RealNetworks

RealVideo 8

External Specification

Video and Audio Technologies
Codec Group
RealNetworks, Inc

September 19, 2002

Version 2.0

Summary

This document is the draft specification of RealVideo8 Codec. RealVideo8 achieves new levels of compression performance at low as well as high data rates. The improvements are due in part to 1/3 pixel interpolation for motion estimation, 4x4 pixel block transforms, additional 4x4 intra prediction modes, the addition of 16x16 intra prediction, and more efficient variable length coding by symbol manipulation. The algorithm also benefits from noise reduction pre-filtering, an in-loop deblocking filter, and a matched post-filter.

Revision History:

Revision	Date	Comment
1.0	03/07/02	Decoder Specification, from Original Encoder + Decoder Spec "Tromso External Specification v10.doc"
2.0	09/19/02	RealVideo 8 Specification new improved

Contributors:

Karl Lillevold
Greg Conklin
Yuriy Reznik
Neelesh Gokhale
Mitch Bodart
Gim Deisher
Tom Gardos
Chris Lord
Bob Reese
Lily Yang

Table of Content

1 High Level Overview

RealVideo8 represents major advances in compression performance. RealVideo8 achieves new levels of compression performance at low as well as high data rates. The improvements are due in part to 1/3 pixel interpolation for motion estimation, 4x4 pixel block transforms, additional 4x4 intra prediction modes, the addition of 16x16 intra prediction, and more efficient variable length coding by symbol manipulation. The algorithm also benefits from noise reduction pre-filtering, an in-loop deblocking filter, and a matched post-filter. RealVideo8 Decoder has built in CPU scalability to ensure best possible Video Experience various hardware configurations.

2 Requirements, Objectives

Minimum Decode Platform: 160x120 pixel, 7.5 fps decode on a Pentium™ 166 MHz with 16 MB of memory.

Target bit rates: < 20 kbps, 30 kbps, 100 kbps, 500 kbps, 1 Mbps

Target frame sizes: 32x32 to 2048x1152, with particular attention to the range SQCIF (128x96) to Half-Horizontal CCIR-601 Resolution (360 x 480).

Video quality requirements: A noticeable improvement in video quality over RealVideo G2 at comparable data rates.

3 Interface Specification

3.1 RealVideo DLL

Decoder DLL will comply to the RealVideo back-end interface detailed in Annex A. These interfaces might change for subsequent releases.

3.2 Console Application

A console application version of the codec will be available for development and testing purposes. The encoder and decoder command line arguments are listed in Annex B.

4 Algorithm Descriptions

As compression quality is still considered the most important development area for improving the streaming video experience, RealVideo8 delivers a quantum jump in compression efficiency.

4.1 Introduction

The RealVideo8 algorithm is largely based on H.26L[3][4] or now called the Joint Video Team proposal Mpeg4 part 10 / Advanced Video Codec, which experiments have shown provides significant and very visible coding gains over H.263+. RealVideo8 deviates by:

- not performing the chroma DC coefficient manipulation
- not including the 16x8, 8x16, 8x4, 4x8 and 4x4 motion compensated modes
- additional 4x4 intra prediction modes (These were proposed to JVT and have been accepted in simplified form (mode 7))
- addition of B frames (26L now has generalised Bipredictive-frames)
- inloop filter definition and usage
- addition of an alternate VLC for coefficients

4.2 Overview

RealVideo 8 is a hybrid predictive coder that uses temporal prediction (motion compensation) and spatial prediction (intra-prediction), transform-based residual coding and an inloop deblocking filter. Figure 4.1 provides a high-level block diagram of the algorithm.

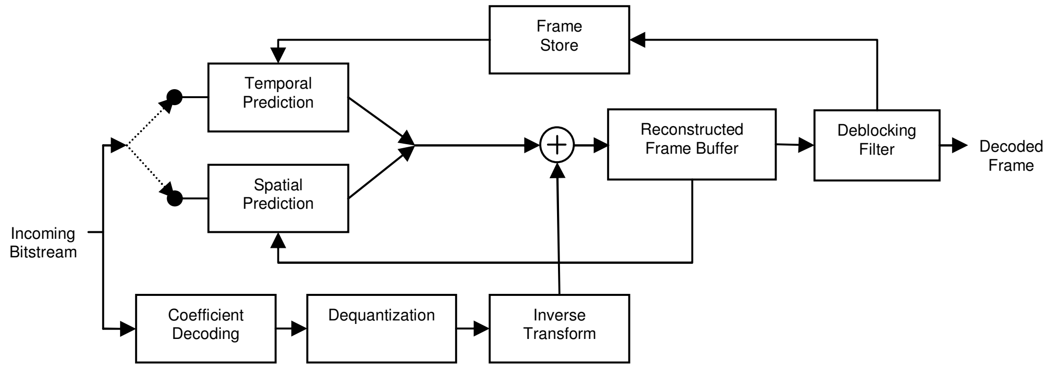


Figure 4.1: Block diagram of the RealVideo 8 decoder algorithm

The Incoming Bitstream describes how to reconstruct pictures in groups of non-overlapping 16x16 pixels (macroblocks). For each macroblock, the bitstream indicates whether Spatial Prediction or Temporal Prediction is to be used. Once a prediction is formed, the image residual is formed through the Coefficient Decoding, Dequantization and Inverse Transform process. The prediction and residual are added and stored in memory for use in future spatial prediction. Once the entire picture has been reconstructed, an inloop deblocking filter is used to remove blocking artifacts. This filtered image is then ready to be rendered (a post-filter is applied) and, in addition, used for future temporal prediction.

The RealVideo 8 decoding algorithm is defined to reconstruct video images in YUV 4:2:0 format. It is the function of the video renderer (or equivalent player module) to format the picture to the appropriate color space for display.

4.2.1 Picture Types

There are 3 picture types in RealVideo 8 -- I-Pictures, P-Pictures and B-Pictures.

I-Pictures are also referred to as Intra-Frames or Key Frames. They do not use temporal prediction and, therefore, do not require other decoded reference frames to be in the decoder for proper reconstruction. I-Pictures provide entry or access points to the video sequence.

P-Pictures use both spatial and temporal prediction. The temporal prediction always uses one reference frame. That reference frame shall always be the most previous reconstructed I-Picture or P-Picture.

B-Pictures use both spatial and temporal prediction. However, temporal prediction uses up to 2 reference frames. These reference frames shall always be the 2 most previous reconstructed I-Pictures or P-Pictures that were found in the bitstream (i.e. in "bitstream" order, not display order). Because the display time of one reference picture is always before the B-Picture and the other is always after the B-Picture, the placement of B-Pictures in the bitstream is not in display order. Figure 4.2 provides an example of display and bitstream ordering of I, P and B Pictures.

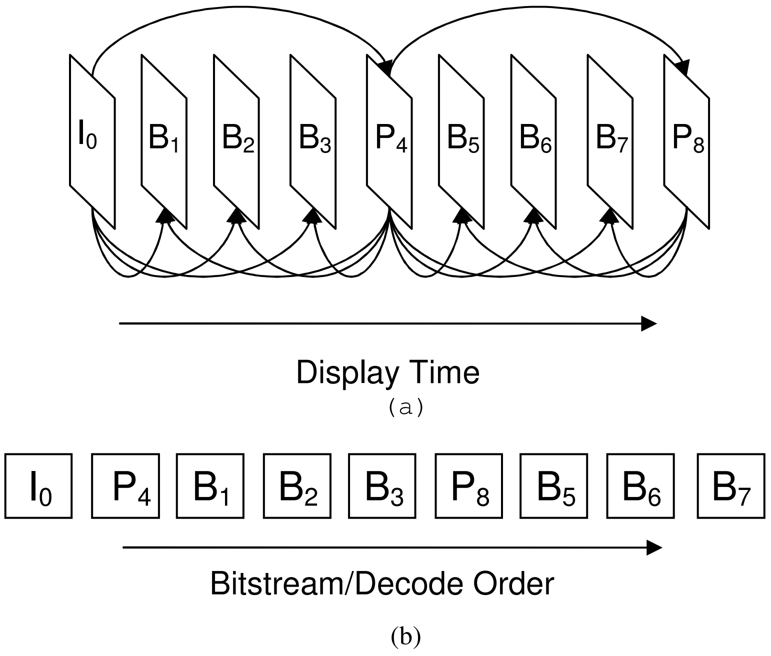


Figure 4.2: (a) Display Order. (b) Bitstream and Decode Order

4.2.2 Picture Structure

Pictures are divided into non-overlapping 16x16 group of pixels called macroblocks. For instance, a QCIF picture (176x144 pixels) is divided into 99 macroblocks as indicated in Figure 4.3.

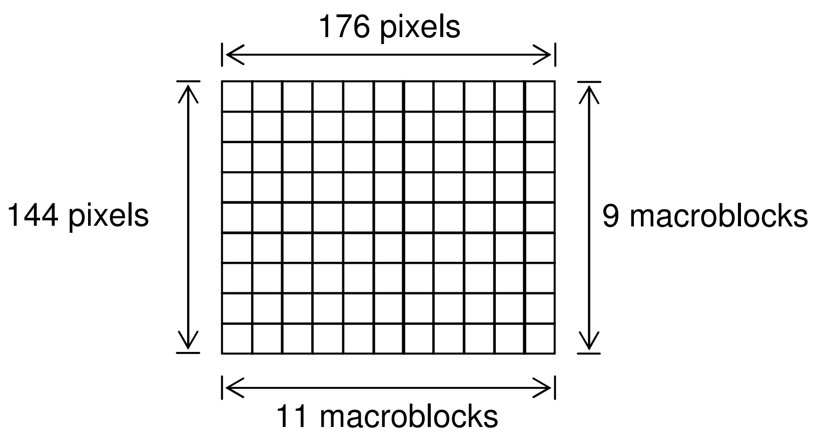


Figure 4.3: A picture with 11 x 9 macroblocks (QCIF picture)

When parsing and decoding the video bitstream macroblocks are scanned from left to right starting at the top left of the picture. Once an entire row of macroblocks are decoded the next row down proceeds.

4.2.3 Macroblock Structure

The basic transform used for residual coding is a 4x4 2-D transform. Figure 4.4 below indicate how a macroblock is divided into 4x4 regions and the scanning order of these regions.

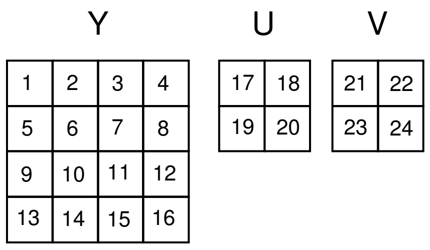


Figure 4.4: Macroblock scanning order of 4x4 blocks during residual coding

4.3 Core Compression Algorithm

4.3.1 Macroblock Types

Each macroblock is given a categorization (macroblock type) that indicates both the way prediction is done for that macroblock (e.g. spatial or temporal) and the way residual transform is done (e.g. single 4x4 transforms or a double transforms). The complete list of macroblock types is given below in Table 4.1.

TABLE 4.1: List of macroblock types

MB Types	**Description**	**I-Pic**	**P-Pic**	**B-Pic**
INTRA	Intra, 16 4x4 predictions	X	X	X
INTRA_16x16	Intra, 16x16 prediction, Dbl Xfm	X	X	X
INTER	Inter, 1MV		X	
INTER_4V	Inter, 4MVs for 4 8x8 blocks		X	
SKIPPED	Inter, no residual, MV=(0,0)		X	
FORWARD	Fwd MV, 1MV			X
BACKWARD	Bwd MV, 1MV			X
DIRECT	Direct, Derived 2MV for 16x16 block			X
SKIPPED	Direct, no residual, Derived MV for 16x16 block			X

4.3.2 1/3 sub-pel prediction

Motion vectors in RealVideo 8 are transmitted in 1/3 pixel units. When the motion vectors for a macroblock have been decoded the full-pixel offset can be obtained by dividing by 3.

```
      MVx_int = (MVx_luma / 3)
      MVy_int = (MVy_luma / 3).
```

The “phase” or sub-pixel location can be obtained as follows.

```
MVx_sub = (MVx_luma - MVx_int*3)
MVy_sub = (MVy_luma - MVy_int*3).
```

For luma sub-pixel interpolation is calculated with a 4-tap filter. For chroma, a 2-tap filter is used. In addition, one of the 9 interpolated pixels, MVx_sub = 2, MVy_sub = 2, in the luma plane is created using a stronger filter. Thus, using 1/3-pel prediction instead of ½-pel prediction has two advantages

- more accurate motion estimation
- automatic adaptation of, and a larger variation in filter strength

The different horizontal and vertical filters are illustrated in Table 4.2.

TABLE 4.2: Luma Horizontal and vertical motion compensation filters

(MVx_sub, MVy_sub)	Horizontal, Vertical Filter $p_{i,j}$ = inter pixels, $t_{i,j}$ = temporary buffer, $y_{i,j}$ = interpolated image Note: (i,j) = coordinates x,y pair and not row,column pair
(0,0)	$t_{i,j} = p_{i,j}$ $y_{i,j} = t_{i,j}$
(0,1)	$t_{i,j} = p_{i,j}$ $y_{i,j} = (-1t_{i,j-1} + 12t_{i,j} + 6t_{i,j+1} - 1t_{i,j+2} + 8) >> 4$
(0,2)	$t_{i,j} = p_{i,j}$ $y_{i,j} = (-1t_{i,j-1} + 6t_{i,j} + 12t_{i,j+1} - 1t_{i,j+2} + 8) >> 4$
(1,0)	$t_{i,j} = (-1p_{i-1,j} + 12p_{i,j} + 6p_{i+1,j} - 1p_{i+2,j} + 8) >> 4$ $y_{i,j} = t_{i,j}$
(1,1)	$t_{i,j} = (-1p_{i-1,j} + 12p_{i,j} + 6p_{i+1,j} - 1p_{i+2,j})$ $y_{i,j} = (-1t_{i,j-1} + 12t_{i,j} + 6t_{i,j+1} - 1t_{i,j+2} + 128) >> 8$
(1,2)	$t_{i,j} = (-1p_{i-1,j} + 12p_{i,j} + 6p_{i+1,j} - 1p_{i+2,j})$ $y_{i,j} = (-1t_{i,j-1} + 6t_{i,j} + 12t_{i,j+1} - 1t_{i,j+2} + 128) >> 8$
(2,0)	$t_{i,j} = (-1p_{i-1,j} + 6p_{i,j} + 12p_{i+1,j} - 1p_{i+2,j} + 8) >> 4$ $y_{i,j} = t_{i,j}$

(2,1)	$t_{i,j} = (-1p_{i-1,j} + 6p_{i,j} + 12p_{i+1,j} - 1p_{i+2,j})$ $y_{i,j} = (-1t_{i,j-1} + 12t_{i,j} + 6t_{i,j+1} - 1t_{i,j+2} + 128) >> 8$
(2,2)	$t_{i,j} = (-0p_{i-1,j} + 6p_{i,j} + 9p_{i+1,j} +1p_{i+2,j})$ $y_{i,j} = (-0t_{i,j-1} + 6t_{i,j} + 9t_{i,j+1} +1t_{i,j+2} + 128) >> 8$

The final value of y clipped to 0-255.

Motion vectors for chroma motion compensation are derived from the motion vectors for the luma. Specifically, the chroma MVs are calculated as

```
      MVx_chroma = MVx_luma >> 1
      MVy_chroma = MVy_luma >> 1
```

Then the integer offset and sub-pixel location can be obtained by

```
      MVx_chroma_int = (MVx_chroma / 3)
      MVy_chroma_int = (MVy_chroma / 3).

      MVx_chroma_sub = (MVx_chroma - MVx_chroma_int*3)
      MVy_chroma_sub = (MVy_chroma - MVy_chroma_int*3).
```

Additionally, the size of motion compensation blocks are half the size, horizontally and vertically, from those used in luma. Thus, motion compensation block sizes for chroma include 8x8, 8x4, 4x8 and 4x4. Chroma motion compensation filters are given in Table 4.3.

Note when these filters are used for chroma, the drift due to rounding errors is much smaller than in H.263, and the rounding control mechanism (RCONTROL) that was added to H.263+ is not needed.

TABLE 4.3: Chroma Horizontal and vertical motion compensation filters

(MVx_chroma_sub, MVy_chroma_sub)	Filter (input $p_{y,x}$, output $f_{y,x}$)
(0,0)	$f_{i,j} = p_{i,j}$
(0,1)	$f_{i,j} = (3p_{i,j} + 5p_{i,j+1} + 4) >> 3$
(0,2)	$f_{i,j} = (5p_{i,j} + 3p_{i,j+1} + 4) >> 3$
(1,0)	$f_{i,j} = (5p_{i,j} + 3p_{i+1,j} + 4) >> 3$
(1,1)	$f_{i,j} = (25p_{i,j} + 15p_{i+1,j} + 15p_{i,j+1} + 9p_{i+1,j+1} + 32) >> 6$
(1,2)	$f_{i,j} = (15p_{i,j} + 9p_{i+1,j} + 25p_{i,j+1} + 15p_{i+1,j+1} + 32) >> 6$
(2,0)	$f_{i,j} = (3p_{i,j} + 5p_{i+1,j} + 4) >> 3$
(2,1)	$f_{i,j} = (15p_{i,j} + 25p_{i+1,j} + 9_{i,j+1} + 15p_{i+1,j+1} + 32) >> 6$
(2,2)	$f_{i,j} = (9p_{i,j} + 15p_{i+1,j} + 15p_{i,j+1} + 25p_{i+1,j+1} + 32) >> 6$

The final value of f clipped to 0-255.

4.3.3 One VLC code

Previous standards-based codecs used different variable-length (VLC) and fixed-length (FLC) code tables for different coding parameters where the VLCs and FLCs were matched to the statistics of the parameters to be coded. This algorithm uses one variable-length code table to code all the information in the video sequence. To achieve a high coding efficiency, the coding model has been designed so that the symbols match the statistics of this single VLC. The particular VLC that is used has a regular structure:

VLC in compressed	**Code number (N)**	**Explicit**	**Inf**	**L**
1	0	1		1
0 x_0 1	1	0 0 1	0	3
	2	0 1 1	1	3
0 x_1 0 x_0 1	3	0 0 0 0 1	00	5
	4	0 0 0 1 1	01	5
	5	0 1 0 0 1	10	5
	6	0 1 0 1 1	11	5
0 x_2 0 x_1 0 x_0 1 …	7	0 0 0 0 0 0 1	000	7
	8	0 0 0 0 0 1 1	001	7
	…	…	…	…

Table 1 The one and only VLC

The table of codewords may be written in the following compressed form.

$$\begin{array}{c} 1 \\ 0\ x_0\ 1 \\ 0\ x_1\ 0\ x_0\ 1 \\ 0\ x_2\ 0\ x_1\ 0\ x_0\ 1 \\ 0\ x_3\ 0\ x_2\ 0\ x_1\ 0\ x_0\ 1 \\ \cdots\cdots\cdots\cdots \end{array}$$

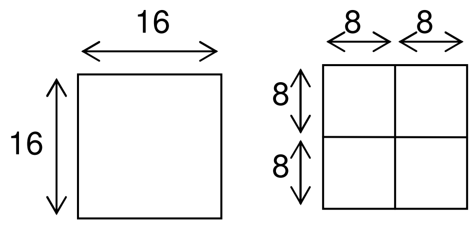
where x_n take values 0 or 1. We will sometimes refer to a codeword with its length in bits (L) and INFO = x_n .. x_1 x_0 . Notice that the number of bits in INFO is L/2 (division by truncation).

When L and INFO is known, the regular structure of the table makes it easy to create a codeword bit by bit. Similarly, a decoder may easily read bit by bit until the last "1" which gives the end of the codeword. L and INFO is then readily available. For each parameter to be coded, there is a conversion rule from the parameter value to the code number (or L and INFO). Table 3 lists the connection between code number and most of the parameters used in the present coding method.

RealVideo8 uses alternative VLC coding for coefficient information. See Block Layer Description.

4.3.4 Block sizes for Inter prediction

In this model it is possible to estimate motion and compensate motion on 16x16, and 8x8 pixel block sizes. The encoder chooses one motion compensation mode for each macroblock. Motion vectors off the edge of the frame are allowed and used. A valid MV is defined such that the interpolation of that MV is possible within the padded image.



INTER INTER_4V
INTER_16X16
FORWARD
BACKWARD
DIRECT

Figure 4.5: Motion compensation block sizes for Inter macroblocks

4.3.5 Improved intra coding

An improved advanced intra coding mode is used. Relative to the AIC mode in H.263+, this version is 4x4 based, the prediction is done in the spatial domain using one of nine prediction modes. There are six additional prediction modes relative to the AIC mode in H.263+. These additional prediction modes are diagonal. The other modes are (1) the average of the block above and to the left, (2) column based from above, and (3) row based from the left. See below for a detailed description. In the reference model, the choice of coding mode is based on SAD.

A 4x4 block is to be coded (pixels labeled a to p below). The pixels A to P and X from neighboring blocks are already decoded and may be used for prediction. "//" means division with rounding.

```
X A B C D E F G H
I a b c d
J e f g h
K i j k l
L m n o p
M
N
O
P
```

Under some situations pixels A,B,C,D or I,J,K,L or X are not available for use at the decoder. These situations include

1. These pixels are located outside the picture boundary
2. These pixels belong to another independent slice

In these cases, modes that require these pixels will not be encountered by the decoder. Similarly, pixels E,F,G,H or M,N,O,P may also not be available for use at the decoder. These situations include the ones above with the additional case

3. These pixels are located in parts of the current frame that have yet to be decoded and reconstructed

In these situations the decoder shall use the value of D for pixels E,F,G,H when they are not available. T he decoder shall use the value of L for pixels M,N,O,P when they are not available. For example, E,F,G,H are not valid for 4x4 blocks on the right edge of the 16x16 macroblock except the top row when the macroblock is not at the right edge of the picture. M,N,O,P are only valid for 4x4 blocks on the left edge of the 16x16 macroblock except on the bottom row.

Mode 0:
Generally all pixels are predicted by (A+B+C+D+I+J+K+L)//8. If four of the pixels are outside the picture, the average of the remaining four is used for prediction. If all 8 pixels are outside the picture the prediction for all pixels in the block is set to 128. A block may therefore always be predicted in this mode.
Mode 1:

If pixels A,B,C,D are inside the picture, a,e,i,m are predicted by A, b,f,j,n by B etc.

Mode 2:

If pixels I,J,K,L are inside the picture, a,b,c,d are predicted by I, e,f,g,h by J etc.

Mode 3:

This mode is used only if all A,B,C,D,I,J,K,L,X are inside the picture. This is a diagonal prediction.

m is predicted by	(L+2K+J)//4
i,n are predicted by	(K+2J+I)//4
e,j,o are predicted by	(J+2I+X)//4
a,f,k,l are predicted by	(I+2X+A)//4
b,g,l are predicted by	(X+2A+B)//4
c,h are predicted by	(A+2B+C)//4
d is predicted by	(B+2C+D)//4

Mode 4 - 8:

These diagonal modes are used only if all A,B,C,D,I,J,K,L,X are inside the picture. When E,F,G,H are not valid predictors D is used instead. When M,N,O,P are not valid predictors L is used instead. E,F,G,H are not valid for 4x4 blocks on the right edge of the 16x16 macroblock except the top row right edge block when the macroblock is not at the right edge of the picture. M,N,O,P are only valid for 4x4 blocks on the left edge of the 16x16 macroblock which are not on the bottom row.

Mode 4:

a is predicted by	(A+2B+C+I+2J+K)//8
b,e are predicted by	(B+2C+D+J+2K+L)//8
c,f,i are predicted by	(C+2D+E+K+2L+M)//8
d,g,j,m are predicted by	(D+2E+F+L+2M+N)//8
h,k,n are predicted by	(E+2F+G+M+2N+O)//8
l,o are predicted by	(F+2G+H+N+2O+P)//8
p is predicted by	(G+H+O+P)//4

Mode 5:

a,j are predicted by	(X+A)//2
b,k are predicted by	(A+B)//2
c,l are predicted by	(B+C)//2
d is predicted by	(C+D)//2
e,n are predicted by	(I+2X+A)//4
f,o are predicted by	(X+2A+B)//4
g,p are predicted by	(A+2B+C)//4
h is predicted by	(B+2C+D)//4
i is predicted by	(X+2I+J)//4
m is predicted by	(I+2J+K)//4

Mode 6:

a is predicted by	(2A+2B+J+2K+L)//8
b,i are predicted by	(B+C)//2
c,j are predicted by	(C+D)//2
d,k are predicted by	(D+E)//2
l is predicted by	(E+F)//2
e is predicted by	(A+2B+C+K+2L+M)//8
f,m are predicted by	(B+2C+D)//4
g,n are predicted by	(C+2D+E)//4
h,o are predicted by	(D+2E+F)//4
p is predicted by	(E+2F+G)//4

Mode 7:

a is predicted by	(B+2C+D+2I+2J)//8
b is predicted by	(C+2D+E+I+2J+K)//8
c,e are predicted by	(D+2E+F+2J+2K)//8
d,f are predicted by	(E+2F+G+J+2K+L)//8
g,i are predicted by	(F+2G+H+2K+2L)//8
h,j are predicted by	(G+3H+K+3L)//8
l,n are predicted by	(L+2M+N)//4
m,k are predicted by	(G+H+L+M)//4
o is predicted by	(M+N)//2
p is predicted by	(M+2N+O)//4

Mode 8:

a,g are predicted by	(X+I)//2
b,h are predicted by	(I+2X+A)//4
c is predicted by	(X+2A+B)//4
d is predicted by	(A+2B+C)//4
e,k are predicted by	(I+J)//2
f,l are predicted by	(X+2I+J)//4
i,o are predicted by	(J+K)//2
j,p are predicted by	(I+2J+K)//4
m is predicted by	(K+L)//2
n is predicted by	(J+2K+L)//4

For chroma blocks the mode used is the mode of the corresponding upper left luma block.

4.3.6 16x16 Intra Coding

For Intra16x16 macroblocks, one of four prediction modes are used to form a 16x16 prediction for the entire macroblock. Three modes are similar to modes 0 – 2 for 4x4 intra plus a new planar prediction mode. The image residual of Intra16x16 macroblocks are Double Transformed (see section QQ).

Define P(i,-1), i=0..15 to be the 16 pixels above the macroblock to be predicted, and P(-1,j), j=0..15 to be the 16 pixels to the left of the macroblock to be predicted.

Mode 0: DC Prediction

If all P(i,-1) and P(-1,i) are inside the picture and current slice then all 256 pixels are predicted by

$$\text{pred} = \left(\left(\sum_{i=0}^{15} P(i,-1) + P(-1,i)\right) + 16\right) >> 5$$

If P(i,-1) are inside the picture and current slice then all 256 pixels are predicted by

$$\text{pred} = \left(\left(\sum_{i=0}^{15} P(i,-1)\right) + 8\right) >> 4$$

If P(-1,i) are inside the picture and current slice then all 256 pixels are predicted by

$$\text{pred} = \left(\left(\sum_{i=0}^{15} P(-1,i)\right) + 8\right) >> 4$$

If all 32 pixels are outside the picture, the prediction for all pixels in the block is set to 128. A block may therefore always be predicted in this mode.

Mode 1: Vertical Prediction
If pixels P(i,-1), i=0..15 are inside the picture and current slice, P(0,j), j=0..15 are predicted by P(0,-1) etc.

Mode 2: Horizontal Prediction
If pixels P(-1,j), j=0..15 are inside the picture and current slice, P(i,0), i=0..15 are predicted by P(-1,0) etc.

Mode 3: Planar Prediction

This mode is used only if all P(i,-1), i=0..15 and P(-1,j), j=0..15 are inside the picture and current slice. The following calculations are performed:

$$H = \overset{8}{\underset{i=1}{\bullet}} i \cdot (P(7+i,-1) - P(7-i,-1))$$

$$V = \overset{8}{\underset{j=1}{\bullet}} j \cdot (P(-1,7+j) - P(-1,7-j))$$

$$a = 16x(P(-1,15) + P(15,-1))$$
$$b = (H+(H>>2))>>4$$
$$c = (V+(V>>2))>>4$$

And finally the actual prediction:

$$\text{pred}(i,j) = (a + b\cdot(i-7) + c\cdot(j-7) + 16) >> 5$$

All calculations shall be integer. No divisions (only shifts) are needed, and all calculations shall be within 16 bits.

For chroma the mode used is the mode chosen for luma, except when the luma mode is 3 then mode 0 is used. Modes 0,1, and 2 are predicted in the same way as luma except 8x8 blocks are used.

4.3.7 4x4 based transform

For the transform to the frequency domain of the residual after INTRA coding or motion prediction, a 4x4 pixel block transform is used instead of the H.263+ 8x8 pixel block size. The transform is defined in section 4.3.7.1.

4.3.7.1 Exact integer transform instead of DCT

A 4x4 integer transform is used for image residuals. By having an exact definition of the inverse transform, there is no encoder/decoder mismatch. The transformation of the pixels a,b,c,d into four transform coefficients in one dimension is defined by:

$$A = 13a + 13b + 13c + 13d$$
$$B = 17a + 7b - 7c - 17d$$
$$C = 13a - 13b - 13c + 13d$$
$$D = 7a - 17b + 17c - 7d$$

The inverse transform is defined by:

$$a' = 13A + 17B + 13C + 7D$$
$$b' = 13A + 7B - 13C - 17D$$
$$c' = 13A - 7B - 13C + 17D$$
$$d' = 13A - 17B + 13C - 7D$$

The relationship between the transform in one dimension without normalization is $a' = 676 \times a$. This is used in the quantization step (see below). The actual transform is 2D and since it is a separable transform, it implemented as a horizontal 1D transform followed by a vertical 1D transform.

4.3.7.2 Double Transform

An additional 4x4 transform is used for the 16 DC coefficients of the 16 4x4 transforms inside a macroblock. The coefficients of this second transform are coded and transmitted as a block in addition to the 16 4x4 luma blocks (each then having only 15 coefficients). Since we use the same integer transform to DC coefficients, we have to perform additional normalization to those coefficients, which implies a division by 676. To avoid the division we performed normalization by $49/2^{15}$ on the encoder side and $48/2^{15}$ on the decoder side, which gives sufficient accuracy.

4.3.8 Quantization

The quantization is table-based and designed in such a way that the bit usage as a function of the quantization parameter is fairly linear. In H.263+, this function is highly non-linear. In the encoder and decoder, the QP range 0-31 is mapped into the tables A[QP] and B[QP], respectively, where the relationship between A[] and B[] is:

$$A[QP] \times B[QP] \times 676^{2} = 2^{34}.$$

with

```
A(QP=0,..,31) = {620, 553, 492, 439, 391, 348, 310, 276, 246, 219, 195,
      174, 155, 138, 123, 110, 98, 87, 78, 69, 62, 55, 49, 44, 39, 35,
      31, 27, 24, 22, 19, 17}

B(QP=0,..,31) = {60, 67, 76, 85, 96, 108, 121, 136, 152, 171, 192, 216,
      242, 272, 305, 341, 383, 432, 481, 544, 606, 683, 767, 854, 963,
      1074, 1212, 1392, 1566, 1708, 1978, 2211}
```

Quantization of coefficient level K is performed as

```
LEVEL = ((K>>4) x A[QP]x32 + f )>>16/32
```

where f is 5 for Inter macroblocks and 10 for Intra macroblocks.
Dequantization is defined as

```
K' = ((LEVEL x B[QP]) + 8)>>4
```

For the coefficients of the second transform in INTRA_16x16 macroblocks quantization is performed as

```
LEVEL = (K x A[QP] + f)>>20
```

where f is 0x55555, while dequantization is as above.
The DC coefficient of this block is given a lower QP than the AC coefficients:

```
luma_quant_DC[32] =
      {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22
      ,22,22,23,23,23,24,24,25,25};
```

Quantization is performed the same way for chroma as for luma, except the the QP value used is derived from the QP used for luma using the tables below. The chroma DC coefficient is given an even lower QP than the chroma AC coefficients:

```
chroma_QP = chroma_QP_map[luma_QP];

chroma_QP_map_AC[32] =
      {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,17,18,
      19,20,20,21,22,22,23,23,24,24,25,25};

chroma_QP_map_DC[32] =
      {0,0,0,1,2,3,4,5,6,7, 8, 9,10,11,12,13,14,15,15,16,
      17,18,18,19,20,20,21,21,22,22,23,23};
```

After inverse transformation, the pixel values will then be 2^{10} too high, and a 10 bit downshift is needed as a part of the frame reconstruction. The definition of the transform and quantization is designed so that no overflow will occur with the use of 32-bit arithmetic.

4.3.8.1 *Dynamic Range for Various Methods.*

A*B*676*676 = 2^34
Transform Input = 9 Bits per pixel
Double Xfrm input is DC coeff of 16 4x4 blocks normalized by 49/2^15. So the input to the remaining chain is 11 bits.
Transform Intermediates = 13*13*4*4 * 2^9 = 21 Bits.
Transform Output = 21 Bits
(11 bits can represent normalized Xfrm at QP0)

Quant Input = 21Bits
Transform Coeff Value Reduced to 17 Bits and then saturated to 16 bits during Quantization. (Corresponds to ½ LSB Granularity for Table A)
The down shift & saturation is not done for the double Xfrm.
Quant Output = 10 Bits. (signed)
(In case of Double Xfrm and SuperVLC, if the output exceeds 10Bits, the Double transform is not done. The MB is recoded as INTRA MB)
Tranform + Quant normalization = 2^20

QVAL * B = **Level * [A * B * 676 * 676] / [13 * 13] * 2^20 <** 2^16
Dquant Input = 10 Bits
Dquant Intermediates = 16 Bits
Dquant Normalisation = 2^4
Dquant Output = Max 12 Bits

Ixfrm Input = 12 Bits
Ixfrm Intermediate = 13*13 * 2^12 < 2^20
Ixfrm Normalization = 2^10
Ixfrm Output = 9 bits

4.3.9 Deblocking / smoothing filters

For I, P and B frames (B frames optional in the encoder) an in-loop deblocking filter very similar to our previous deblocking filter is used. Chroma planes are deblock filtered for I frames only. For all frames a combined smoothing / deblocking standalone postfilter is used. Combined smoothing / deblocking means that all pixels along the 4x4 grid are filtered more strongly than other pixels.

4.3.9.1 In-loop deblocking

The in-loop deblocking filter is similar to the in-loop deblocking filter defined in annex J in H.263 version 2, but works on 4x4 edges instead of 8x8 edges, and only one instead of two pixels on each side of the edge is filtered.

The filter operations are performed across 4x4 block edges at the encoder as well as on the decoder side. The reconstructed image data (the sum of the prediction and the reconstructed prediction error) are clipped to the range 0 to 255. Then the filtering is applied, which alters the picture that is to be stored in the picture store for future prediction. The filtering operation includes an additional clipping to ensure the resulting pixel values stay in the range 0…255.

The deblocking filter operates using a set of four (clipped) pixel values on a horizontal or vertical line in the reconstructed picture, denoted as A, B, C and D, of which A and B belong to one block called block1 and C and D belong to a neighboring block called block2 which is to the right or below block1. All edges of intra blocks are always filtered while edges of non-intra blocks are filtered only if the block has at least one coded coefficient or if any motion vector component differs from its neighbor by more than 3.

The pixels denoted B and C are replaced by B1 and C1, which are computed as:

```
B1 = clip(B + d2)
C1 = clip(C - d2)
d2 = clipd1(d1, strength[QP])
d1 = ((A - B - C - D) << 2) >> 3
strength[QP] = {0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,2,2,2,2,2,3,3,3,3,4,4,4,4,5,5,5}
```

The above strength table is the default table; alternate strength tables are available to be used, however keeping the encoder and decoder in sync (both using the same table) must be handled outside of the bitstream as no means is provided for signaling the use of an alternate table. The alternate strength tables are:

```
{0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,2,2,2,2,2,2,3,3,3,3,3,3,4,4,4}
{0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,2,2,2,2,2,3,3,3,3,4,4,4,4,5,5,5}
{0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,2,2,2,2,2,3,3,3,4,4,4,5,5,5,6,6,6}
{0,0,0,0,0,0,0,0,0,0,0,1,1,1,2,2,2,3,3,3,4,4,4,5,5,5,6,6,6,7,7,7}
{0,0,0,0,0,0,0,0,0,0,0,1,1,2,2,3,3,4,4,4,5,5,5,6,6,6,7,7,7,8,8,8}
{0,0,0,0,0,0,0,0,0,0,0,1,2,3,4,4,4,5,5,5,6,6,6,7,7,7,8,8,8,9,9,9}
```

The function clip(x) clips x to the range 0…255. The function clipd1(x, lim) clips x to the range +/- abs(lim).

4.3.9.2 Combined smoothing/deblocking postfilter

The combined smoothing / deblocking postfilter filters all pixels, but to obtain the deblocking effect, the filter strength on pixels next to 8x8 and 4x4 block edges is stronger. Only the filtering in one dimension will be defined. To obtain a two-dimensional effect the filter is first applied in one direction (vertically), then the other direction (horizontally)[‡].

Given the pixels A, B, C, D, E, the pixel C is replaced by C1, which is computed as:

```
C1 = clip(C + d2)
d2 = clipd1(d1, strength[QP][edgeposition])
d1 = (A + 2B - 6C + 2D + E) >> 3
```

The edge_position table provides a pel position dependent index into the strength table, the index into the edge_position table being the position of the pel to be filtered with respect to an 8x8 block edge.

```
edge_position = {2,1,0,1,1,0,1,2}
```

There are four sets of strength tables, default is strength_select = 1.

```
strength[strength_select][QP][edge_position] =
{      // 0
{0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0},
{0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0},
{0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0},
{0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0},
},
{      // 1 (default)
{0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0},
{0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0},
```

[‡] Whether the postfilter is first applied vertically or horizontally, can be changed to accommodate optimization preferences.

```
{0,1,1}, {0,1,1}, {0,1,1}, {0,1,1}, {0,1,1}, {0,1,2}, {0,1,2}, {0,1,2},
//  16       17       18       19       20       21       22       23
{1,2,3}, {1,2,4}, {1,3,5}, {2,4,6}, {2,5,7}, {2,6,8}, {2,6,9}, {2,6,9},
//  24       25       26       27       28       29       30       31
},
{     // 2
{0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0},
{0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0},
{0,1,1}, {0,1,1}, {0,1,1}, {0,1,1}, {0,1,1}, {0,1,1}, {0,1,1}, {0,1,1},
//  16       17       18       19       20       21       22       23
{1,1,2}, {1,1,3}, {1,2,3}, {1,2,4}, {1,2,5}, {2,3,5}, {2,4,6}, {2,4,6},
//  24       25       26       27       28       29       30       31
},
{     // 3
{0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0},
{0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0},
{0,1,1}, {0,1,1}, {0,1,1}, {0,1,1}, {0,1,1}, {0,1,1}, {0,1,1}, {0,1,1},
//  16       17       18       19       20       21       22       23
{0,1,1}, {0,1,1}, {1,1,2}, {1,1,2}, {1,1,2}, {1,1,2}, {1,1,2}, {1,1,2},
//  24       25       26       27       28       29       30       31
}
```

4.4 B Frames

RV8 supports the same type of B frame mode as in H.263+ (Annex O B pictures) except no Bi-Dir prediction types. For the direct prediction mode the prediction type is determined by the reference macroblock prediction type (16x16 or 8x8), and is 16x16 with zero motion vector when the reference macroblock is INTRA or SKIPPED.
In B frames, there are four methods for motion compensating a macroblock - forward, backward, direct, and skipped. Forward and backward macroblocks are estimated and differentially encoded in a similar fashion to 16x16 MV's in a P frame, except the reference picture that is used can be either the preceding or future P frame, respectively.

A direct macroblock uses as a reference the motion vectors from the macroblock in the same spatial position in the future P frame. There may be one 16x16 motion vector, or four 8x8 motion vectors in the reference frame (if the reference macroblock is Intra coded, it is treated as a zero motion vector for these purposes). The forward and backward motion vectors are derived by scaling the reference motion vectors based on the relative distance between the B frame and the surrounding P frames. These derived motion vectors are then clipped to ensure that the referenced blocks lie within the padded image. The motion compensation prediction is formed by averaging the motion compensated block from the future P frame with the motion compensated block from the previous P frame. A weighted average is used, where the weighting factors are proportional to the temporal distance between the B frame and the corresponding P frame (iRatio0, iRatio1). The motion compensated residual is then transformed and coded. The chroma is compensated with the same scaled motion vectors.
The forward and backward motion vectors for direct mode macroblocks are calculated as follows.

$$MV_F = (TR_B * MV) / TR_D$$

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

Implemented as:

$iRatio0 = (TR_B << TR\_SHIFT) / TR_D;$

$MV_Fx = (iRatio0 * MV_{REF}x + TR\_RND) >> TR\_SHIFT$

$MV_Fy = (iRatio0 * MV_{REF}y + TR\_RND) >> TR\_SHIFT$

$MV_Bx = MV_Fx - MV_{REF}x$

$MV_By = MV_Fy - MV_{REF}y$

TR_SHIFT = 14

TR_RND   = (1 << (TR_SHIFT - 1)

$iRatio1 = ((TR_D - TR_B) << TR\_SHIFT) / TR_D;$

And Weighted Average:

U32 v1 = (U32) $pf_{i,j}$ << 7;

U32 v2 = (U32) $pp_{i,j}$ << 7;

U32 w = ((v1 * iRatio0) >> 16) + ((v2 * iRatio1) >> 16);

$pb_{i,j}$ = (U8) ((w + 0x10) >> 5);

pf = pixel from future reference frame

pb = pixel from prev reference frame

pb = predicted direct mode pixel

Where the vector component MV_F is the forward motion vectors, MV_B is the backward motion vector, and MV_{REF} represents the motion vectors in the corresponding macroblock in the subsequent reference picture. TR_D is the temporal distance between the temporally previous and next reference frame, and TR_B is the temporal distance between the current frame and previous reference frame. Since iRatio0 <=1, no clipping is needed for MV_F. Clipping is need for MV_B. The luma frame data is padded by 16 on each side, and the subpel interpolation filter is 4-tap.

```
right edge: pos_x*3 + MVx < (width + 16-16-2)*3
left edge:  pos_x*3 + MVx > -(16-1)*3
upper edge: pos_y*3 + MVy > -(16-1)*3
bottom edge: pos_y*3 + MVy < (height + 16-16-2)*3
```

assuming reference MV is ok.

In case the the corresponding macroblock in the subsequent reference picture is of type INTER_4V, four corresponding MV_F and MV_B's are calculated and four 8x8 such blocks are averaged.

A skipped macroblock in a B frame is motion compensated the same way as a direct macroblock, and it is understood that no transform coefficients are sent for the entire macroblock.

4.5 Reference Picture Resampling (RPR)

RV8 supports RPR in a manner identical to RVG2. (H263+ based) Reference picture resampling allows an encoder and decoder to change image dimensions on the fly, without having to generate a key frame.

Rather than generating a key frame for the first image having the new dimensions, the encoder simply interpolates the previous reference image to the new size before using it as a predictor for the next frame. The implementation is exactly like H263+ spec annexes O, P, and Q. All Edge displacement, Warping, and Fill parameters are zero. RPR information is transmitted in the SPO (once). The front-end uses the ILVC_MSG_ID_Set_RealVideo_RPR_Data to communicate this information to the back-end. Then, if num_sizes != 0, the back-end will read the following information from the slice header:

```
[TR (as before)]
      if num_sizes != 0
            PCTSZ = getbits(log2(num_sizes))
[MBA (as before)]
```

The maximum number of sizes is currently limited to 8. The decoder will use PCTSZ as an index to look up in the array transmitted in SPO:

```
new_width = horizontal_size[PCTSZ]
new_height = vertical_size[PCTSZ]
```

4.6 CPU Scalability

<Section to be expanded.>

Based on experiments the following Decoder CPU scalability is allowed.

Dropping B-frames.

- Snap to Integer Motion Vectors in B-frames.

Disable De-Blocking in B-Frames.

5 Bitstream Syntax

This is the specification of the bitstream syntax. The bitstream is not based on any standard and is not upward or downward compatible with other RealVideo Codecs.

5.1 Picture Layer

For RealVideo the SPO is used to signal global stream parameters. There is no Picture Header for RealVideo but the slice layer header has been kept. Every picture starts with a Slice Layer Header.

Every stream is Initialised with

32 bit SPO FLAG
32 bit Bitsream Version
Variable Number of Bytes representing (Width/4) followed by (Height/4) of the Re-Sampled Images.

Number of Re-Sampled Images can be got from RV20_SPO_BITS_NUMRESAMPLE_IMAGES. This is later used to the read the PCT SIZE variable in the slice header. PCT SIZE is represented in the bitstream by the least bits required to represent RV20_SPO_BITS_NUMRESAMPLE_IMAGES. The value being the index into the Width and Height Table constructed from the SPO information.

5.1.1SPO Flags

	Name	Mask	Description
1	RV20_SPO_FLAG_UNRESTRICTEDMV	0x00000001	Annex D
2	RV20_SPO_FLAG_EXTENDMVRANGE	0x00000002	IMPLIES NEW VLC TABLES
3	RV20_SPO_FLAG_ADVMOTIONPRED	0x00000004	ANNEX F
4	RV20_SPO_FLAG_ADVINTRA	0x00000008	ANNEX I

5	RV20_SPO_FLAG_INLOOPDEBLOCK	0x00000010	ANNEX J
6	RV20_SPO_FLAG_SLICEMODE	0x00000020	ANNEX K
7	RV20_SPO_FLAG_SLICESHAPE	0x00000040	0: free running; 1: rect
8	RV20_SPO_FLAG_SLICEORDER	0x00000080	0: sequential; 1: arbitrary
9	RV20_SPO_FLAG_REFPICTSELECTION	0x00000100	ANNEX N
10	RV20_SPO_FLAG_INDEPENDSEGMENT	0x00000200	ANNEX R
11	RV20_SPO_FLAG_ALTVLCTAB	0x00000400	ANNEX S
12	RV20_SPO_FLAG_MODCHROMAQUANT	0x00000800	ANNEX T
13	RV20_SPO_FLAG_BFRAMES	0x00001000	SETS DECODE PHASE
14	RV20_SPO_BITS_DEBLOCK_STRENGTH	0x0000e000	deblocking strength
15	RV20_SPO_BITS_NUMRESAMPLE_IMAGES	0x00070000	max of 8 RPR images sizes
16	RV20_SPO_FLAG_FRUFLAG	0x00080000	FRU BOOL: if 1 then OFF
17	RV20_SPO_FLAG_FLIP_FLIP_INTL	0x00100000	FLIP-FLOP interlacing
18	RV20_SPO_FLAG_INTERLACE	0x00200000	de-interlacing prefilter has been applied
19	RV20_SPO_FLAG_MULTIPASS	0x00400000	Encoded with multipass
20	RV20_SPO_FLAG_INV_TELECINE	0x00800000	Inverse-telecine prefilter has been applied
21	RV20_SPO_FLAG_VBR_ENCODE	0x01000000	Encoded using VBR

5.2 Slice Layer

Once the stream has been initialized, the RealVideo data is received as a series of slices that follow the syntax given in Figure 5.1. The Slice Header is indicated in this diagram as the first 9 fields of every slice.

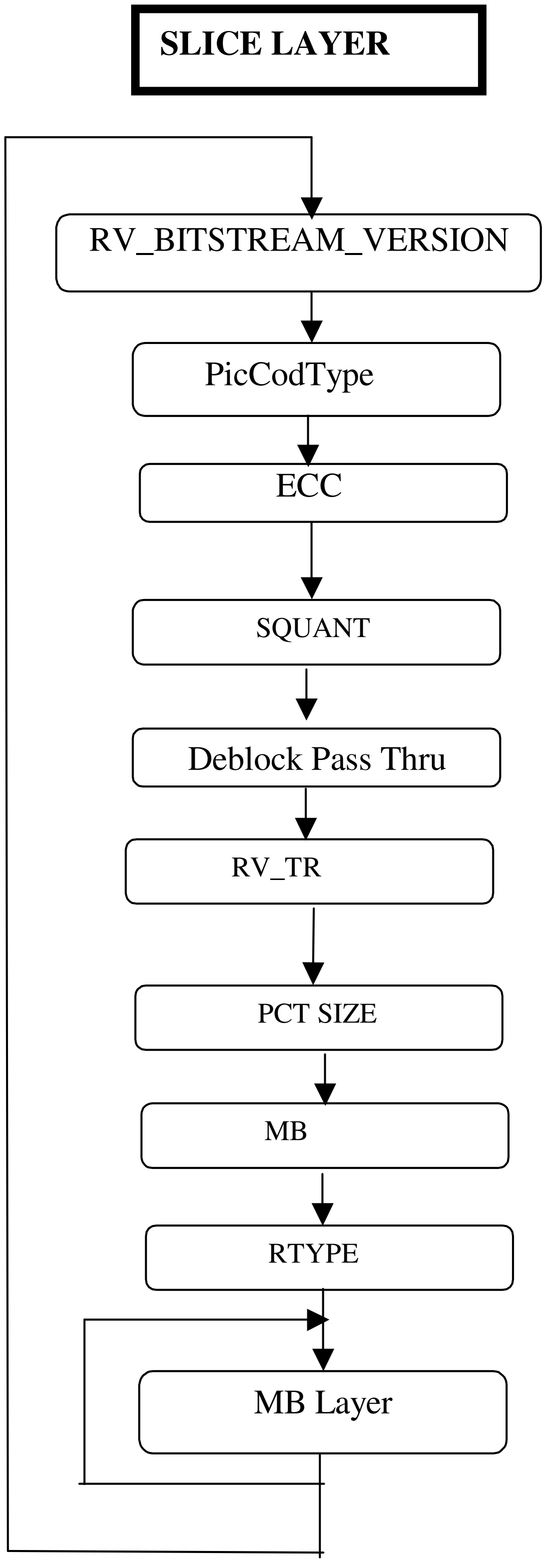


Figure 5.1 Slice Layer

Each slice in the bitstream is corresponds to a independently decodable section. Coding across a slice boundary is defined as in H.263. Motion vector prediction and intra mode prediction behaves as if the area outside the current slice is outside the picture.

Field Lengths

Field	Length	Description
FIELDLEN_RV_BITSTREAM_VERSION	3	
PicCodType	2	RV_INTRAPIC RV_FORCED_INTRAPIC RV_INTERPIC RV_TRUEBPIC
ECC	1	0
FIELDLEN_SQUANT	5	Quant
Deblock Pass Thru	1	1 Enable Deblocking filter
FIELDLEN_TR_RV	13	TR
PCT SIZE	See SPO	
MBA	Var	MBA_NumMBs= (width + 15)>>4 * (height + 15)>>4 - 1 MBA_FieldWidth 47 | 98 | 395 | 1583 | 6335 | 9215 6 | 7 | 9 | 11 | 13 | 14 SQCIF, QCIF, CIF, 4CIF, 16CIF, 2048x1152
FIELDLEN_RTYPE	1	Rounding

5.2.1ECC

When ECC bit is set the decoder shall skip that slice. ECC Packets contains forward error correction data and is not normative to the decoder. Layers above the decoder should perform the error correction and consume these packets.

5.3 Macroblock Layer

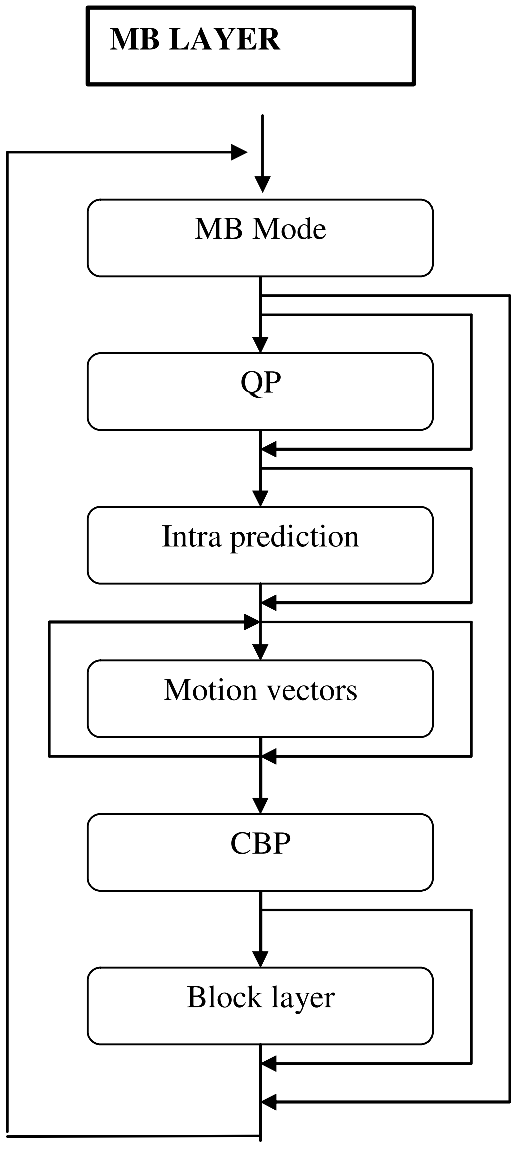


Figure 2 Macroblock Layer

5.3.1 MB Mode (Macroblock Mode)

The MB mode is signaled using the general VLC with code numbers as defined in the table below. If QP needs to be transmitted, one of the [QP] modes should be chosen.

N	**MB Mode (I)**	**MB Mode (P)**	**MB Mode (B)**
0	INTRA	SKIPPED	SKIPPED
1	INTRA_16x16	INTER	DIRECT
2		INTER_4V	FORWARD
3		INTRA	BACKWARD
4		INTRA 16x16	INTRA
5		INTER[QP]	INTRA 16x16
6		INTER_4V[QP]	DIRECT[QP]
7		INTRA[QP]	FORWARD[QP]
8		INTRA 16x16[QP]	BACKWARD[QP]
9			INTRA[QP]
10			INTRA 16x16[QP]

Table 2 MB modes

Skip in a P frame is defined as INTER prediction with zero motion vectors and no coefficients. Skip in a B frame is defined as Direct prediction and no coefficients.

5.3.2 QP (Quantization Parameter)

If the MB mode indicates that QP (0…31) should be transmitted, a 5 bit FLC is used, as in H.263. Note this is not available in I frames.

5.3.3 4x4 Intra Prediction

The signaling of the 4x4 intra prediction mode only occurs for INTRA 4x4 type macroblocks.

Since each of the 4x4 luma blocks shall be assigned a prediction mode, this will require a considerable number of bits if coded directly. We have therefore tried to find a more efficient way of coding the mode information. First of all we observe that the chosen prediction mode for a block is highly correlated with the prediction modes for adjacent blocks. We have three blocks, A, B, and C, as given in figure a) below. When the prediction modes of A and B are known (including the case that A or B or both are outside the picture) an ordering of the most probable, next most probable etc. mode of C can be found. This ordering table is listed below in the shortened length example Table 3 followed by the complete definition. For each prediction mode of A and B a list of 9 numbers is given.
In case of INTRA_16x16 the 4x4 block in consideration A, or B is given the mode number same as the INTRA_16x16 prediction mode. In case of other MBtypes, A and B are given mode number 0. (see section 5.3.4)

Example: The prediction mode for A and B is 1. The list "1 2 5 6 3 0 4 8 7" indicates that mode 1 is also the most probable mode for block C. Mode 2 is the next most probable one. In the bitstream there will for instance be information that Prob0 = 1 (see Table1) indicating that the next most probable mode after 1 shall be used for block C. In our example this means Intra prediction mode 2. If Prob0 had been 2, the mode would have been 5. Use of '-' in the table indicates that this mode can not occur in this position.

For more efficient coding, the intra prediction mode for two 4x4 luma blocks are coded in one codeword (Prob0 and Prob1 in the table below). The transmission order for the 8 codewords in one macroblock is indicated in figure b) below. The two probabilities are signaled using the general VLC with code numbers as defined below.

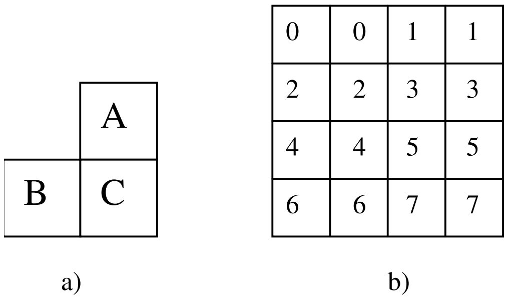


Figure 3 Intra prediction

B\A	outside	0	1
outside	0 - - - - - - - -	0 1 - - - - - - -	1 0 - - - - - - -
0	0 2 - - - - - - -	0 2 1 6 4 8 5 7 3	1 2 5 6 3 0 4 8 7
1	- - - - - - - - -	1 0 2 6 5 4 3 8 7	1 6 2 5 3 0 4 8 7
2	2 0 - - - - - - -	2 8 0 1 7 4 3 6 5	2 1 7 6 8 3 5 0 4
3	- - - - - - - - -	2 0 1 3 8 5 4 7 6	1 2 5 3 6 8 4 7 0
4	- - - - - - - - -	2 0 1 4 6 7 8 3 5	1 6 2 0 4 5 8 7 3
5	- - - - - - - - -	0 1 5 2 6 3 8 4 7	1 5 2 6 3 8 4 0 7
6	- - - - - - - - -	0 1 6 2 4 7 5 8 3	1 6 0 2 4 5 7 3 8
7	- - - - - - - - -	2 7 0 1 4 8 6 3 5	2 1 7 6 0 8 5 4 3
8	- - - - - - - - -	2 8 0 1 7 3 4 5 6	1 2 7 8 3 4 5 6 0

Table 3 Example Intra mode probability orders

Complete intra mode probability order definition, all modes:

```
// A=outside
{0,9,9,9,9,9,9,9,9}, // B=outside
{0,2,9,9,9,9,9,9,9}, // B=mode0
{9,9,9,9,9,9,9,9,9}, // B=mode1
{2,0,9,9,9,9,9,9,9}, // B=mode2
{9,9,9,9,9,9,9,9,9}, // B=mode3
{9,9,9,9,9,9,9,9,9}, // B=mode4
{9,9,9,9,9,9,9,9,9}, // B=mode5
{9,9,9,9,9,9,9,9,9}, // B=mode6
{9,9,9,9,9,9,9,9,9}, // B=mode7
{9,9,9,9,9,9,9,9,9}, // B=mode8

// A=mode0
{0,1,9,9,9,9,9,9,9},
{0,2,1,6,4,8,5,7,3},
{1,0,2,6,5,4,3,8,7},
{2,8,0,1,7,4,3,6,5},
{2,0,1,3,8,5,4,7,6},
{2,0,1,4,6,7,8,3,5},
{0,1,5,2,6,3,8,4,7},
{0,1,6,2,4,7,5,8,3},
{2,7,0,1,4,8,6,3,5},
{2,8,0,1,7,3,4,5,6},

// A=mode1
{1,0,9,9,9,9,9,9,9},
{1,2,5,6,3,0,4,8,7},
{1,6,2,5,3,0,4,8,7},
{2,1,7,6,8,3,5,0,4},
{1,2,5,3,6,8,4,7,0},
{1,6,2,0,4,5,8,7,3},
{1,5,2,6,3,8,4,0,7},
{1,6,0,2,4,5,7,3,8},
{2,1,7,6,0,8,5,4,3},
{1,2,7,8,3,4,5,6,0},
// A=mode2
{9,9,9,9,9,9,9,9,9},
{0,2,1,8,7,6,5,4,3},
{1,2,0,6,5,7,4,8,3},
{2,8,7,1,0,6,4,3,5},
{2,0,8,1,3,7,5,4,6},
{2,0,4,1,7,8,6,3,5},
{2,0,1,5,8,4,6,7,3},
{2,0,6,1,4,7,8,5,3},
{2,7,8,1,0,5,4,6,3},
{2,8,7,1,0,4,3,6,5},

// A=mode3
{9,9,9,9,9,9,9,9,9},
{0,2,1,3,5,8,6,4,7},
{1,0,2,5,3,6,4,8,7},
{2,8,1,0,3,5,7,6,4},
{3,2,5,8,1,4,6,7,0},
{4,2,0,6,1,5,8,3,7},
{5,3,1,2,8,6,4,0,7},
{1,6,0,2,4,5,8,3,7},
{2,7,0,1,5,4,8,6,3},
{2,8,3,5,1,0,7,6,4},

// A=mode4
{9,9,9,9,9,9,9,9,9},
{2,0,6,1,4,7,5,8,3},
{1,6,2,0,4,5,3,7,8},
{2,8,7,6,4,0,1,5,3},
{4,2,1,0,6,8,3,5,7},
{4,2,6,0,1,5,7,8,3},
{1,2,5,0,6,3,4,7,8},
{6,4,0,1,2,7,5,3,8},
{2,7,4,6,0,1,8,5,3},
{2,8,7,4,6,1,3,5,0},

// A=mode5
{9,9,9,9,9,9,9,9,9},
{5,1,2,3,6,8,0,4,7},
{1,5,6,3,2,0,4,8,7},
{2,1,5,3,6,8,7,4,0},
{5,3,1,2,6,8,4,7,0},
{1,6,2,4,5,8,0,3,7},
{5,1,3,6,2,0,8,4,7},
{1,6,5,2,0,4,3,7,8},
{2,7,1,6,5,0,8,3,4},
{2,5,1,3,6,8,4,0,7},

// A=mode6
{9,9,9,9,9,9,9,9,9},
{1,6,2,0,5,4,3,7,8},
{1,6,5,4,2,3,0,7,8},
{2,1,6,7,4,8,5,3,0},
{2,1,6,5,8,4,3,0,7},
{6,4,1,2,0,5,7,8,3},
{1,6,5,2,3,0,4,8,7},
{6,1,4,0,2,7,5,3,8},
{2,7,4,6,1,5,0,8,3},
{2,1,6,8,4,7,3,5,0},

// A=mode7
{9,9,9,9,9,9,9,9,9},
{2,0,4,7,6,1,8,5,3},
{6,1,2,0,4,7,5,8,3},
{2,7,8,0,1,6,4,3,5},
{2,4,0,8,3,1,7,6,5},
{4,2,7,0,6,1,8,5,3},
{2,1,0,8,5,6,7,4,3},
{2,6,4,1,7,0,5,8,3},
{2,7,4,0,8,6,1,5,3},
{2,8,7,4,1,0,3,6,5},

// A=mode8
{9,9,9,9,9,9,9,9,9},
{2,0,8,1,3,4,6,5,7},
{1,2,0,6,8,5,7,3,4},
{2,8,7,1,0,3,6,5,4},
{8,3,2,5,1,0,4,7,6},
{2,0,4,8,5,1,7,6,3},
{2,1,0,8,5,3,6,4,7},
{2,1,6,0,8,4,5,7,3},
{2,7,8,4,0,6,1,5,3},
{2,8,3,0,7,4,1,6,5}
```

Note that the table above is the decoder table. Since this is the decoder table, given the intra prediction mode for A and B, the mode for C can be found from prob0 or prob1, as transmitted in the bitstream.

The next table defines the VLC coding for the Prob0,Prob1 pairs. Given VLC code N, Prob0= dec_iprob[N*2] and Prob1=dec_iprob[N*2+1].

```
const U8 dec_iprob[162] = {
0,0,                      // 1 bit

0,1, 1,0,                 // 3 bits

1,1, 0,2, 2,0, 0,3, // 5 bits

3,0, 1,2, 2,1, 0,4,       // 7 bits
4,0, 3,1, 1,3, 0,5,

5,0, 2,2, 1,4, 4,1,       // 9 bits
0,6, 3,2, 1,5, 2,3,
5,1, 6,0, 0,7, 4,2,
2,4, 3,3, 6,1, 1,6,

7,0, 0,8, 5,2, 4,3,       // 11 bits
2,5, 3,4, 1,7, 4,4,
7,1, 8,0, 6,2, 3,5,
5,3, 2,6, 1,8, 2,7,
7,2, 8,1, 5,4, 4,5,
3,6, 6,3, 8,2, 4,6,
5,5, 6,4, 2,8, 7,3,
3,7, 6,5, 5,6, 7,4,

4,7, 8,3, 3,8, 7,5,       // 13 bits
8,4, 5,7, 4,8, 6,6,
7,6, 5,8, 8,5, 6,7,
8,6, 7,7, 6,8, 8,7,
7,8, 8,8,
};
```

Table 6 Intra prediction probability coding

5.3.4 16x16 Intra Prediction

The signaling of the16x16 intra prediction mode only occurs for INTRA 16x16 type macroblocks. The prediction mode for the 16x16 macroblock is coded as a 2-bit FLC:

Code	**Prediction**
00	DC
01	Vertical, from above
10	Horizontal, from left
11	Planar

5.3.5 Motion Vectors

The motion vectors are differentially encoded. The predictor is found in a way very similar to the description in H.263+, including how to handle the cases where the block size chosen for the current macroblock is larger than the block size for one or more of the surrounding macroblocks. With no special edge conditions the predictor is the median of the motion vectors to the left, above, and above right, relative to the current block. See Figure 4 for details. If the macroblock is coded in 8x8 mode, the median candidates for block 0 are found in the blocks marked with **boldface** numbers. If the macroblock is coded in 16x16 mode, the candidates are found from the blocks in *italic*.

If there is no block above right the current block, a candidate is instead found above left, or left if above left does not exist. This is different from H.263+, where the zero vector is used in this case. If there is no block above, the block to the left is used. If there is no block to the left, the zero vector is used. Motion vectors are restricted to values which go no further than sixteen pels beyond picture edges and reference pictures must be padded 16 pels beyond the edges (eight for chroma planes) by replicating the edge pixels.

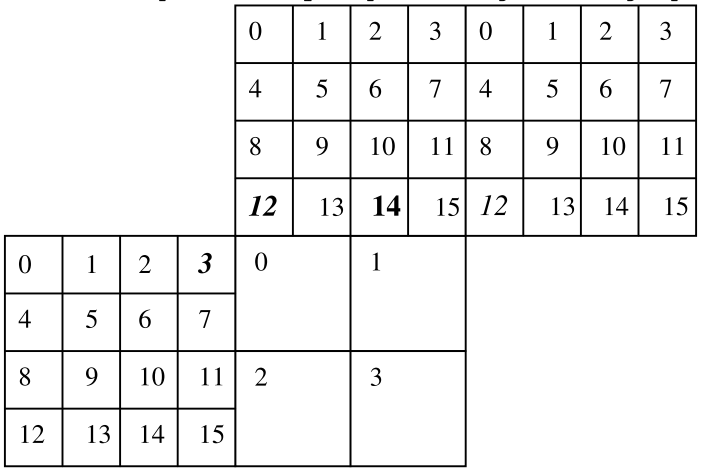
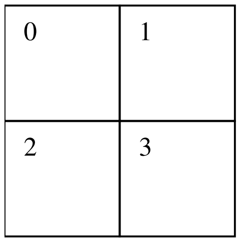


Figure 4 Motion vector prediction

Depending on the MB mode, from 0 to 4 motion vectors need to be transmitted. Each motion vector is transmitted as a horizontal and vertical component. The horizontal component is transmitted first, then the vertical component, and then the next vector. If more than one motion vector is to be sent, the transmission order is upper left block first, and then a regular right to left zig-zag scanning, as shown in

Figure 5 for 8x8 block sizes. See Table 4 for which code numbers to use. Each code number is transmitted using Structured UVLC. (see section 4.3.3)



8x8 block size

Figure 5 Motion vector transmission order

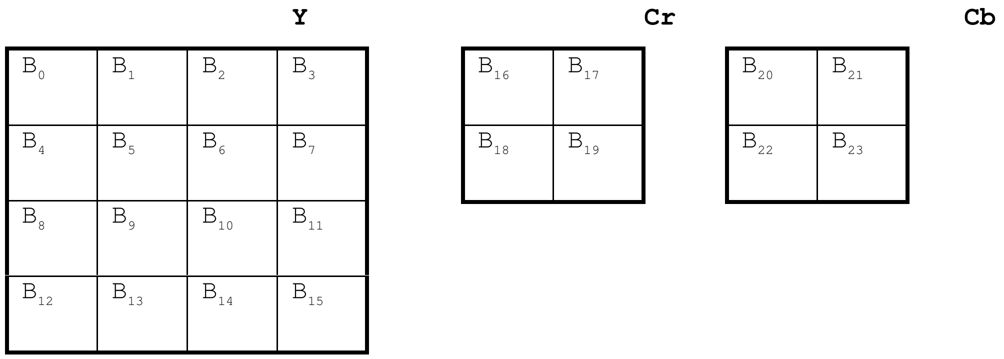
N	Vector
0	0
1	1
2	-1
3	2
4	-2
5	3
6	-3
7	4
8	-4
9	5
10	-5
11	6
12	-6
…	…

Table 4 Motion vector code numbers

5.3.6 CBP (Coded Block Pattern)

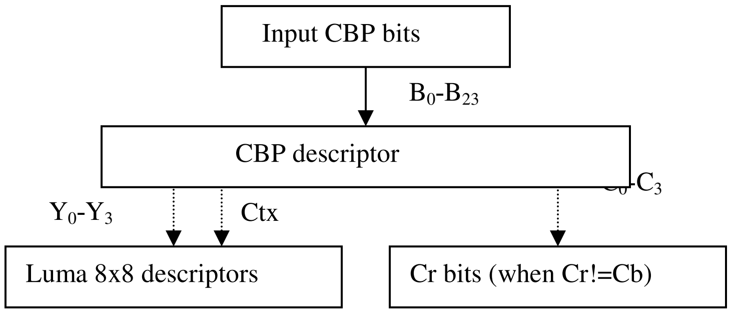
5.3.6.1 CBP length and bit order

CBP contains 24 bits representing 16 luminance blocks and 4 * 2 chrominance blocks in a macroblock. Bits that are set to 1 correspond to coded 4x4 blocks, bits that are set to 0 correspond to skipped (empty) blocks. The following diagram gives the correspondence between bits and luma/chroma blocks.



5.3.6.2 The structure of CBP code.

The overall structure of CBP codes is presented below.



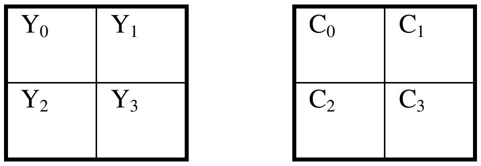
The main CBP object, CBP descriptor is transmitted first using one of the canonic Huffman codes (see Section 5.4.6) corresponding to the current macro-block type, and quantizer step size.

In turn, values of the components of CBP descriptor indicate the presence of the subsequent code objects: 8x8 descriptors and CR bits. Among these, 8x8 descriptors are transmitted first, using context-dependent canonic Huffman codes. CR bits required by the CBP descriptor are transmitted directly.

Below we describe each of these CBP code objects in details.

5.3.6.3 CBP descriptor.

CBP descriptor has the following components:



Composition rule:

$$\text{Cbp\_dsc} = ((((((C_0 * 3 + C_1) * 3 + C_2) * 3 + C_3) * 2 + Y_0) * 2 + Y_1) * 2 + Y_2) * 2 + Y_3;$$

Mappings between the CBP bits and descriptor's components are established as follows:

Y_0, Y_1, Y_2, Y_3	[B_0,B_1,B_4,B_5], [B_2, B_3, B_6, B_7], [B_8,B_9,B_{12},B_{13}], [B_{10},B_{11},B_{14},B_{15}]
0	all 4 bits = 0
1	at least 1 bit != 0 (8x8 descriptor to follow)

C_0, C_1, C_2, C_3	[B_{16},B_{20}], [B_{17}, B_{21}], [B_{18}, B_{22}], [B_{19},B_{23}]
0	both (Cr,Cb) bits = 0
1	only 1 bit (Cr or Cb) = 1 (extra bit to follow)
2	both (Cr,Cb) bits = 1

5.3.6.4 8x8 descriptor and contexts.

Each 8x8 descriptor is represented by a non-zero group of 4 bits $[B_0,B_1,B_4,B_5]$, $[B_2, B_3, B_6, B_7]$, $[B_8,B_9,B_{12},B_{13}]$, or $[B_{10},B_{11},B_{14},B_{15}]$ in CBP.

B_0	B_1
B_2	B_3

Composition rule:

$$8x8\_dsc = ((B_0 * 2 + B_1) * 2 + B_2) * 2 + B_3;$$

There are 4 different tables describing 8x8 descriptors based on their context:

$$Ctx = Y_0 + Y_1 + Y_2 + Y_3 -1;$$

where Y_0-Y_3 are the corresponding Y components of the CBP descriptor.

5.3.6.5 Cr bits.

Cr bits are transmitted every time when any of the C_0-C_3 CBP components is set to 1.

5.4 Block Layer

5.4.1 Block size, scan order, and types of coefficients.

Quantized DCT transform coefficients are encoded in blocks of 16 coefficients each, corresponding to their original 4x4 layout:

C_0	C_1	C_2	C_3
C_4	C_5	C_6	C_7
C_8	C_9	C_{10}	C_{11}
C_{12}	C_{13}	C_{14}	C_{15}

The following types of coefficients are encoded using separate groups of tables:

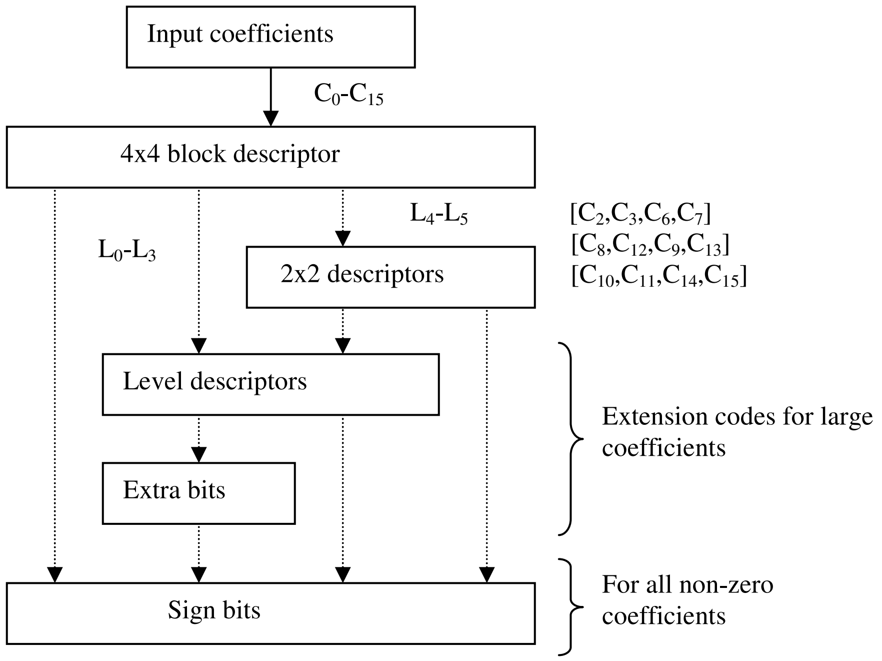
1. Luma coefficients from Inter-coded 4x4 blocks
2. Chroma coefficients from Inter-coded 4x4 blocks
3. Luma coefficients from 4x4-transformed Intra blocks

4. Chroma coefficients from 4x4-transformed Intra blocks
5. Luma DC coefficients from 16x16-transformed Intra blocks
6. Chroma DC coefficients from 16x16-transformed Intra blocks
7. Luma DC-removed coefficients from 16x16 transformed Intra blocks
8. Chroma DC-removed coefficients from 16x16 transformed Intra blocks.

To simplify the processing in the last two cases (dc-removed coefficients) the encoding is still done assuming there is a full 4x4 matrix of the coefficients, but the actual code tables are designed such that coefficient C_1 is always 0.

5.4.2 The structure of the code.

The code for each block of 16 coefficients has the following structure:



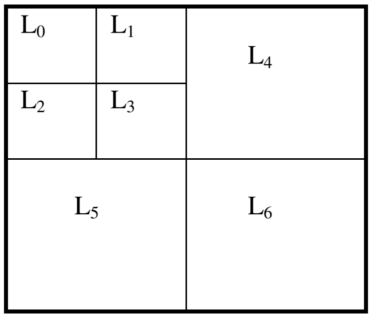
The main code object, 4x4 block descriptor is transmitted first. Based on the values of its components, subsequent code objects: 2x2 descriptors, level descriptors, and sign bits may follow.

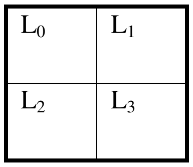
The order of these code objects follows the natural order of components in descriptors. E.g. if 4x4 descriptor indicates that there is a large DC coefficient, then, the next code object is its Level descriptor. If the level descriptor is not sufficient to represent the absolute value of this coefficient exactly, it will indicate how many Extra bits will follow. The Sign bit is transmitted right after.

Below we describe each of these code components.

5.4.3 4x4 and 2x2 block descriptors.

4x4 and 2x2 descriptors have the following components:





Composition rules:

$4x4\_dsc = ((((L_0 * 3 + L_1) * 3 + L_2) * 3 + L_3) * 2 + L_4) * 2 + L_5) * 2 + L_6;$

$2x2\_dsc = ((L_0 * 3 + L_1) * 3 + L_2) * 3 + L_3;$

Mappings between coefficients' values and descriptor's components:

L_0	C_0, C_2, C_8, C_{10}
0	0
1	+1,-1
2	+2,-2
3	>2,<-2 (escape code)

L_1, L_3, L_3	$C_1, C_4, C_5, C_3, C_6, C_7, C_{12}, C_9, C_{13}, C_{11}, C_{14}, C_{15}$
0	0
1	+1,-1
2	>1, <-1 (escape code)

L_4, L_5, L_6	$[C_2, C_3, C_6, C_7], [C_8, C_{12}, C_9, C_{13}], [C_{10}, C_{11}, C_{14}, C_{15}]$
0	0 (all 4 coefficients = 0)
2	!0 (escape code)

The encoding of 2x2 descriptors for L_4 and L_5 blocks is done using the same tables (with an inverse scan order of coefficients in the L_5 block). The 2x2 descriptor for block L6 is encoded using separate tables.

5.4.4 Level descriptors.

When coefficients are large (which is signalized by escape codes in 4x4 or 2x2 descriptors), their absolute residual values are transmitted using additional level descriptors and extension bits as specified below:

Level descriptors	Extra bits	Absolute residual values
0-22	0	0-22
23	1	23-24
24	2	25-28
25	3	29-36
26	4	37-52
27	5	53-84
28	6	85-148
29	7	147-274
30	8	275-530

5.4.5 Sign bits.

Sign bits are transmitted for all non-zero coefficients following the description of their absolute values (by the corresponding combination of 4x4, 2x2, or level descriptors).

Encoding of all non-zero DCT coefficients is done in the order as they appear in 4x4 and 2x2 descriptors.

5.4.6 Code Tables.

5.4.6.1 Partition of code tables based on Inter/Intra coding and quantization step sizes.

The tables for all code components are separate for Inter- and Intra-coded macroblocks. Additionally, different code tables are used based on QP values used to encode macroblocks. The mappings between QPs and indices of code tables are provided below.

QP range partition for Intra-coded macroblocks	
Region #	QP range
0	0-9
1	10-15
2	16-20
3	21-25
4	26-30

QP range partition for Inter-coded macroblocks	
Region #	QP range
0	0-6
1	7-10
2	11-14
3	15-18
4	19-22
5	23-26
6	27-30

5.4.6.2 Variable-length codes and code tables.

Variable length codes represent sequences of bits packed in bytes such that earlier bits correspond to the leftmost (more significant) digits within a byte.

Encoding and decoding algorithms discussed herein employ Canonic Huffman Codes (see, e.g., A.Moffat, and A.Turpin, "On the Implementation of Minimum-Redundancy Prefix Codes", IEEE Transactions on Communications, 45(10): 1200-1207, 1997).

For the description of such codes we will only specify code lengths. The reconstruction of the corresponding codewords can be accomplished using the following algorithm.

```
/*
 * Given:    n - the number of codes, and len[] - code lengths
 * Produces: code[] - canonic Huffman codewords
 */
make_code (int n, unsigned char *len, unsigned int *code)
{
    unsigned int leaves [MAX_DEPTH+1], start [MAX_DEPTH+2];
    register int i;

    /* count the number of leaves on each level: */
    for (i = 0; i <= MAX_DEPTH; i++) leaves [i] = 0;
    for (i = 0; i < n; i++) leaves [len [i]]++;

    /* set start codes for each level: */
    start [1] = 0;
    for (i = 1; i <= MAX_DEPTH; i++)
        start [i + 1] = (start [i] + leaves [i]) * 2;

    /* assign codewords: */
    for (i = 0; i < n; i++)
        code [i] = start [len [i]]++;
}
```

5.4.6.3 Code tables.

The following tables represent lengths of the canonic Huffman codes for all the above described components of codes for transform coefficients and CBP types.

```
/* intra tables: */
char intra_cbp[MAX_INTRA_QP_REGIONS][2][MAX_CBP] = {};
char intra_8x8_dsc[MAX_INTRA_QP_REGIONS][2][4][MAX_8x8_DSC] = {};
char intra_luma_4x4_dsc[MAX_INTRA_QP_REGIONS][3][MAX_4x4_DSC] = {};
char intra_luma_2x2_dsc[MAX_INTRA_QP_REGIONS][2][MAX_2x2_DSC] = {};
char intra_chroma_4x4_dsc[MAX_INTRA_QP_REGIONS][MAX_4x4_DSC] = {};
char intra_chroma_2x2_dsc[MAX_INTRA_QP_REGIONS][2][MAX_2x2_DSC] = {};
char intra_level_dsc[MAX_INTRA_QP_REGIONS][MAX_LEVEL_DSC] = {};

/* inter tables: */
char inter_cbp[MAX_INTER_QP_REGIONS][MAX_CBP] = {};
char inter_8x8_dsc[MAX_INTER_QP_REGIONS][4][MAX_8x8_DSC] = {};
char inter_luma_4x4_dsc[MAX_INTER_QP_REGIONS][MAX_4x4_DSC] = {};
char inter_luma_2x2_dsc[MAX_INTER_QP_REGIONS][2][MAX_2x2_DSC] = {};
char inter_chroma_4x4_dsc[MAX_INTER_QP_REGIONS][MAX_4x4_DSC] = {};
char inter_chroma_2x2_dsc[MAX_INTER_QP_REGIONS][2][MAX_2x2_DSC] = {};
char inter_level_dsc[MAX_INTER_QP_REGIONS][MAX_LEVEL_DSC] = {};
```

6 QA Test Procedures

See QA test procedures doc.

7 References

[1] ISO/IEC 14496-2, "Information technology - Generic coding of audio-visual objects: Visual,", March 1999.

[2] G. Sullivan and T. Wiegand, "Rate-Distortion Optimization for Video Compression," IEEE Signal Processing Magazine, Vol. 15, No. 6, Nov. 1998.

[3] G. Bjontegaard, "Response to Call for Proposals for H.26L," Q15-F-11, ITU-T Advanced Video Meeting, Seoul, Nov. 98, ftp://standard.pictel.com/video-site/9811_Seo/q15f11.doc .

[4] G. Bjontegaard, "Enhancement of the Telenor proposal for H.26L," Q15-G-25, ITU-T Advanced Video Meeting, Monterey, Feb. 99, ftp://standard.pictel.com/video-site/9902_Mon/q15g25.doc .

[5] G. Bjontegaard, "Adding Intra mode suitable for coding of flat regions," COM-16 D.360, ITU-T Advanced Video Meeting, Geneva, Feb. 2000, ftp://standard.pictel.com/video-site/0002_Gen/Telenor_intra.doc.

[6] Gary Sullivan, "Draft Text of Recommendation H.263 Version 2 ("H.263+") for Decision"

[7] Gregory J. Conklin, Gary S. Greenbaum, Karl O. Lillevold, Alan F. Lippman and Yuriy A. Reznik, "Video Coding for Streaming Media Delivery on the Internet," IEEE Transactions on Circuits and Systems for Video Technology.

8 Annex A

RealVideo Decoders are Split into 2 parts, RealVideo Frontend and the decoder Backend.

RealVideo Frontend: Exposes the RealMedia Codec Interface.
The Frontend handles all the Initialization, pre-post filtering, frame-rate up sampling, statistics, and scalability decisions.

RealVideo Backend: Exposes the Hive/PIA Codec Interface.
The Backend decodes the Bitstream. The Backend maybe referred to as ILVC in general or in RV8 by codename "Tromsø" to refer to specific algorithms.

Back End Interface:

```
RV20toYUV420Init (RV10_INIT *prv10Init, void **decoderState)
RV20toYUV420Free (void *global)
RV20toYUV420Transform (
              UCHAR     *pRV20Packets,
              UCHAR     *pDecodedFrameBuffer,
              void      *pInputParams, // H263DecoderInParams
              void      *pOutputParams, // H263DecoderOutParams
              void      *global )
RV20toYUV420CustomMessage (
              PIA_Custom_Message_ID *msg_id, void *global
              )
```

The RV20toYUV420CustomMessage function exposes decoder interfaces that are specific to the "ILVC" decoder. These interfaces are defined in "ilvcmsg.h".

```
RV20toYUV420HiveMessage (ULONG32 *msg_id, void *global)
```

The RV20toYUV420HiveMessage function exposes decoder interfaces that may be applicable to a variety of decoders, not just to "ILVC". The 'msg' parameter points to a ULONG32 that identifies a particular interface or feature. This ULONG32 is actually the first member in a larger struct, similar to the PIA_Custom_Message_ID usage. See "hivervi.h" for a complete list of supported messages.

```
typedef struct tagRV10_INIT
{
    UINT16 outtype;
    UINT16 pels;
    UINT16 lines;
    UINT16 nPadWidth;
    /* number of columns of padding on right to get 16 x 16 block*/
    UINT16 nPadHeight;
    /* number of rows of padding on bottom to get 16 x 16 block*/
    UINT16 pad_to_32;
    // to keep struct member alignment independent of compiler options
    ULONG32 ulInvariants;
    // ulInvariants specifies the invariant picture header bits -- SPO
    LONG32 packetization;
    ULONG32 ulStreamVersion;
} RV10_INIT;

typedef struct tag_H263DecoderInParams
{
    ULONG32 dataLength;
    LONG32    bInterpolateImage;
    ULONG32 numDataSegments;
    PNCODEC_SEGMENTINFO *pDataSegments;
    ULONG32 flags;
    // 'flags' should be initialized by the front-end before each
    // invocation to decompress a frame.  It is not updated by the
    // decoder.
    // If it contains RV_DECODE_MORE_FRAMES, it informs the decoder
    // that it is being called to extract the second or subsequent
    // frame that the decoder is emitting for a given input frame.
    // The front-end should set this only in response to seeing
    // an RV_DECODE_MORE_FRAMES indication in H263DecoderOutParams.
    // If it contains RV_DECODE_DONT_DRAW, it informs the decoder
    // that it should decode the image (in order to produce a valid
    // reference frame for subsequent decoding), but that no image
    // should be returned.  This provides a "hurry-up" mechanism.
    ULONG32 timestamp;
} H263DecoderInParams;

typedef struct tag_H263DecoderOutParams
{
    ULONG32 numFrames;
    ULONG32 notes;
    //'notes' is assigned by the transform function during each call to
    // decompress a frame.  If upon return the notes parameter contains
    // the indication RV_DECODE_MORE_FRAMES, then the front-end
    // should invoke the decoder again to decompress the same image.
    // For this additional invocation, the front-end should first set
    // the RV_DECODE_MORE_FRAMES bit in the 'H263DecoderInParams.flags'
    // member, to indicate to the decoder that it is being invoked to
    // extract the next frame.
    // The front-end should continue invoking the decoder until the
    // RV_DECODE_MORE_FRAMES bit is not set in the 'notes' member.
    // For each invocation to decompress a frame in the same
    // "MORE_FRAMES"
    // loop, the front-end should send in the same input image.
    //
    // If the decoder has no frames to return for display, 'numFrames'
    // will be set to zero.  To avoid redundancy, the decoder does
    // *not* set the RV_DECODE_DONT_DRAW bit in 'notes' in this case.

    ULONG32 timestamp;
    // The 'temporal_offset' parameter is used in conjunction with the
    // RV_DECODE_MORE_FRAMES note, to assist the front-end in
    // determining when to display each returned frame.
    // If the decoder sets this to T upon return, the front-end should
    // attempt to display the returned image T milliseconds relative to
    // the front-end's idea of the presentation time corresponding to
    // the input image.
    // Be aware that this is a signed value, and will typically be
    // negative.

    ULONG32 width;
    ULONG32 height;
    // Width and height of the returned frame.
    // This is the width and the height as signalled in the bitstream.
} H263DecoderOutParams;
```

9 Annex B

9.1 Encoder Command line Interface

Usage: tromsoe infile [options]
In the following syntax descriptions, arglist is a comma-separated list of the form \"arg[=value],arg[=value],...\". Some arguments take values, some do not. If arglist contains any whitespace, it must be enclosed in quotes. For example, -d 4,l=mylog.txt,a specifies the debug level to be 4, that the debug log file is named \"mylog.txt\", and that the file should be opened in append mode rather than being overwritten.

Infile	Specify raw YUV12 input file
-a letter,arglist	Enable a specific H.263 annex.
Letter	letter is mandatory and must be the first argument in arglist. It is the annex's upper case letter. For some annexes, this letter argument takes a value, as described below. The remaining elements of arglist are specific to each annex. For annexes that can be applied on a per-layer basis, arglist can contain \"l=<level>\", indicating the option is being applied to the given level.
K[=<bytes_per_slice>]	Slice Structured Mode [default slice size is 512]
O	Add a new scalability layer. First O option describes layer 0, second layer 1, etc (deprecated). Options include:
w=<width>	Layer width [default: layer 0 QCIF, layer n prev] (deprecated)
h=<height>	Layer height [default: layer 0 QCIF, layer n prev] (deprecated)
p=profile	String profile of 'P's, 'B's and '-'s [default \"P\"]
r=<ref_layer>	[default: 0 for layer 0, n - 1 for layer n] (deprecated)
-z	Enable Interlaced Encode.
-b <image_range>	Images to encode [encode all by default]
	<image_range> is <m>-<n>:
	3-5 means frames 3, 4 and 5
	4- means frames 4 and beyond
	-5 means frames 0 through 5
	7 means frames 0 through 7
-c <cpu_usage>	Specify CPU scalability setting. cpu_usage is a number between 0 and 100
-d arglist	Specify debugging output.
<level>	Detail level. Use -1 to suppress. [default is 0]
l=logfile	Output file for debug messages. [default is stdout]
a	Append to logfile. [default is to overwrite]
-f arglist	Specify format of compressed output file.
r	Use raw format. [default]

x	Use extended raw format.
-h	Display this command line help and exit.
-i arglist	Specify input file format.
w=<width>	Source image width [default is 176]
h=<height>	Source image height [default is 144]
fps=<frame_rate>	Source frame rate [default is 30 fps] (deprecated)
sf=<skip_factor>	Source frames to skip between each encoded frame [0]
pcf=<clock_freq>	Picture clock frequency [default is 29.97] (deprecated)
par=par_description	Pixel aspect ratio. par_description is a string (deprecated)
-m <speed>	Specify machine clock rate in MHz.
-r mode,arglist	Specify data rate control for non-I frames.
mode	mode is mandatory and must be the first argument in arglist. It is one of the following strings:
q[=<qual>]	Use PIA_RCM_QUALITY with the given quality [5000].
Q[=<qp>]	Map fixed QP into PIA_RCM_QUALITY [5000].
fs=<frame_size>	(deprecated)
fd	(deprecated)
q=<quality>	Specifies minimum quality level in range 0 .. 10000. [default is 0].
fps=<frame_rate>	(deprecated)
d=<data_rate>	(deprecated)
kb=<data_rate>	(deprecated)
B=<QP>	Use the given QP for B frames
l=<layer>	(deprecated)
-k mode,arglist	Specify rate control for I frames.
mode	mode is mandatory and must be the first argument in arglist. It is one of the following strings:

i=<interval>
interval = 0
interval > 0
	Specify key frame period.
	Use PIA_KFCM_AUTO.
	Use PIA_KFCM_INTERVAL, with the given interval.

a	Use PIA_RCM_AUTO rate control [this is the default].
q[=<quality>]	Use PIA_RCM_QUALITY with the given quality [5000].
fs=<frame_size>	Use method PIA_RCM_FRAME_SIZE with the given target frame size (in bytes). (deprecated)
q=<quality>	Specifies quality level in range 0 .. 10000. [default is 5000 for mode PIA_RCM_QUALITY, else 0].
l=<layer>	(deprecated)
-o outfile	Specify output file. Output suppressed if unspecified.
-q	Quiet mode (no summary statistics).
-v	Verbose mode. Displays progress messages and statistics about the compressed bitstream to stdout.

9.2 Decoder Command line Interface

Usage: tromsod infile [options]
In the following syntax descriptions, arglist is a comma-separated list of the form \"arg[=value],arg[=value],...\". Some arguments take values, some do not. If arglist contains any whitespace, it must be enclosed in quotes. For example, -d 4,l=mylog.txt,a specifies the debug level to be 4, that the debug log file is named \"mylog.txt\", and that the file should be opened in append mode rather than being overwritten.

infile	Specify TROMSO bitstream input file.
-b <image_range>	Images to decode [decode all by default] <image_range> is <m>-<n>: 3-5 means frames 3, 4 and 5 4- means frames 4 and beyond -5 means frames 0 through 5 7 means frames 0 through 7
-d arglist	Specify debugging output.
<level>	Detail level. Use -1 to suppress. [default is 0]
l=logfile	Output file for debug messages. [default is stdout]
a	Append to logfile. [default is to overwrite]
-e arglist	Specify post filtering options
smoothing	Smoothing [default is off]
-f arglist	Specify display attributes (deprecated)
-h	Display this command line help and exit.
-i arglist	Specify input file format.
w=<width>	Compressed image width [default is 176]
h=<height>	Compressed image height [default is 144]
-l	Enable latency mode [default is off]
-m <speed>	Specify machine clock rate in MHz. (WIN32 IA only)
-o outfile	Specify output file. Output suppressed if unspecified.
-p	Enable smoothing postfilter [default is off]
-q	Quiet mode. Suppresses display of summary information.
-v	Verbose mode. Displays progress messages to stdout.
-x arglist	Specify packet loss characteristics.
<percent>	Percent packet loss [default is 0].