would be performed. After several collapses are performed, an edge in the simplified surface can correspond to a pair of non-adjacent edges in the original surface (this is quite easy to see, for instance when collapsing in sequence two adjacent edges). It follows that the triangles that are removed during the collapse (purple triangles) are not necessary adjacent. Two cases are distinguished, depending on whether the purple triangles are adjacent or not. In the first case, the two purple triangles, which we refer to as T0 and T1, are adjacent. We use the notations of Figure O.11. We first test whether T0 and T1 are adjacent to 3 or more blue triangles, in which case the test fails. Otherwise, if T0 and T1 are adjacent to 0 or 1 blue triangle, we consider the vertices among V0, V1, V2 and V3 (shown in Figure O.11) that are not incident to a blue triangle that is adjacent to either T0 or T1. If there are incident to any blue triangle the test is failed (Figure O.11B), and otherwise the test is passed. Otherwise, if T0 and T1 are adjacent to exactly two blue triangles, we first check whether the two adjacent blue triangles are incident to a common vertex, if which case the test is failed. Otherwise, the two adjacent blue triangles must belong to a different connected component of blue triangles in order to pass the test (Figure O.11A). The connected components of triangles are maintained using a partition of the triangles.

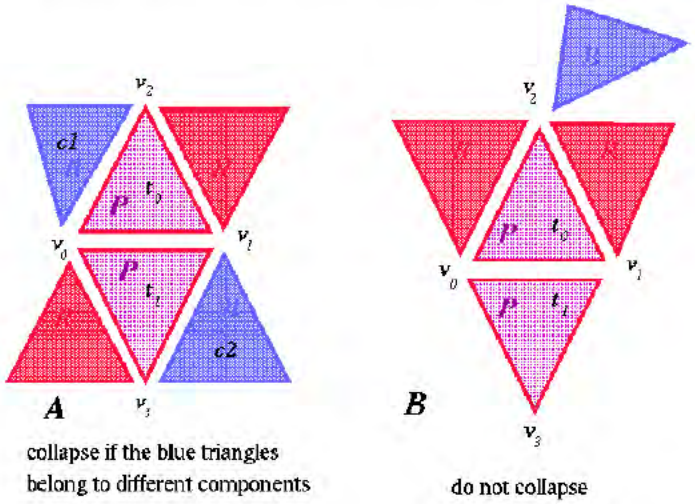


Figure O.11 — Additional edge collapsibility tests necessary to obtain a PFS representation. The cases illustrated here apply when the two triangles of the original surface that are removed by the edge collapse are adjacent

The second case applies when the two purple triangles T0 and T1 are not adjacent. This case is illustrated in Figure O.12. We first test whether Vertex V2 is a boundary vertex. If yes, T0 must be a boundary triangle otherwise the test fails (Case A in). If T0 is a boundary triangle, then the test is the same as when V2 is not a boundary vertex (Cases B and C in Figure O.12): all triangles incident on V2 must be red. The same sequence of tests is applied to Vertex V4: if Vertex V4 is a boundary vertex, T1 must be a boundary triangle otherwise the test fails (Case A in). If T1 is a boundary triangle, then the test is the same as when V4 is an interior vertex (Cases B and C in): All triangles incident on V4 must be red.

© ISO/IEC 2004 – All rights reserved

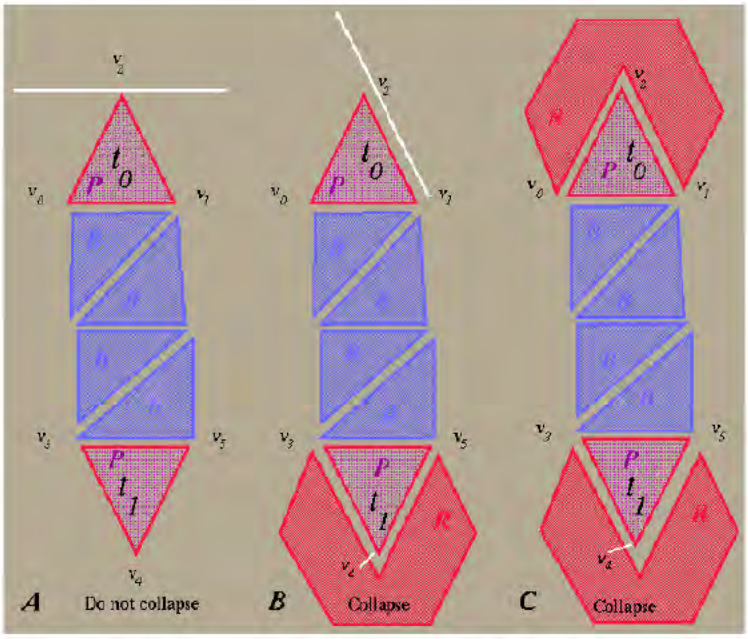


Figure O.12 — Additional edge collapsibility tests necessary to obtain a PFS representation. The cases illustrated here apply when the two triangles of the original surface that are removed (the purple triangles T0 and T1) are not adjacent. Boldface letters inside triangles indicate colors. A: do not collapse. B, C: collapse.

O.5.5 Geometric Tests

The geometric tests described in this section are not compulsory in order to represent the level of detail hierarchy in PFS. However, they were experimentally useful to produce levels of detail that are visually satisfactory. This part of the algorithm is most open for changes and improvements by parties implementing an encoder. The first geometric test of the sample implementation verifies that the normals of the triangles incident to any vertex of the collapsed edge do not rotate by more that a user-specified angle during the collapse. The user can specify this maximum angle. The second geometric test verifies that the minimum compactness values of the triangles incident to any endpoint of the collapsed edge, before the edge collapse (C0) and after the edge collapse (C1) satisfy the following relation: C1 > r C0, where r is a user-specified ratio. The compactness of a triangle is defined using the above formula, $c = \frac{4\sqrt{3}a}{l_0^2 + l_1^2 + l_2^2}$, where a is the area of the triangle and the l_i are the three side lengths.

O.6 Complexity estimation for Computational Graceful Degradation

Computational Graceful Degradation (CGD) is the process of gracefully degrading the quality of the rendering of the 3D scene in case the 3D rendering engine has not enough processing power for performing the rendering in real-time. In this case, some lower-quality rendering processes can be performed instead, as for example flat shading instead of smooth shading, mesh-rendering with a smaller number of triangles, … The implementation of specific CGD algorithms which provide the highest complexity reduction for the lowest image degradation is an implementation dependent issue which has no normative implications.

For being able to apply Computational Graceful Degradation, each specific 3D rendering engine should be able to estimate its required rendering processing power, before the 3D rendering is actually started. The encoder must therefore transmit, as soon as possible, some complexity estimation (ce) characteristics of the 3D objects, which are relevant to 3D rendering process. The definition of these complexity parameters is normative. These parameters are defined as follows:

© ISO/IEC 2004 – All rights reserved

1. Number of vertices
2. Number of triangles
3. Number of edges
4. Projected surface of the object in any viewpoint

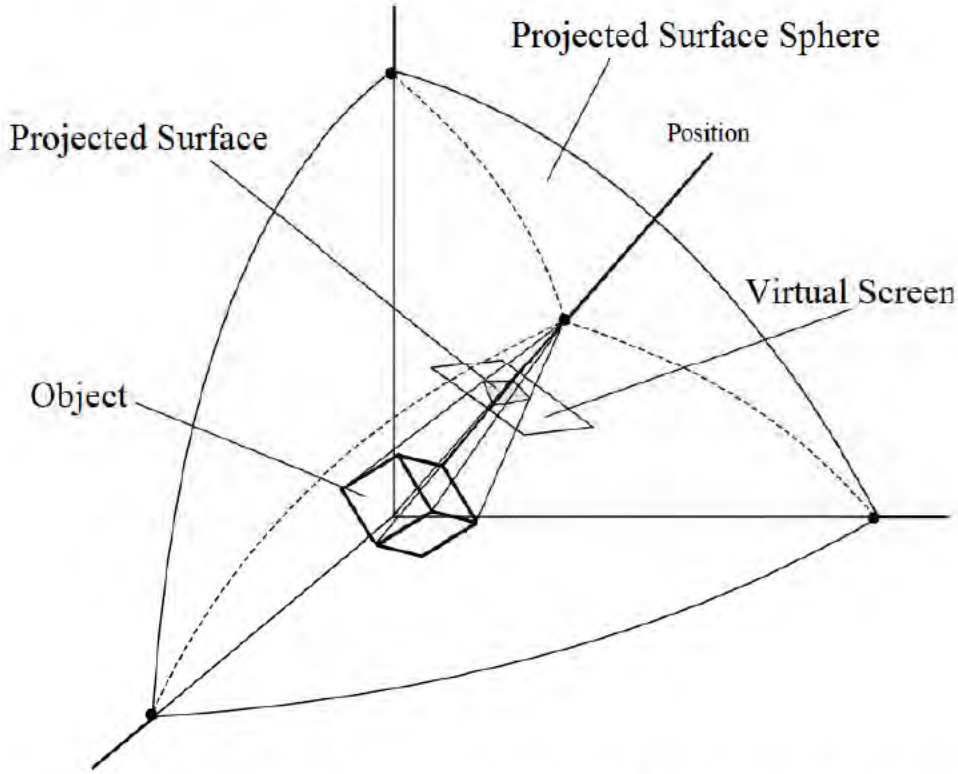


Figure O.13 — Projected Surface Sphere constructed by projecting the pixels of the object onto a virtual screen for different viewing angles

The projected surface is the number of pixels that are actually projected onto the screen (called the virtual screen) during the rendering of the object in that viewing position (see Figure O.13). This projected surface largely varies with orders of magnitude, depending on the viewing angle and viewing distance to the object, which can be modified by the user, without any encoder interaction. In principle, the projected surface should be transmitted:

— for any position around the object, with constant distance to a reference point of the object (e.g. gravity point), creating the so-called Projected Surface Sphere (see Figure O.13).

— for one particular viewing angle, in a variety of distances to the reference point.

However, simulations have shown (see [9]) that the projected surface should only be measured in a limited number of positions around the object, from which the projected surface for any other position can be recovered - within an acceptable error bound - by linear interpolation or scaling. Linear interpolation is used within a Projected Surface Sphere with constant radius, while scaling is required for extrapolating (interpolating) information from one (different) Projected Surface Sphere(s) to another Projected Surface Sphere. For better accuracy, we advise to apply the following scaling formula, using one particular Projected Surface Sphere:

© ISO/IEC 2004 – All rights reserved

$$p_i = \frac{R_{ref}}{R_i}\left(p_{ref} - \min_{ref}\right) + \min_i$$

in which:

p_i is the estimated projected surface in a specific direction for radius R_i.

p_{ref} is the measured projected surface in that specific direction for radius R_{ref}.

R_{ref} is the radius of the reference Projected Surface Sphere.

R_i is the radius of the estimated Projected Surface Sphere.

min_{ref} is the minimum projected surface for the reference Projected Surface Sphere.

min_i is the minimum projected surface for the estimated Projected Surface Sphere.

The accuracy can even be increased by allowing some additional measured Projected Surface Spheres (thus at different radii). Typically, only a very limited points will be added for each additional Projected Surface Sphere.

A 1-bit switch should indicate in the 3D_Mesh_Object_Header whether Computational Graceful Degradation is enabled or not. This bit is called the ce_SNHC. If the ce_SNHC is set to off, no additional information with respect to complexity estimation is sent through the bitstream. Otherwise, the Projected Surface Sphere(s) is (are) sent, from which the required processing power for 3D rendering of the 3D object can be estimated. Since the projected surface does not drastically change with the 3D mesh resolution, it is specified in the 3D_Mesh_Object_Header for all layers, instead of being specified in 3D_Mesh_Object_Layer.

For each 3D_Mesh_Object_Layer, information regarding the number of vertices, triangles and edges should be transmitted if the ce_SNHC is enabled in 3D_Mesh_Object_Header. Transmitting the number of vertices, triangles and edges at the level of the 3D_Mesh_Object_Layer has the advantage that the 3D rendering estimation can be performed at any 3D mesh resolution.

O.7 QoS for SNHC through upstream

Defining profiles and levels in SNHC has proved to be considerably difficult, since processing power required for decoding and rendering the 3D content varies with orders of magnitudes in time, while additionally the performance of decoders is highly dependent on the target platforms (software versus hardware implementations, very powerful graphics cards, …).

Figure O.14 shows how this can be resolved by using Computational Graceful Degradation techniques in the decoder.

However, this does not resolve the problem of profile and level definitions, in which basically the data that is transmitted should comply to some predefined thresholds (equivalent to "compliant bitstreams"). Therefore, we provide an upstream, which syntax conveys some information that reveals to the encoder (server) the performances of the decoder (see Figure O.14). This basically means that the decoder determines its capabilities in advance (one-shot calibration process), transmits this information to the encoder, which will "restrict" its data to the capabilities of the decoder (e.g. by mesh simplification). The data to be transmitted through the upstream is defined in subclause 6.3.12.3 (upstream_SNHC_QoS).

The decoder performances are measured by decoding and rendering different SNHC objects and measuring the frame rate in different configurations, i.e. a varying number of triangles, a varying number of pixels projected on the screen and different rendering modes. The performance information is transmitted using different performance curves. Each performance curve corresponds to the frame rate achieved for one particular setting of the number of triangles and rendering mode, but for a varying number of screen coverages.

© ISO/IEC 2004 – All rights reserved

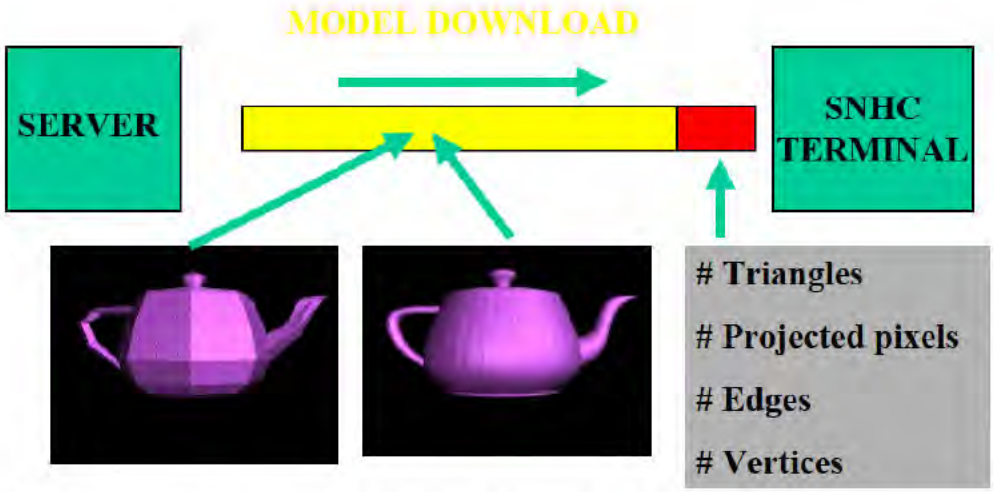


Figure O.14 — Downloading complexity estimation (Computational Graceful Degradation) parameters to the decoder

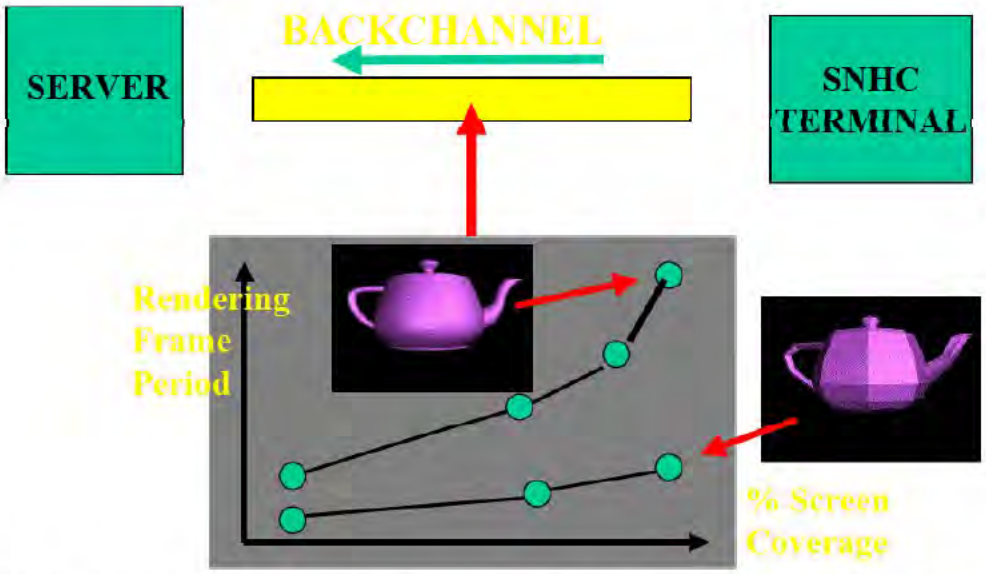


Figure O.15 — Uploading terminal (renderer) performance figures to the server, which will in turn transmit the data model at the correct Level of Detail for enabling real-time rendering at the decoder

O.8 References

[1] W.B. PENNEBAKER and J. MITCHELL, "*JPEG Still Image Data Compression Standard*," New York: Van Nostrand Reinhold, 1993

[2] JIANKUN LI, JIN LI and C.-C. JAY KUO, "An Embedded DCT Approach to Progressive Image Compression, ICIP'96", *IEEE International Conference on Image Processing*, Vol. 1

© ISO/IEC 2004 – All rights reserved

[3] David TAUBMAN and Avideh ZAKHOR, "Multirate 3D Sub-band Coding of Video", *IEEE Transactions on Image Processing*, Vol 3.,No. 5, September 1994

[4] G. TAUBIN, "A Signal Processing Approach To Fair Surface Design", *Siggraph'95 Conference Proceedings*, pp 351-358, Los Angeles, August 1995

[5] G. TAUBIN and J. ROSSIGNAC, "Geometry Compression through Topological Surgery", *ACM Transactions on Graphics*, pp 84-115, April 1998

[6] G. TAUBIN, W.P. HORN, F. LAZARUS and J. ROSSIGNAC, "Geometric Coding and VRML", *Proceedings of the IEEE, Special Issue on Multimedia Signal Processing*, July 1998

[7] G. TAUBIN, W.P. HORN and F. LAZARUS, "*The VRML Compressed Binary Format*", 1997. http://www.research.ibm.com/vrml/binary

[8] G. TAUBIN, A.GUEZIC, W.P. HORN and F. LAZARUS, "Progressive Forest Split Compression", *Siggraph'98 Conference Proceedings*, pp. 123-132, Orlando, July 1998

[9] Gauthier LAFRUIT, Lode NACHTERGAELE, Kristof DENOLF and Jan BORMANS, "3D Computational Graceful Degradation", *The 2000 IEEE International Symposium on Circuits and Systems*, Geneva, Switzerland, May 2000

[10] A. GUEZIEC, G. TAUBIN, F. LAZARUS and W.P. HORN, "Converting Sets of Polygons to Manifold Surfaces by Cutting and Stitching," in *Visualization'98 Conference Proceedings*, Research Triangle Park, NC, October 1998

[11] A. GUEZIEC, G. TAUBIN, F. LAZARUS and B. HORN, "A Framework for Streaming Geometry in VRML", *IEEE Computer Graphics & Applications*, special issue on VRML, March-April 1999

[12] A. GUEZIEC, F. BOSSEN, G. TAUBIN and C. SILVA, "Efficient Compression of Non-Manifold Polygonal Meshes", IEEE Visualization'99, 1999

[13] A. GUEZIEC and G. TAUBIN, "A Framework for Memory-Efficient Levels of Detail", *Tenth Canadian Conference on Computational Geometry*, Montreal, Canada, August 1998

[14] G. TAUBIN and J. ROSSIGNAC, "Course on 3D Geometry Compression", *Siggraph'98*, Orlando, FL, July 1998

[15] G. TAUBIN and J. ROSSIGNAC, "Course on 3D Geometry Compression", *Siggraph'99*, Los Angeles, CA, 1999

[16] G. TAUBIN, W. HORN and P. BORREL, "Compression and Transmission of Multi-Resolution Clustered Meshes", *IBM Technical Report* RC-21398, February 1999

[17] F. BOSSEN, "*On the Art of Compressing Three-dimensional Polygonal Meshes and their Associated Properties*", Thesis no. 2012, Ecole Polytechnique Federale de Lausanne, September 1999

© ISO/IEC 2004 – All rights reserved

Bibliography

[1] Arun N. NETRAVALI and Barry G. HASKELL, "*Digital Pictures, representation and compression*", Plenum Press, 1988

[2] Barry G. HASKELL, Atul PURI and Arun N. NETRAVALI, "*Digital Video: An Introduction to MPEG-2*", Chapman & Hall, ISBN 0-412-08411-2, 1997

[3] F. I. PARKE and K. WATERS, *Computer Facial Animation*, A K Peters, Wellesley, MA, USA, 1996

ISO/IEC 11172-1:1993, *Information technology — Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s — Part 1: Systems*

ISO/IEC 11172-3:1993, *Information technology — Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s — Part 3: Audio*

ISO/IEC 13818-3:1998, *Information technology — Generic coding of moving pictures and associated audio information — Part 3: Audio*

IEC 60461:1986, *Time and control code for video tape recorder*

ITU-T Recommendation T.81 (1992) | ISO/IEC 10918-1:1994, *Information technology — Digital compression and coding of continuous-tone still images: Requirements and guidelines*

© ISO/IEC 2004 – All rights reserved

Annex P
(normative)

Data mapping format of SMPTE12M (time code)

The 64bit contents of time_code_smpte12m in Group of Studio Video Object Plane header and Studio Object Plane header is mapped as follows:

Syntax	Bits	Mnemonic
Time_code[63..0]{		
Color Frame Flag	1	bslbf
Drop Frame Flag(NTSC)/Unused(PAL)	1	bslbf
TV Frame Tens	2	uimsbf
TV Frame Units	4	uimsbf
Field Phase(NTSC)/Binary group flag 0(PAL)	1	bslbf
TV Seconds Tens	3	uimsbf
TV Seconds Units	4	uimsbf
Binary group flag 0 (NTSC)/Binary group flag 2(PAL)	1	bslbf
TV Minutes Tens	3	uimsbf
TV Minutes Units	4	uimsbf
Binary group flag 2 (NTSC)/ Field Phase (PAL)	1	bslbf
Binary group flag 1 (NTSC)/Binary group flag 1(PAL)	1	bslbf
TV Hours Tens	2	uimsbf
TV Hours Units	4	uimsbf
2nd Binary Group	4	uimsbf
1stBinary Group	4	uimsbf
4th Binary Group	4	uimsbf
3rd Binary Group	4	uimsbf
6th Binary Group	4	uimsbf
5thBinary Group	4	uimsbf

© ISO/IEC 2004 – All rights reserved

8th Binary Group	4	uimsbf
7thBinary Group	4	uimsbf
}		

NOTE If the definition of SMPTE12M is modified, time code in this specification shall also be modified in the same manner.

© ISO/IEC 2004 – All rights reserved

Annex Q
(informative)

Syntax correspondence of MPEG-2 4:2:2 Profile to MPEG-4 Studio Profile

This informative annex describes syntax correspondence of the MPEG-2 4:2:2 Profile to the MPEG-4 Studio Profile. This Annex provides useful information for bitstream translation.

Q.1 Sequence header

MPEG-2 4:2:2 Profile syntax	*MPEG-4 Studio Profile*		*Note*
	layer	*syntax*	
horizontal_size_value	StudioVideoObjectLayer()	**video_object_layer_width**	MPEG-2:12bits + 2bits (**horizontal_size_extension** in sequence_extension()) MPEG-4:14bits
vertical_size_value	StudioVideoObjectLayer()	**video_object_layer_height**	MPEG-2:12bits + 2bits (**vertical_size_extension** in sequence_extension()) MPEG-4:14bits
aspect_ratio_information	StudioVideoObjectLayer()	**aspect_ratio_info**	
frame_rate_code	StudioVideoObjectLayer()	**frame_rate_code**	
bit_rate_value	StudioVideoObjectLayer()	**first_half_bit_rate, latter_half_bit_rate**	MPEG-2:18bits+12bits (**bit_rate_extension** in sequence_extension()) MPEG-4:15bits+15bits
vbv_buffer_size_value	StudioVideoObjectLayer()	**first_half_vbv_buffer_size, latter_half_vbv_buffer_size**	MPEG-2:10bits+8bits (**vbv_buffer_size_extension** in sequence_extension()) MPEG-4:15bits+3bits
constrained_parameters_flag	-	-	used in ISO/IEC 11172-2 (MPEG-1)
load_intra_quantiser_flag	-	-	
load_non_intra_quantiser_flag	-	-	

© ISO/IEC 2004 – All rights reserved

Q.2 Extension and user data

Extension data and user data syntax of MPEG-2 are included in the MPEG-4 Studio Profile.

Q.3 Sequence extension

MPEG-2 4:2:2 Profile syntax	*MPEG-4 Studio Profile*		*Note*
	layer	*syntax*	
profile_and_level_indication	VisualObjectSequence()	profile_and_level_indication	
progressive_sequence	StudioVideoObjectLayer()	**progressive_sequence**	
chroma_format	StudioVideoObjectLayer()	**chroma_format**	
horizontal_size_extension	-	-	See **horizontal_size_value** in sequence header
vertical_size_extension	-	-	See **vertical_size_value** in sequence header
bit_rate_extension	-	-	See **bit_rate_value** in sequence header
vbv_buffer_size_extension	-	-	See **vbv_buffer_size_value** in sequence header
low_delay	StudioVideoObjectLayer()	**low_delay**	
frame_rate_extension_n	-	-	
frame_rate_extension_d	-	-	

Q.4 Sequence display extension

Sequence display extension of MPEG-2 is included in the MPEG-4 Studio Profile.

© ISO/IEC 2004 – All rights reserved

Q.5 Group of pictures header

MPEG-2 4:2:2 Profile syntax	*MPEG-4 Studio Profile*		*Note*
	layer	*syntax*	
time_code	Group_of_StudioVideoObjectPlane()	**time_code_smpte12m**	Studio Profile uses SMPTE 12M time code MPEG-2:25bits MPEG-4:64bits
closed_gop	Group_of_StudioVideoObjectPlane()	**closed_gov**	
broken_link	Group_of_StudioVideoObjectPlane()	**broken_link**	

Q.6 Picture header

MPEG-2 4:2:2 Profile syntax	*MPEG-4 Studio Profile*		*Note*
	layer	*syntax*	
temporal_reference	StudioVideoObjectPlane()	**temporal_reference**	
picture_coding_type	StudioVideoObjectPlane()	**vop_coding_type**	MPEG-2:3bits MPEG-4:2bits
vbv_delay	-	-	not affect decoding process
full_pel_forward_vector	-	-	used in ISO/IEC 11172-2 (MPEG-1)
forward_f_code	-	-	used in ISO/IEC 11172-2 (MPEG-1)
extra_bit_picture	-	-	not affect decoding process

© ISO/IEC 2004 – All rights reserved

Q.7 Picture coding extension

MPEG-2 4:2:2 Profile syntax	*MPEG-4 Studio Profile*		*Note*
	layer	*syntax*	
f_code[0][0]	StudioVideoObjectPlane()	vop_fcode[0][0]	
f_code[0][1]	StudioVideoObjectPlane()	**vop_fcode[0][1]**	
f_code[1][0]	StudioVideoObjectPlane()	**vop_fcode[1][0]**	
f_code[1][1]	StudioVideoObjectPlane()	**vop_fcode[1][1]**	
intra_dc_precision	StudioVideoObjectPlane()	**intra_dc_precision**	
picture_structure	StudioVideoObjectPlane()	**vop_structure**	
top_field_first	StudioVideoObjectPlane()	**top_field_first**	
frame_pred_frame_dct	StudioVideoObjectPlane()	**frame_pred_frame_dct**	
concealment_motion_vectors	-	-	
q_scale_type	StudioVideoObjectPlane()	**q_scale_type**	
intra_vlc_format	-	-	not affect decoding process
alternate_scan	StudioVideoObjectPlane()	**alternate_scan**	
repeat_first_field	StudioVideoObjectPlane()	**repeat_first_field**	
chroma_420_type	-	-	not affect decoding process and exists for historical reasons
progressive_frame	StudioVideoObjectPlane()	**progressive_frame**	
composite_display_flag	-	-	not affect decoding process

© ISO/IEC 2004 – All rights reserved

Q.8 Quant matrix extension, Picture display extension, Copy right extension

Quant matrix extension, Picture display extension and Copy right extension of MPEG-2 are included in the MPEG-4 Studio Profile.

Q.9 Slice

MPEG-2 4:2:2 Profile syntax	*MPEG-4 Studio Profile*		*Note*
	layer	*syntax*	
slice_vertical_ position_extension	-	-	calculated by **macroblock_number**
quantiser_scale_ code	StudioSlice()	**quntiser_scale_ code**	
intra_slice_flag	StudioSlice()	**intra_slice_flag**	
intra_slice	StudioSlice()	**intra_slice**	
reserved_bits	StudioSlice()	**reserved_bits**	
extra_bit_slice	StudioSlice()	**extra_bit_slice**	
extra_information_ slice	StudioSlice()	**extra_information_ slice**	

Q.10 Macroblock

MPEG-2 4:2:2 Profile syntax	*MPEG-4 Studio Profile*		*Note*
	layer	*syntax*	
macroblock_ escape	-	-	related to **not_coded**
macroblock_ address_increment	-	-	related to **not_coded**
quantiser_scale_ code	StudioMacroblock()	**quantiser_scale_ code**	

© ISO/IEC 2004 – All rights reserved

Q.11 Macroblock modes

MPEG-2 4:2:2 Profile) syntax	*MPEG-4 Studio Profile*		*Note*
	layer	*syntax*	
macroblock_type	StudioMacroblock_ modes()	macroblock_type	
frame_motion_type	StudioMacroblock_ modes()	**frame_motion_type**	
field_motion_type	StudioMacroblock_ modes()	**field_motion_type**	
dct_type	StudioMacroblock_ modes()	**dct_type**	

Q.12 Motion vectors

MPEG-2 4:2:2 Profile syntax	*MPEG-4 Studio Profile*		*Note*
	layer	*syntax*	
motion_vertical_ field_select[0][s]	motion_vectors()	motion_vertical_field_ select[0][s]	
motion_vertical_ field_select[1][s]	motion_vectors()	**motion_vertical_ field_select[1][s]**	

Q.13 Motion vector

MPEG-2 4:2:2 Profile syntax	*MPEG-4 Studio Profile*		*Note*
	layer	*syntax*	
motion_code[r][s][0]	motion_vector()	motion_code[r][s][0]	
motion_ residual[r][s][0]	motion_vector()	**motion_ residual[r][s][0]**	
dmvvector[0]	-	-	
motion_ code[r][s][1]	motion_vector()	**motion_ code[r][s][1]**	
motion_ residual[r][s][1]	motion_vector()	**motion_ residual[r][s][1]**	
dmvvector[1]	-	-	

© ISO/IEC 2004 – All rights reserved

Q.14 Coded block pattern

MPEG-2 4:2:2 Profile syntax	*MPEG-4 Studio Profile*		*Note*
	layer	*syntax*	
coded_block_ pattern_420	coded_block_ pattern()	**coded_block_ pattern_420**	Simple Studio Profile does not use 'cbp'
coded_block_ pattern_1	coded_block_ pattern()	**coded_block_ pattern_1**	Simple Studio Profile does not use 'cbp'

Q.15 Block

MPEG-2 4:2:2 Profile syntax	*MPEG-4 Studio Profile*		*Note*
	layer	*syntax*	
dct_dc_size_ luminance	StudioBlock()	**dct_dc_size_ luminance**	MPEG-2:2-9bits MPEG-4:2-14bits
dct_dc_differential	StudioBlock()	**dct_dc_differential**	MPEG-2:1-11bits MPEG-4:1-15bits
dct_dc_size_ chrominance	StudioBlock()	**dct_dc_size_ chrominance**	MPEG-2:2-10bits MPEG-4:2-15bits
Subsequent DCT coefficients	StudioBlock()	Subsequent DCT coefficient	
End of block	StudioBlock()	End of block	MPEG-2:2 or 4bits MPEG-4:3-15 bits

© ISO/IEC 2004 – All rights reserved

Annex R
(informative)

Forming (RUN, EOP) Symbols

After taking residues between the original VOP and the reconstructed VOP in the base layer, the DCT coefficients in each 8x8 block are obtained (in the case of FGST, the residues are the original VOP). (RUN, EOP) symbols are formed to represent the absolute values of the DCT coefficients. The 64 absolute values of the 8x8 DCT block are zigzag ordered into a one-dimensional array. The formation of (RUN, EOP) symbols is described as the following procedure:

- Find the maximum value of all the absolute values of the DCT coefficients in an fgs vop, and find the minimum number of bits, N, needed to represent this maximum value in the binary format. N is the number of vop-bps for this fgs vop.
- Within each 8x8 block, represent every one of the 64 absolute values of the DCT coefficients with N bits in the binary format and form N block-bps. A block-bp is defined as an array of 64 bits, taken one from each absolute value of the DCT coefficients at the same significant position (after bit shifting if frequency weighting is used).
- Starting from the most significant block-bp (MSB block-bp), 2-D symbols are formed of two components: (a) number of consecutive 0's before a 1 (RUN), (b) whether there are any 1's left on this bit block-bp, i.e. End-Of-Plane (EOP). If a block-bp after the MSB block-bp contains all 0's, a special symbol ALL-ZERO is formed to represent an all-zero block-bp.

The following example illustrates the procedure. Assume that the absolute values after zigzag ordering are given as follows and N = 6:

10, 0, 6, 0, 0, 3, 0, 2, 2, 0, 0, 2, 0, 0, 1, 0, ... 0, 0

The maximum value in this block is found to be 10 and the minimum number of bits to represent 10 in the binary format (1010) is 4. Therefore, two all-zero block-bps before the MSB block-bp are indicated by the fgs_cbp or fgs_msb_not_reached syntax. The remaining 4 block-bps are considered in forming the (RUN, EOP) symbols. Writing every value in the binary format using 4 bits, the 4 block-bps are formed as follows:

1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0 (MSB block-bp)

0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0 (Second MSB block-bp)

1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, ... 0, 0 (Third MSB block-bp)

0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, ... 0, 0 (Fourth MSB block-bp)

Converting the bits of the four block-bps into (RUN, EOP) symbols, we have

(0, 1)	(MSB block-bp)
(2, 1)	(Second MSB block-bp)
(0, 0), (1,0), (2,0), (1,0), (0, 0), (2, 1)	(Third MSB block-bp)
(5, 0), (8, 1)	(Fourth MSB block-bp)

Therefore, 10 (RUN, EOP) symbols are formed in this example.

© ISO/IEC 2004 – All rights reserved

Annex S
(informative)

A method of decoding a truncated bitstream

In a typical application of FGS, the bitstream at the input of an FGS decoder is a truncated version of the bitstream at the output of an FGS encoder. It is likely that, at the end of each fgs vop bitstream before the next fgs_vop_start_code, only partial bits of the fgs vop are at the input of the decoder due to truncation of the fgs vop bitstream. Decoding of the truncated bitstream is not normative. One possible method for decoding a truncated bitstream of an fgs vop is to look ahead 32 bits at every byte-aligned position in the bitstream indicated by a pointer. If the 32 bits form the fgs_vop_start_code, the decoder either completes decoding up to the fgs_vop_start_code or discards the bits before the fgs_vop_start_code. Otherwise, the first 8 bits of the 32 bits are information bits to be decoded for the fgs vop. The decoder moves forward the pointer by one byte and looks ahead another 32 bits to check for fgs_vop_start_code.

Because not all enhancement layer bits are available for reconstruction of the enhancement DCT coefficients, the decoder may have no enhancement DCT coefficients or partial enhancement DCT coefficients. In such a case, the optimal reconstruction of the DCT coefficients is performed according to the following procedure. Let DIFF be the partial absolute value obtained for a DCT coefficient, SIGN be the sign of the DCT coefficient, and ENH_COEFF be the reconstructed enhancement DCT coefficient.

```
if (DIFF != 0) {
        DIFF += offset
        ENH_COEFF = SIGN*DIFF
}
```

where offset = (decoded_level>2) ? (1<<(decoded_level-3)) : 0 and decoded_level is the index of the last block-bp decoded from the bitstream for an 8x8 DCT block. The decoded_level is '1' for the least significant block-bp and increments its value toward the most significant block-bp in the 8x8 DCT block.

© ISO/IEC 2004 – All rights reserved

ICS 35.040
Price based on 706 pages

© ISO/IEC 2004 – All rights reserved