RealNetworks

RealVideo 9

External Specification

Video and Audio Technologies
Codec Group
RealNetworks, Inc

November 7, 2003

Version 1.4

Summary

This document is the draft specification of RealVideo9 Codec. RealVideo9 achieves new levels of compression performance at low as well as high data rates. The improvements are due in part to 1/4 pixel interpolation for motion estimation, the addition of 16x8, 8x16 pixel motion compensated blocks to the 8x8 and 16x16 blocks, medium complexity motion estimation technique with better RD characteristics, 16x16 double transforms, better in-loop filter, efficient coding of 4x4 intra prediction modes, run length coding of MB-Types, and more efficient variable length coding by symbol manipulation and adaptive coding. The algorithm also benefits from black level filter, and noise reduction pre-filtering. RealVideo9 does not need a post filter. RealVideo9 adds new Interlace capability to the Codec.

Revision History:

Revision	Date	Comment
1.0	9/18/02	Initial Decoder Specification
1.1	9/20/02	Improvements and Matching to RV8 Spec.
1.2	10/31/02	Fix Typos, Errors, and more Clarification based on AcerAli_RV9_question.doc

Table of Content

1 High Level Overview

RealVideo 9 represents major advances in compression performance. RealVideo 9 achieves new levels of compression performance at low as well as high data rates. The improvements are due in part to 1/4 pixel interpolation for motion estimation, the addition of 16x8 and 8x16 pixel motion compensated blocks to the 8x8 and 16x16 blocks, medium complexity motion estimation technique with better RD characteristics, 16x16 double transforms, better in-loop filter, efficient coding of 4x4 intra prediction modes, run length coding, and more efficient variable length coding by symbol manipulation and adaptive coding. RealVideo 9 doesn't need a post filter. RealVideo 9 adds new Interlace capability to the Codec. RealVideo 9 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™ 200 MHz with 16 MB of memory.

Target bit rates: < 20 kbps, 30 kbps, 100 kbps, 500 kbps, 1–2 Mbps DVD quality bit rates, HDTV bit rates, and above.

Target frame sizes: minimum frame size is 32x32, with particular attention to the range CIF (352 x 288) to VGA Resolution (640 x 480).

Video quality requirements: A noticeable improvement in video quality over RealVideo 8 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, RealVideo9 delivers a quantum jump in compression efficiency.

4.1 Introduction

The RealVideo 9 and RealVideo 8 algorithm is largely based on H.26L or the Joint Video Team proposal Mpeg4 part 10 / Advanced Video Codec, which experiments have shown provides significant and very visible coding gains over Mpeg4v2/H263+.
RealVideo9 deviates from 26L by:

- not performing the chroma DC coefficient manipulation (although a lower chroma DC quantizer achieves almost the exact same result)
- not including the 8x4, and 4x8 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.
- Double transform for Inter and Intra MBs.
- Quantizer Matrix for Double Transform.
- Improved Intra prediction mode coding.
- Adaptive MB Type coding.
- Simplified Interlace coding modes.
- Allows RPR

Changes w.r.t. RealVideo 8 are:

- Improved Intra mode coding
- Advanced Deblocking Filter
- No Post filter required
- Quarter Pel Motion Estimation (includes the Funny position)
- 16x8 and 8x16 motion compensation
- Double Transform for Inter 16x16
- New QP Matrix for Double Transform
- Optimized Entropy coding through explicit Super VLC quantizer.
- Adaptive MB Types
- Run length encoding of Skip Modes
- Better B Frame motion vector prediction
- Bidirectional MB Type for B frames
- Interlaced Coding
- New Picture Size scheme to allow splicing of Files.

4.2 Overview

RealVideo 9 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.

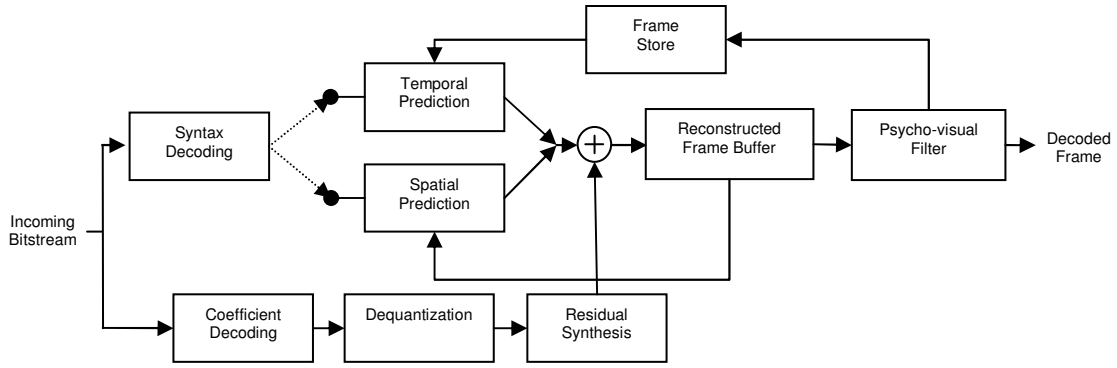


Figure 4.1: Block diagram of the RealVideo 9 decoder algorithm

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 and, in addition, used for future temporal prediction.

The RealVideo 9 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.

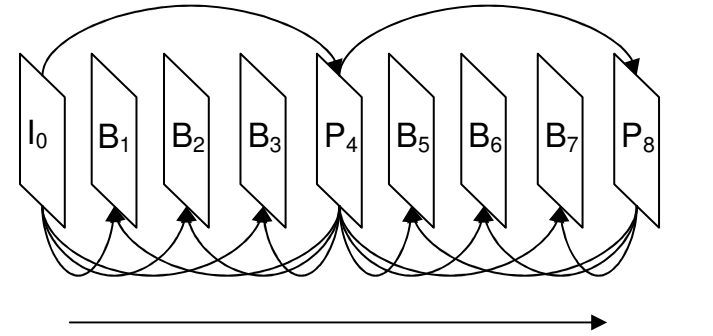
4.2.1 Picture Types

There are 3 picture types in RealVideo 9 - I-Pictures, P-Pictures and B-Pictures.

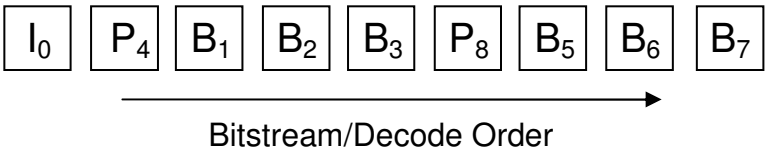
I-Pictures are also referred to as Intra-Frames or Key Frames. They do not use temporal prediction and, therefore, do not require other decoded reference frames to be in the decoder for proper reconstruction. I-Pictures provide entry or access points to the video sequence.

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.



(a)



(b)

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.

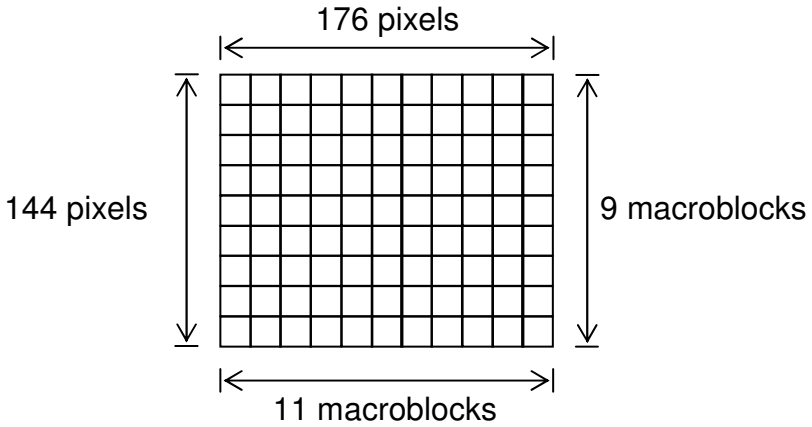


Figure 4.3: A picture with 11 x 9 macroblocks (QCIF picture)

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.

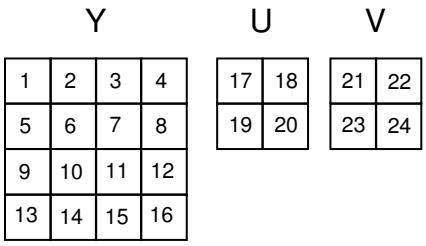


Figure 4.4: Macroblock scanning order of 4x4 blocks during residual coding

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_16x16	Inter, 1MV, Dbl Xfrm		X	
INTER_16x8V	Inter, 2MVs for 2 16x8 blocks		X	
INTER_8x16V	Inter, 2MVs for 2 8x16 blocks		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
BIDIR	Fwd & Bwd MV for 16x16 block			X
SKIPPED	Direct, no residual, Derived MV for 16x16 block			X

4.3.2 1/4 sub-pel prediction

Motion vectors in RealVideo 9 are transmitted in 1/4 pixel units. Motion vectors always point to reference picture relative to MB position in the decoded picture. When the motion vectors for a macroblock have been decoded the full-pixel offset can be obtained by shifting right by 2 bits.

```
MVx_int = (MVx_luma >> 2)
MVy_int = (MVy_luma >> 2).
```

The "phase" or sub-pixel location can be obtained by extracting the 2 least significant bits.

```
MVx_sub = (MVx_luma & 3)
MVy_sub = (MVy_luma & 3).
```

The integer pixels used in the intrepolation are the actual pixels of the reference picture and/or the padded pixels. The MV is illegal if it requires pixels beyond the padded reference image.

For luma sub-pixel interpolation is calculated with a 6-tap filter. For chroma, a 2-tap filter is used. In addition, one of the 16 interpolated pixels, MVx_sub = 3, MVy_sub = 3, in the luma plane is created using a stronger filter. 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_0 = integer pixels, t_0 = temporary buffer, v_0 = interpolated image
(0,0)	$t_0 = p_0$ $v_0 = t_0$
(0,1)	$t_0 = p_0$ $v_0 = (t_{-2} - 5t_{-1} + 52t_0 + 20t_1 - 5t_2 + t_3 + 32) >> 6$
(0,2)	$t_0 = p_0$ $v_0 = (t_{-2} - 5t_{-1} + 20t_0 + 20t_1 - 5t_2 + t_3 + 16) >> 5$
(0,3)	$t_0 = p_0$ $v_0 = (t_{-2} - 5t_{-1} + 20t_0 + 52t_1 - 5t_2 + t_3 + 32) >> 6$
(1,0)	$t_0 = (p_{-2} - 5p_{-1} + 52p_0 + 20p_1 - 5p_2 + p_3 + 32) >> 6$ $v_0 = t_0$
(1,1)	$t_0 = (p_{-2} - 5p_{-1} + 52p_0 + 20p_1 - 5p_2 + p_3 + 32) >> 6$ $v_0 = (t_{-2} - 5t_{-1} + 52t_0 + 20t_1 - 5t_2 + t_3 + 32) >> 6$
(1,2)	$t_0 = (p_{-2} - 5p_{-1} + 52p_0 + 20p_1 - 5p_2 + p_3 + 32) >> 6$ $v_0 = (t_{-2} - 5t_{-1} + 20t_0 + 20t_1 - 5t_2 + t_3 + 16) >> 5$
(1,3)	$t_0 = (p_{-2} - 5p_{-1} + 52p_0 + 20p_1 - 5p_2 + p_3 + 32) >> 6$ $v_0 = (t_{-2} - 5t_{-1} + 20t_0 + 52t_1 - 5t_2 + t_3 + 32) >> 6$
(2,0)	$t_0 = (p_{-2} - 5p_{-1} + 20p_0 + 20p_1 - 5p_2 + p_3 + 16) >> 5$ $v_0 = t_0$
(2,1)	$t_0 = (p_{-2} - 5p_{-1} + 20p_0 + 20p_1 - 5p_2 + p_3 + 16) >> 5$ $v_0 = (t_{-2} - 5t_{-1} + 52t_0 + 20t_1 - 5t_2 + t_3 + 32) >> 6$
(2,2)	$t_0 = (p_{-2} - 5p_{-1} + 20p_0 + 20p_1 - 5p_2 + p_3 + 16) >> 5$ $v_0 = (t_{-2} - 5t_{-1} + 20t_0 + 20t_1 - 5t_2 + t_3 + 16) >> 5$
(2,3)	$t_0 = (p_{-2} - 5p_{-1} + 20p_0 + 20p_1 - 5p_2 + p_3 + 16) >> 5$ $v_0 = (t_{-2} - 5t_{-1} + 20t_0 + 52t_1 - 5t_2 + t_3 + 32) >> 6$
(3,0)	$t_0 = (p_{-2} - 5p_{-1} + 20p_0 + 52p_1 - 5p_2 + p_3 + 32) >> 6$ $v_0 = t_0$
(3,1)	$t_0 = (p_{-2} - 5p_{-1} + 20p_0 + 52p_1 - 5p_2 + p_3 + 32) >> 6$ $v_0 = (t_{-2} - 5t_{-1} + 52t_0 + 20t_1 - 5t_2 + t_3 + 32) >> 6$
(3,2)	$t_0 = (p_{-2} - 5p_{-1} + 20p_0 + 52p_1 - 5p_2 + p_3 + 32) >> 6$ $v_0 = (t_{-2} - 5t_{-1} + 20t_0 + 20t_1 - 5t_2 + t_3 + 16) >> 5$
(3,3)	$t_0 = p_0 + p_1$ $v_0 = (t_0 + t_1 + 2) >> 2$

The value of t_0 is clipped to 0-255 before calculating v_0.The final value of v_0 is again clipped to the range 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 >> 2)
MVy_chroma_int = (MVy_chroma >> 2).

MVx_chroma_sub = (MVx_chroma & 3)
MVy_chroma_sub = (MVy_chroma & 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 the rounding or addition factor for each sub-pixel location. In addition, note that the (3,3) position is the same as the (2,2) position.

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)	$f_{0,0} = p_{0,0}$
(0,1)	$f_{0,0} = (3p_{0,0} + p_{1,0} + 2) >> 2$
(0,2)	$f_{0,0} = (p_{0,0} + p_{1,0}) >> 1$
(0,3)	$f_{0,0} = (p_{0,0} + 3p_{1,0} + 2) >> 2$
(1,0)	$f_{0,0} = (3p_{0,0} + p_{0,1} + 1) >> 2$
(1,1)	$f_{0,0} = (9p_{0,0} + 3p_{0,1} + 3p_{1,0} + p_{1,1} + 7) >> 4$
(1,2)	$f_{0,0} = (3p_{0,0} + p_{0,1} + 3p_{1,0} + p_{1,1} + 4) >> 3$
(1,3)	$f_{0,0} = (3p_{0,0} + p_{0,1} + 9p_{1,0} + 3p_{1,1} + 7) >> 4$
(2,0)	$f_{0,0} = (p_{0,0} + p_{0,1} + 1) >> 1$
(2,1)	$f_{0,0} = (3p_{0,0} + 3p_{0,1} + p_{1,0} + p_{1,1} + 4) >> 3$
(2,2)	$f_{0,0} = (p_{0,0} + p_{0,1} + p_{1,0} + p_{1,1} + 1) >> 2$
(2,3)	$f_{0,0} = (p_{0,0} + p_{0,1} + 3p_{1,0} + 3p_{1,1} + 4) >> 3$
(3,0)	$f_{0,0} = (p_{0,0} + 3p_{0,1} + 1) >> 2$
(3,1)	$f_{0,0} = (3p_{0,0} + 9p_{0,1} + p_{1,0} + 3p_{1,1} + 7) >> 4$
(3,2)	$f_{0,0} = (p_{0,0} + 3p_{0,1} + p_{1,0} + 3p_{1,1} + 4) >> 3$
(3,3)	$f_{0,0} = (p_{0,0} + p_{0,1} + p_{1,0} + p_{1,1} + 1) >> 2$

The final value of f is clipped to the range 0-255.

4.3.3 Block sizes for Inter prediction

In this model it is possible to estimate motion and compensate motion on 16x16, 16x8, 8x16 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. The luma frame data is padded by 16 on each side. Interpolation filter Taps Lengths of 6, 2, and 12 (This is special case in Interlace RV9 only) exist for RV9. A valid MV is defined such that the interpolation of that MV is possible within the padded image.

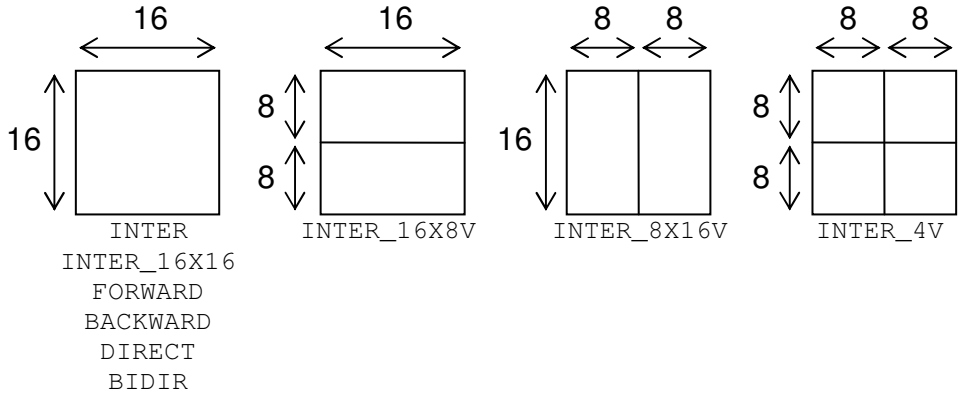


Figure 4.5: Motion compensation block sizes for Inter macroblocks

4.3.4 4x4 Intra Prediction

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. DC prediction (the average of the block above and to the left) mode is always allowed. Two modes use simple spatial prediction (1) column based from above, and (2) row based from the left. Additional prediction modes are diagonal.

In figure 4.6 below, a 4x4 block is to be predicted (pixels labeled a to p below). The pixels A to P and X from neighboring blocks and may already decoded and used for prediction.

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

Figure 4.6: Predicted and predictor pixels

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+4)>>3. If four of the pixels are outside the picture, the average of the remaining four is used for prediction – i.e. (A+B+C+D+2)>>2 or (I+J+K+L+2)>>2. 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 – 8:
These diagonal modes are used only if all A,B,C,D,I,J,K,L,X are inside the picture.

Mode 3:

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

Mode 4:

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

Mode 5:

a,j are predicted by	(X + A + 1) >> 1
b,k are predicted by	(A + B + 1) >> 1
c,l are predicted by	(B + C + 1) >> 1
d is predicted by	(C + D + 1) >> 1
e,n are predicted by	(I + 2X + A + 2) >> 2
f,o are predicted by	(X + 2A + B + 2) >> 2
g,p are predicted by	(A + 2B + C + 2) >> 2
h is predicted by	(B + 2C + D + 2) >> 2
i is predicted by	(X + 2I + J + 2) >> 2
m is predicted by	(I + 2J + K + 2) >> 2

Mode 6:

a is predicted by	(2A + 2B + J + 2K + L + 4) >> 3
b,i are predicted by	(B + C + 1) >> 1
c,j are predicted by	(C + D + 1) >> 1

```
d,k are predicted by            (D + E + 1) >> 1
l is predicted by               (E + F + 1) >> 1
e is predicted by               (A + 2B + C + K + 2L + M + 4) >> 3
f,m are predicted by            (B + 2C + D + 2) >> 2
g,n are predicted by            (C + 2D + E + 2) >> 2
h,o are predicted by            (D + 2E + F + 2) >> 2
p is predicted by               (E + 2F + G + 2) >> 2
```

Mode 7:

```
a is predicted by               (B + 2C + D + 2I + 2J + 4) >> 3
b is predicted by               (C + 2D + E + I + 2J + K + 4) >> 3
c,e are predicted by            (D + 2E + F + 2J + 2K + 4) >> 3
d,f are predicted by            (E + 2F + G + J + 2K + L + 4) >> 3
g,i are predicted by            (F + 2G + H + 2K +2L + 4) >> 3
h,j are predicted by            (G + 3H + K + 3L + 4) >> 3
l,n are predicted by            (L + 2M + N + 2) >> 2
m,k are predicted by            (G + H + L + M + 2) >> 2
o is predicted by               (M + N + 1) >> 1
p is predicted by               (M + 2N + O + 2) >> 2
```

Mode 8:

```
a,g are predicted by            (X + I + 1) >> 1
b,h are predicted by            (I + 2X + A + 2) >> 2
c is predicted by               (X + 2A + B + 2) >> 2
d is predicted by               (A + 2B + C + 2) >> 2
e,k are predicted by            (I + J + 1) >> 1
f,l are predicted by            (X + 2I + J + 2) >> 2
i,o are predicted by            (J + K + 1) >> 1
j,p are predicted by            (I + 2J + K + 2) >> 2
m is predicted by               (K + L + 1) >> 1
n is predicted by               (J + 2K + L + 2) >> 2
```

Spatial prediction in chroma is also done on a 4x4 block basis using the same prediction modes used for luma. No additional prediction mode information is transmitted in the bitstream for chroma prediction. Instead, the prediction modes for chroma are derived from the modes used for luma.

For each chroma 4x4 block there are 4 4x4 blocks for the corresponding location in luma. The prediction mode used for both chroma planes (U and V) is the prediction mode used for the upper left of these luma 4x4 blocks.

4.3.5 16x16 Intra Prediction

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

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

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

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

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

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

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:

$$H = \sum_{i=1}^{8} i \cdot (P(7+i,-1) - P(7-i,-1))$$

$$V = \sum_{j=1}^{8} j \cdot (P(-1,7+j) - P(-1,7-j))$$

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

And finally the actual prediction:

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

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.

4.3.6 4x4 Transform

4.3.6.1 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 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 x 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.

4.3.6.2 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.

4.3.7 Quantization

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 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) >> 16) + f) >> 5,
```

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 and INTER_16x16 macroblocks quantization is performed as

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

where f is 0x55555. Dequantization is as above except that the three lowest frequency coefficients are dequantized with a different quantization level. These special coefficients are shown in Figure 4.7.

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

Figure 4.7: Second transform coefficients with lowered QP (shaded)

Specifically, these coefficients are dequantized using a different QP value. This value is derived from the macroblock QP (used for the other coefficients) using the two tables below. The first table is used for Intra macroblocks, and the second table is used for Inter macroblocks.

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

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

Quantization is performed the same way for chroma as for luma, except the QP value used is derived from the QP used for luma using the tables below. The chroma DC coefficient (c_0) is given an even lower QP than the chroma AC coefficients (c_1–c_{15}).

```
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 for input, output or intermediates. For exact precisions see 4.3.7.1.

4.3.7.1 *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

4.3.8 Deblocking filter

For I, P and B Pictures an in-loop deblocking filter is used. (Note: since B Picture are never used as reference frames, deblocking is optional in the encoder & decoder)

4.3.8.1 I and P Picture In-loop deblocking

4.3.8.1.1 Introduction:

After the reconstruction of a entire picture a conditional filtering of this picture takes place, that effects the *boundaries* of the 4x4 block structure. RV9 deblocking filter is designed to provide PSNR improvement as well high visual quality. Thus there is no smoothing post-filter required for RV9.

Deblocking Filter:

The deblocking filter consists of 3 basic parts.

- Determination of Block Strength.
- Activity Measures
- Filters

Block Strength:

First, each 4x4 luma block in a reconstructed macroblock is assigned a filtering **Strength**, which has the following value:

4x4 block condition	**Strength**
Macro block is Intra-coded, or INTER_16x16 coded	2
4x4 Block is non INTRA, but has nonzero coefficients	1
The absolute difference between one of the motion vector components of the two adjacent 4x4 blocks (above and to the left) is at least one integer pixel (four ¼ pixels)*.	1
The adjacent (above and to the left) motion vectors refer to different reference frames (in B frames)	1 or 2
Else	0

* All blocks are checked. Eg. Four Motion vector. Even if the block and the adjacent block are inside the MB they have different MVs and thus are checked for motion vector difference.

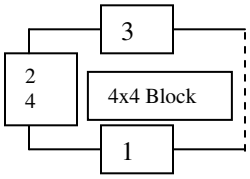
Filter Types:

There are 3 filter types. The Weak, Normal and Strong filter. The selection criterion is.

4x4 block condition	**Filter**
Strength is 2 on at least one side of the 4x4 edge and is MB edge.	Strong
Strength > 0 on at least one side of the 4x4 edge	Normal or Weak
Else	none

Filter Scan:

Filtering takes place on each 4x4 block in the following order.



1. Weak filter on the Horizontal Edge.
2. Weak filter on the Vertical Edge.
3. Strong Filter on Horizontal Edge.
4. Strong Filter on Vertical edge.

* Edges of a 4x4 block.

In essence the Vertical filter lags behind the Horizontal filter, and the Strong filters lag behind the Weak filters. The design is such that 4x4 blocks are traversed from left to right and top to bottom on the whole image. But there are other ways to achieve bit exact results. By carefully analysing the dependensies you could traverse MB's from left to right. Only picture edges are considered, there is No slice distinction.

Activity Measures:

After the strengths and filter types have been selected based on coded information, further selection of filters, recursive depth and strengths in done based on local image properties.

Block based Filter Decision: Parameters Al and Ar:

The type of filter (Strong, Normal or Weak) is made for each full edge of a 4x4 block by calculating a set of parameters (Al, Ar, b3SmoothLeft and b3SmoothRigh).

Al and Ar parameters select which filter to use and also the recursive depth of the filter, and are calculated using the following algorithm.

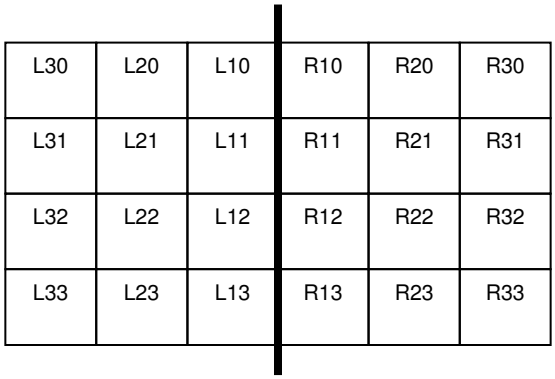


Figure 4.9: Filtering a vertical edge (dark line)

Consider the above vertical edge show in Fig. 4.9.

Compute:

```
deltaL[0] = L20 – L10;
deltaL[1] = L21 – L11;
deltaL[2] = L22 – L12;
deltaL[3] = L23 – L13;
deltaR[0] = R20 – R10;
deltaR[1] = R21 – R11;
deltaR[2] = R22 – R12;
deltaR[3] = R23 – R13;

Al = Ar = 3;

delta = deltaL[0]+deltaL[1]+deltaL[2]+deltaL[3];
```

```
if (ABS(delta) >= beta) Al = 1;
delta = deltaR[0]+deltaR[1]+deltaR[2]+deltaR[3];
if (ABS(delta) >= beta) Ar = 1;
```

NOTE: The DeltaR[] and DeltaL[] calculated can be used again later for the actually filter calculation.

```
deltaL2[0] = L20 – L30;
deltaL2[1] = L21 – L31;
deltaL2[2] = L22 – L32;
deltaL2[3] = L23 – L33;
deltaR2[0] = R20 – R30;
deltaR2[1] = R21 – R31;
deltaR2[2] = R22 – R32;
deltaR2[3] = R23 – R33;

b3SmoothLeft = b3SmoothRight = true;

delta = deltaL2[0]+deltaL2[1]+deltaL2[2]+deltaL2[3];
if (ABS(delta) >= beta2 && Al != 1) b3SmoothLeft = false;
delta = deltaR2[0]+deltaR2[1]+deltaR2[2]+deltaR2[3];
if (ABS(delta) >= beta2 && Ar != 1) b3SmoothRight = false;
```

Where **beta** is 4*β and **beta2** is either 3*β or 4*β. **beta2** is 4*β in the luma component when the number of pixels in the frame is less than or equal to 176*144. β is a function of the MB QP and given as

QP	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
β	0	0	0	0	0	0	0	0	3	3	3	4	4	4	6	6	6	7	8	8	9	9	10	10	11	11	12	13	14	15	16	17

Using Al, Ar, b3SmoothLeft and b3SmoothRight the table below shows the cases when each of the 3 filters are used.

Block Al Ar	**Filter**
Al = 3 && Ar = 3 && b3SmoothLeft=true && b3SmoothRight=true	Strong
Al > 1 && Ar > 1	Normal
Al or Ar >1	Weak
Else	none

Pixel Based Activity Measure:

Once a filter type has been selected for a 4x4 block edge, various clipping conditions are determined for each of the 4 rows or columns of the vertical and horizontal edge being filtered.

The alpha activity parameter is used to determine whether to keep the filtered pixel, clip it or to discard it -- all of which can be viewed as clipping functionality.

Consider the set of eight pixels across a 4x4 block horizontal or vertical boundary shown in Fig. 4.10.

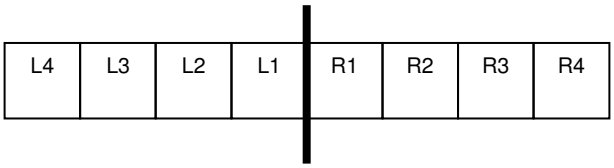


Figure 4.9: Eight pixels across a horizontal or vertical edge (dark line)

Each side of the edge has clip strength defined by **Cl** and **Cr** in the following way

```
Cl  = ClipTbl[QP][strength_left]
Cr  = ClipTbl[QP][strength_right]
```

with ClipTbl defined below

QP	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
ClipTbb(qp,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
ClipTbl(qp,1)	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	2	2	2	2	3	3	3	3	4	5	5
ClipTbl(qp,2)	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	2	2	2	2	3	3	3	4	4	5	5	5	7	8	9

```
            QP_Above
           ---- Top Edge -
QP_Left   |  QP_4x4_Block
```

The values of **strength_left** and **strength_right** defined as the 4x4 block **Strength** for the block to the left (or above) and the block to the right (or below) of the edge, respectively. The QP used to get these strengths are the QP's of the block to the left (QP_Left) or the block above (QP_Above). QP_4x4_Block is the QP of the blcok under consideration, thus is used for strength_right or strength_below, aplha, beta etc.

The below calculations of **delta** and **N** are used to determine the clipping bounds based on the type of filter.

```
delta = (R1 - L1);
N = ABS(delta)*alpha)>>7;
```

with **alpha** determined using the Macro Block QP using the table below.

QP	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
α	128	128	128	128	128	128	128	128	128	128	122	96	75	59	47	37	29	23	18	15	13	11	10	9	8	7	6	5	4	3	2	1

Filter Type Strong:

delta or N	**Clip**
delta = 0	C = 0 (effectively, no filtering)
N = 0	C = 255 (effectively, no clipping)
N = 1	C = (Cl + Cr + Al + Ar) / 2
N > 1	C = 0 (effectively, no filtering)

Filter Type Normal:

delta or N	**Clip**
delta = 0	C = 0 (effectively, no filtering)
N <= 2	C = (Cl + Cr + Al + Ar) / 2
N >= 3	C = 0 (effectively, no filtering)

Filter Type Weak:

delta or N	**Clip**
delta = 0	C = 0 (effectively, no filtering)
N <= 3	C = (Cl + Cr + Al + Ar) / 4, if **Strength** = 2 C = (Cl + Cr + Al + Ar) / 8, if **Strength** = 1
N >= 4	C = 0 (effectively, no filtering)

Pixel based Recursive Depth decision:

This too can be viewed as a clipping decision. You can filter all 4 pixel of the block by the same filter structure and decide to keep or discard the filtered pixel based on β.

Strong Filter:

3rd pel is not filtered for chroma.

Normal Filter:

ABS(deltaL2)<= β

Then filter 2nd pixel on the Left. Same for right.

Weak Filter:

ABS(deltaL2)<= β && Al>1

Then Filter the 2nd pixel on the Left. Same for right.

4.3.8.1.2 Filter Structures:

The following defines each of the three filters using the pixel notation from Fig. 4.9. With L1', L2', L3', R1', R2' and R3' being the resulting filtered output

Weak Filter

Δ = Clip(-C, C, ((R1 - L1 + 1) >> 1)

L1' = Clip(0, 255, (L1+Δ))

R1' = Clip(0, 255, (R1-Δ))

ΔL = Clip(-Cl/2, Cl/2, (L3 + L1 - L2<<1) >> 1)

L2' = Clip(0, 255, (L2+ΔL))

```
      ΔR    = Clip(-Cr/2, Cr/2, (R3 + R1 - R2<<1) >> 1)
      R2'   = Clip(0, 255, (R2+ΔR) )
```

Normal Filter

```
      Δ     = Clip(-C, C, ((R1 - L1) << 2 + (L2 - R2) + 4) >> 3)
      L1'   = Clip(0, 255, (L1+Δ) )
      R1'   = Clip(0, 255, (R1-Δ) )

      ΔL    = Clip(-Cl, Cl, (L3 + L1 - L2<<1 - Δ) >> 1)
      L2'   = Clip(0, 255, (L2+ΔL) )

      ΔR    = Clip(-Cr, Cr, (R3 + R1 - L2<<1 + Δ) >> 1)
      R2'   = Clip(0, 255, (R2+ΔR) )
```

Strong Filter

```
      L1''  = (25*L3 + 26*L2 + 26*L1 + 26*R1 + 25*R2 + D1) >> 7
      L1'   = Clip(-C, C, L1'' - L1) + L1;

      R1''  = (25*L2 + 26*L1 + 26*R1 + 26*R2 + 25*R3 + D2) >> 7
      R1'   = Clip(-C, C, R1'' - R1) + R1;

      L2''  = (25*L4 + 26*L3 + 26*L2 + 26*L1 + 25*R1 + D1) >> 7
      L2'   = Clip(-C, C, L2'' - L2) + L2;

      R2''  = (25*L1 + 26*R1 + 26*R2 + 26*R3 + 25*R4 + D2) >> 7
      R2'   = Clip(-C, C, R2'' - R2) + R2;

    If Luma
      L3'   = (        26*L4 + 51*L3 + 26*L2 + 25*L1 + 64) >> 7
      R3'   = (25*R1 + 26*R2 + 51*R3 + 26*R4         + 64) >> 7
```

Dither:

The Strong Filter output is dithered by adding variable offset D1 and D2 before down shifting and truncation. D1 and D2 value can be looked up by the relative position of the pixel in a 16x16 grid. The lookup index is simply the least significant nibble of the X or Y value. The appropriate Left or Right table is to be used.

```
ditherL[16]=
{ 64, 80, 32, 96, 48, 80, 64, 48, 80, 64, 80, 48, 96, 32, 80, 64 };

ditherR[16] =
{ 64, 48, 96, 32, 80, 48, 48, 64, 64, 64, 80, 48, 32, 96, 48, 64 };
```

Comments:

- The Basic Design of this filter is:
  - Block based decision of the Filter Structure
  - Pixel based clipping of the Filtered value.
- There might not be a need to clamp the output of certain filters to 0-255 since they always produce output within 0-255.
- Dither removes the Constant edge difference, which can be seen on high contrast displays even after filtering.

- Weak filter similar to Dither filters away the sharp edges of blocks, which would generally fail all activity tests for the Normal and Strong Filter.
- 5 tap filter has larger activity range with N<=1 thus clipping of the 5 tap output has been introduced for N=1.
- alpha, beta have been detuned above QP=23 (already reflected in the tables). beta2 is 4β for video equal or smaller than QCifs and 3β for all pictures larger than 176x144.

The peculiar Filter Scan allows for the output of the Strong 5 Tap filter with Dither to be the last operation on the image. This retains the Dither and 3rd filtered pel which otherwise would require special code to retain.

4.3.8.2 B Picture Deblocking Filter

4.3.8.2.1 Introduction:

Since RV9 deblocking filter is highly complex and B-Frames are not used for prediction, 2 Deblocking filters for B-frames are provided. Only under conditions when CPU is unable to handle Full Frame rate video should this simple filter be used. The RV9 In-loop filter described above with the modifications described below provide a high visual quality for B-frames.

4.3.8.2.2 In Loop Filter for B-frames

In B-frames certain blocks are filtered because they reference different reference frames. Using the scheme as described in section 4.3.8.1 these blocks are already tagged as to be filtered or not. Since this scheme will promote blocks to be filtered either by the strong or weak filter certain precautions have to be taken. The clipping strength of such a block is changed.

- Use Strong filter but use clipping strength corresponding to the reference frame.
- Use Normal filter but use the clipping strength of current frame QP and **Strength** = 2.

The clipping strengths on only the side corresponding to the block will be changed. The strength of the adjacent side is calculated based on Original strength of this block and the current QP.

4x4 block condition	**Filter Type**	**Clipping Strength**
The adjacent motion vectors refer to different reference frames. RefDiff == true		
Edge set to be Filtered	Strong	RefQP Strength 2
Edge not set to be Filtered	Normal	Qp Strength 2
Else	0	

MBtype	Adjacent MB type	RefDiff
Forward	Not Forward	True
Backward	Not Backward	True
Skipped Direct BiDir	Forward Backward	True
Intra	Any	False
Intra16x16	Any	False

4.3.8.2.3 Simple Filter

Use of Simple RV8 in-loop deblocking filter for B-frames (Luma only) is allowed for CPU scalability.

4.4 B Frames

RV9 supports the B frame mode with Forward, Backward, Direct, and Bi-predictive MB 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 five methods for motion compensating a macroblock - forward, backward, direct, Bi-predictive 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 can be interpolated 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 components are 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:

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

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

$U32\ w = ((v1 * uRatio0) >> 16) + ((v2 * uRatio1) >> 16);$

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

pf = pixel from future reference frame

pb = pixel from prev reference frame

pb = predicted direct mode pixel

(U32 is unsigned 32 bit integer. TR_SHIFT and TR_RND are constants required for the integer calulation of the ratios. Using any other scheme to get the ratio may not lead to bit exact reconstruction.)

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 R_F <=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 6-tap. (The filter length for Interlaced mode can be 12 taps see section 4.5.)

right edge: pos_x*3 + MVx < (width + 16-16-3)*3

left edge: pos_x*3 + MVx > -(16-2)*3

upper edge: pos_y*3 + MVy > -(16-2)*3

bottom edge: pos_y*3 + MVy < (height + 16-16-3)*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.

In Bi-directional mode 2 motion vectors are differentially coded and transmitted (see section 5.3.4.1), one forward and one backward. Two

predictions are interpolated based on these motion vectors. The final motion compensation is performed exactly like the Direct mode weighted average on these two predictions. iRatio0 and iRatio1 are set to 8192 each.

4.5 Interlaced Mode

<incomplete>

4.6 Reference Picture Resampling (RPR)

RV9 supports RPR in a manner identical to RV8/RVG2. (H263+ based) Reference picture resampling allows an encoder and decoder to change image dimensions on a frame-by-frame basis, without having to generate a key frame. When a new image dimension is received the decoder simply interpolates/decimates 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.

At the slice level the Picture size is transmitted using a Variable length and Fixed length scheme for I / P / B frames. (see section 5.2.2)
RPR is normative to Rv9 compatibility and this mode could be found in Media encoded by others on the net since it is ON for all releases of RV9 encoders. RPR is an encoder choice and be disabled if so required for closed loop implementations.

4.7 CPU Scalability

Based on experiments the following Decoder CPU scalability is allowed.

- Simpler In Loop Filter for B-Frames
- Disable De-Blocking in B-Frames.
- Snap to Integer Motion Vectors in B-frames.
- Dropping B-frames.

5 Bitstream Syntax

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

5.1 Stream Layer

For RealVideo the SPO (or codec opaque data) 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 Header.

Every stream is initialized with

- 32 bit SPO FLAG
- 32 bit Bitsream Version

5.1.1 SPO Flags

The set of 32 SPO Flags are used to indicate stream level options. Since RealVideo 9 has few optional stream-level modes, only a few SPO flags are useful.

TABLE 5.1: SPO Flags

Name	Mask	Description
RV40_SPO_FLAG_SLICEMODE	0x00000020	When equal to 1, indicates that slices are in use*.
RV40_SPO_FLAG_BFRAMES	0x00001000	When equal to 1, indicates that the stream may contain B-Frames*.
RV40_SPO_FLAG_FRUFLAG	0x00080000	When equal to 1, indicates that FRU should not be applied on this stream.
RV40_SPO_FLAG_MULTIPASS	0x00400000	When equal to 1, indicates the content was encoded with multipass**.
RV40_SPO_FLAG_VBR_ENCODE	0x01000000	When equal to 1, indicates the content was encoded using VBR**.

* For all RealVideo 9 streams this flag is set to 1.
** These flags are merely informational and do not affect the decoding process.

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 10 fields of every slice.

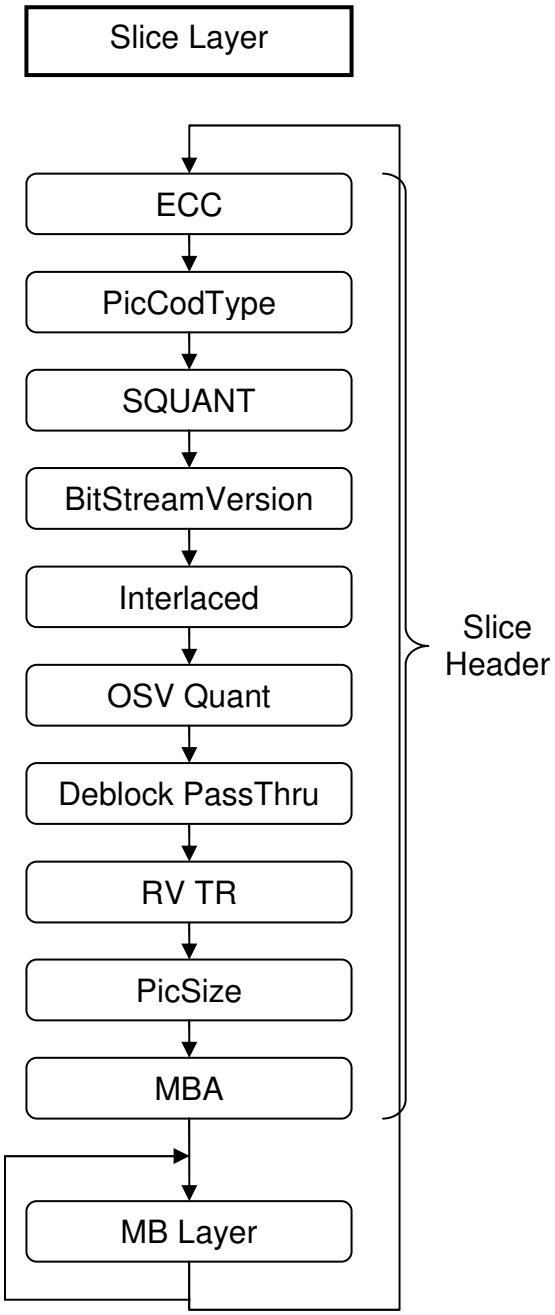


Figure 5.1: Slice Layer syntax

Each slice in the bitstream is corresponds to a independently decodable section. Thus, prediction across a slice boundary is not allowed. Motion vector prediction and intra mode prediction behaves as if the area outside the current slice is outside the picture.

TABLE 5.1: Slice Header field lengths

Field	Length	Description
ECC	1	0 if slice contains picture data 1 if slice contains ECC information
PicCodType	2	(00) = RV_INTRAPIC (01) = RV_FORCED_INTRAPIC (10) = RV_INTERPIC (11) = RV_TRUEBPIC
SQUANT	5	Initial Slice Quantization Parameter
Bitstream Version	1	Reserved – always zero.
Interlaced	1	Interlaced Slice Coding
OSV Quant	2	Super VLC Quantizer
Deblock PassThru	1	0 if deblocking filter is to be used 1 if deblocking filter is to be disabled
RV TR	13	Temporal reference (in units of millisecs)
PicSize	Var	Decoded Picture size.
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

5.2.1 ECC

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.

5.2.2 PicSize Syntax

PicSize(PicCodType) {	Bits
if(PicCodType == RV_INTERPIC || PicCodType == RV_TRUEBPIC) {	
use_prev_width	1
if(!use_prev_width) GetDimensions()	
} else {	
GetDimensions()	
}	
GetDimensions() {	
width_code	3
width = RPR_Width[width_code]	
if(width == 0) {	
width = explicit_dimension()	Var
}	
height_code	Var 3-4
height = RPR_Height[height_code]	
if(height == 0) {	
height = explicit_dimension()	Var
}	
PicWidth = width	
PicHeight = height	
}	

explicit_dimension() {	
Dimension = 0	
do {	
dim_code	8
Dimension = Dimension + dim_code * 4;	
} while(dim_code == 0xff)	
return Dimension	
}	

width_code	**Width**	**height_code**	**Height**
000	160	000	120
001	176	001	132
010	240	010	144
011	320	011	240
100	352	100	288
101	640	101	480
110	704	1100	180
111	0	1101	360
		1110	576
		1111	0

5.3 Macroblock Layer

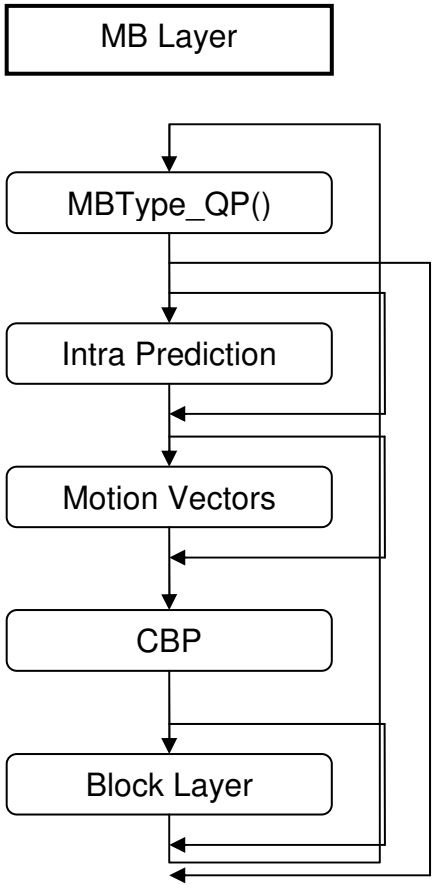


Figure 5.2: Macroblock Layer

5.3.1 Structured VLC code

RealVideo 9 uses a structured variable-length code table to code some the information in the video sequence. The structure of this VLC is shown in Table 5.2.

TABLE 5.2: Structured VLC

VLC Structure	Code number (N)	Explicit
1	0	1
0 x_0 1	1	0 0 1
	2	0 1 1
0 x_1 0 x_0 1	3	0 0 0 0 1
	4	0 0 0 1 1
	5	0 1 0 0 1
	6	0 1 0 1 1
0 x_2 0 x_1 0 x_0 1 …	7	0 0 0 0 0 0 1
	8	0 0 0 0 0 1 1
	…	…

In Table 5.2 x_n take values 0 or 1.

When code number 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.

The structured VLC is used in two places.

1. Representation of the number of consecutive SKIPPED macroblocks
2. Representation of motion vectors (in 1/4 pixel units)

5.3.2 MBType & DQuant

MBType_QP() {	Bits
if(PicCodType == RV_INTRAPIC) {	
IntraMBtype()	
} else {	
RLESkip_MB(SkipLeft)	
}	
}	

5.3.2.1 Intra Picture MB Type Syntax

IntraMBtype() {	Bits
mbtype_intra_16x16	1
if(!mbtype_intra_16x16) {	
mbtype_intra	1
if(!mbtype_intra) {	
dquant(PrevQP)	Var
mbtype_bit	1
}	
}	
}	

5.3.2.2 Run Length coding of Skipped MB

RLESkip_MB (SkipLeft) {	Bits
if(SkipLeft) {	
SkipLeft--	
} else {	
skip_run	Var
SkipLeft = VLC(skip_run)	
}	
if(SkipLeft) mbtype = Skipped	
else {	
AdaptiveMBType()	
}	
}	

5.3.2.3 Adaptive MB Type

The MB type for P and B-Frames are adaptively mapped to variable length codes as described below and signaled using a special VLC table.

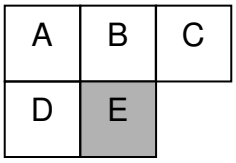


Figure 5.3: Macroblocks used to decode the macroblock type of macroblock E (shaded)

Decoding of the current macroblock type (macroblock E in Fig. 5.3) is based on neighboring macroblock types (macroblock s A, B, C, D).

The most probable mode near MB E is calculated by building a histogram of the MB types A, B, C, and D. If any is unavailable it is not considered for building the histogram. If a neighboring macroblock is a SKIPPED MB then in P-Frames it is considered a INTER MB and in B-Frames it is considered a Direct MB.

The MB Type with the largest frequency and the lowest Histogram Index (see Table 5.3) is the **mostProbableType.** Given the **mostProbableType** the appropriate VLC table is selected and used to read the MB type for the current macroblock.

TABLE 5.3: Macroblock types and Histogram Index

MB Types	Description	Pic Types	Histogram Index.
INTRA	Intra, 4x4 prediction	I/P/B	0
INTRA_16X16	Intra, 16x16 prediction, Dbl Xfm	I/P/B	1
INTER	Inter, 1MV, 16x16	P	2
INTER_4V	Inter, 4MV, 8x8	P	3
FORWARD	Fwd MV, 1MV, 16x16	B	4
BACKWARD	Bwd MV, 1MV, 16x16	B	5
DIRECT	Direct, Derived 2MV, 16x16	B	6
INTER_16X16	Inter, 1MV, 16x16, Dbl Xfrm	P	7
INTER_8X16V	Inter, 2MV, 8x16	P	8
BIDIR	Fwd & Bwd MV, 2MV, 16x16	B	9
INTER_16X8V	Inter, 2MV, 16x8	P	10

Tables 5.4 and 5.5 give the VLC codewords for P-Frames and B-Frames for each possible **mostProbableType.**

TABLE 5.4: VLC codes for MB Types in P-Frames

MB Type for macroblock E	VLC code for **mostProbableType**			
	Intra	Intra16x16	Inter	Inter_4V
INTRA	1	001	01101	1001
INTRA_16x16	01	1	0101	10001
INTER	001	01	1	01
INTER_4V	000001	0000001	0100	00
INTER_16X8V	00001	000001	001	101
INTER_8X16V	0001	00001	000	11
INTER_16x16	0000001	0001	0111	100001
DQUANT	0000000	0000000	01100	100000

MB Type for macroblock E	VLC code for **mostProbableType**		
	Inter16x8	Inter8x16	Inter16
INTRA	00001	00001	000001
INTRA_16x16	000001	000001	001
INTER	1	1	01
INTER_4V	0001	0001	0000001
INTER_16X8V	01	001	00001
INTER_8X16V	001	01	0001
INTER_16x16	0000001	0000001	1
DQUANT	0000000	0000000	0000000

TABLE 5.5: VLC codes for MB Types in B-Frames

MB Type for macroblock E	VLC code for **mostProbableType**		
	Intra	Intra16x16	Forward
INTRA	01	0001	000001
INTRA_16x16	101	1	0001
FORWARD	00	001	1
BACKWARD	11	01	01
BIDIR	10001	000001	00001
DIRECT	1001	00001	001
DQUANT	10000	000000	000000

MB Type for macroblock E	VLC code for **mostProbableType**		
	Backward	Bi-Direct	Direct
INTRA	01001	000001	000001
INTRA_16x16	001	00001	00001
FORWARD	000	001	001
BACKWARD	1	01	1
BIDIR	0101	0001	0001
DIRECT	011	1	01
DQUANT	01000	000000	000000

AdaptiveMBType () {	Bits
mb_code	Var
if(mb_code == DQuant) {	
mb_code	Var
dquant(PrevQP)	Var
}	
}	

5.3.2.4 DQuant

dquant(PrevQP) {	Bits
use_delta_QP	1
if(use_delta_QP) {	
delta_QP	1
dquant = gNewTAB_DQUANT_MQ[PrevQP][delta_QP];	
QP = PrevQP + dquant	
} else {	
QP	5
}	
}	

Dquant Table (gNewTAB_DQUANT_MQ)

PrevQP	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
δ=0	0	2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-2	-2	-2	-2	-2
δ=1	0	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2
PrevQP	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
δ=0	-2	-2	-2	-2	-2	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3
δ=1	2	2	2	2	2	3	3	3	3	3	3	3	3	2	1	-5

5.3.3 4x4 Intra Prediction Mode Coding

The signaling of the 4x4 intra prediction mode only occurs for INTRA 4x4 type macroblocks. A single VLC codeword can represent 1, 2 or 4

individual intra prediction modes. However, a single VLC codeword cannot represent intra prediction modes located on different rows of the macroblock. Therefore, in the bitstream a single row of intra prediction mode can be represented in the following combinations.

1. [4 Mode VLC]
2. [2 Mode VLC] [2 Mode VLC]
3. [2 Mode VLC] [1 Mode VLC] [1 Mode VLC]
4. [1 Mode VLC] [2 Mode VLC] [1 Mode VLC]
5. [1 Mode VLC] [1 Mode VLC] [2 Mode VLC]
6. [1 Mode VLC] [1 Mode VLC] [1 Mode VLC] [1 Mode VLC]

Four intra prediction modes are coded in one VLC codeword only when (a) it is the top row of a macroblock and (b) this macroblock is on the top edge of the image or current slice. This VLC table is listed as *aic_top_vlc[index]*.

Two intra prediction modes are coded in one VLC only if surrounding intra prediction modes are of a specific combination. In Fig. 5.4 below, modes *a* and *b* are being considered whether they are to be decoded as a single codeword.
In case of INTRA_16x16 the 4x4 block in consideration A, B, or C is given the mode number same as the INTRA_16x16 prediction mode (see Table5.7). In case of other MB mtypes, A, B, C are given mode number 0. When surrounding modes A, B and C are known, their combination is searched in Table 5.5. If the specific combination of A, B, and C are found, then a single VLC table, specified by **Table Number** is used to decode both a and b, together. These 20 VLC tables are listed as *aic_2mode_vlc[Table Numebr][index]*.

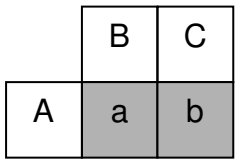


Figure 5.4: 4x4 Intra Prediction modes to be decoded (shaded) and The 4x4 prediction modes used to determine VLC table used

If A, B and C do not match the available patterns in Table 5.6, a single mode is decoded. VLC tables given in aic_1mode_vlc[A][B] are used to code this mode. The VLC table used in this case is listed as *aic_1mode_vlc[A][B]*.

The scan of coded intra prediction modes then moves left to the next one to be decoded. Once a row is finished, the scan proceeds to the next row of prediction modes.

TABLE 5.6: Intra prediction mode pattern for decoding two intra prediction modes as a single VLC codeword

Pattern			**Table Number**
A	**B**	**C**	
0	0	0	0
1	0	0	1
2	0	0	2
0	1	1	3
1	1	1	4
2	1	1	5
5	1	1	6
6	1	1	7
0	2	2	8
1	2	2	9
2	2	2	10
7	2	2	11
2	7	2	12
2	2	7	13
8	2	2	14
2	8	2	15
2	2	8	16
1	1	2	17
1	1	6	18
2	2	1	19

5.3.4 16x16 Intra Prediction Mode Coding

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 as shown in Table 5.7.

TABLE 5.7: VLC codewords used for the intra prediction mode of a Intra16 macroblock.

VLC Codeword	**Prediction Mode**
00	DC
01	Vertical, from above
10	Horizontal, from left
11	Planar

5.3.5 Motion Vectors

5.3.5.1 Prediction in P Frames

Motion vectors are differentially encoded from a predictor motion vector. 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 Fig. 5.5 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 and to the right of the current block, a candidate is instead found above and to the left, or just to the left if above and to the 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 can be interpolated from the padded picture. The reference pictures must be padded 16 pixels beyond the edges (eight for chroma planes) by replicating the edge pixels.

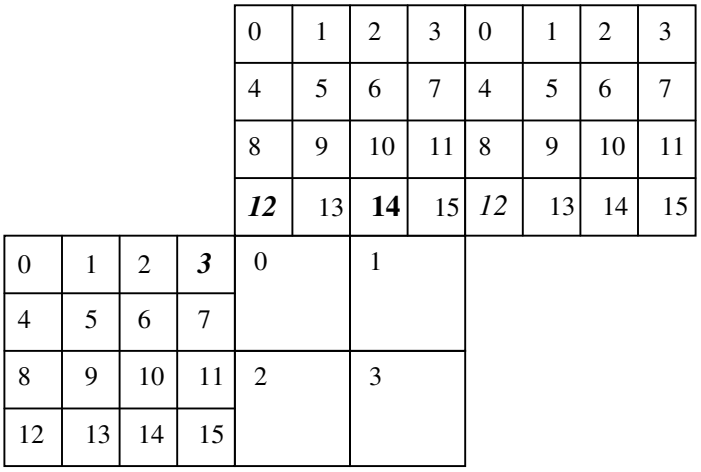


Figure 5.5: Motion vector prediction

5.3.5.2 Prediction in B frames

Motion vectors in B-Frames are only predicted from available motion vectors of neighboring macroblocks that use the same reference frame. The algorithm for determining *neighboring macroblocks* is the same as used in P-Frame MV prediction. Namely, If the Above-Right MV is unavailable due to slice or picture edges, the Above-Left MV is checked.

Therefore,

- The MV predictor for a Backward MB type can only consider
  - MVs from neighboring Backward MB types or
  - "backward" MVs from neighboring Bi-Direct MB types
- The MV predictor for a Forward MB type can only consider
  - MVs from neighboring Forward MB types or
  - "forward" MVs from neighboring Bi-Direct MB types
- The "backward" MV predictor for a Bi-Direct MB type can only consider
  - MVs from neighboring Backward MB types or
  - "backwards" MVs from neighboring Bi-Direct MB types
- The "forward" MV predictor for a B-Direct MB type can only consider
  - MVs from neighboring Forward MB types or

- o "forwards" MVs from neighboring Bi-Direct MB types

Depending on number of neighboring motion vectors which pass this criteria, a median, average or copy of the motion vectors from those macroblocks is used as the predictor.

TABLE 5.8: MV prediction in B-frames

Number of MVs	Prediction Type
3	Median
2	Average
1	Copy
0	0

5.3.5.3 Motion Vector Transmission

Depending on the MB type, 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, followed by 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 scanning, as shown in Fig. 5.6. See Table 5.9 for which code numbers to use.

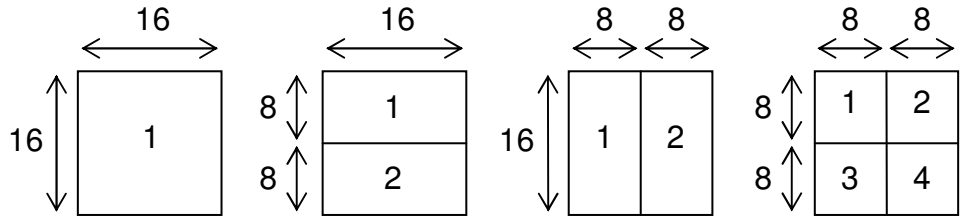


Figure 5.6: Motion vector transmission order

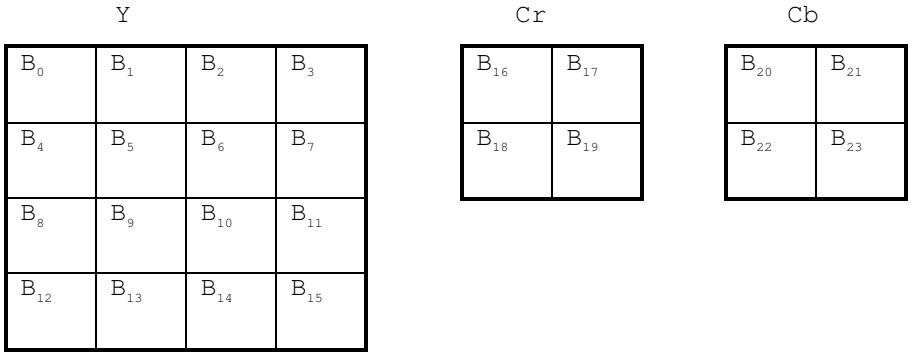
TABLE 5.9: Motion Vector codewords

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
:	:

5.3.6 CBP (Coded Block Pattern)

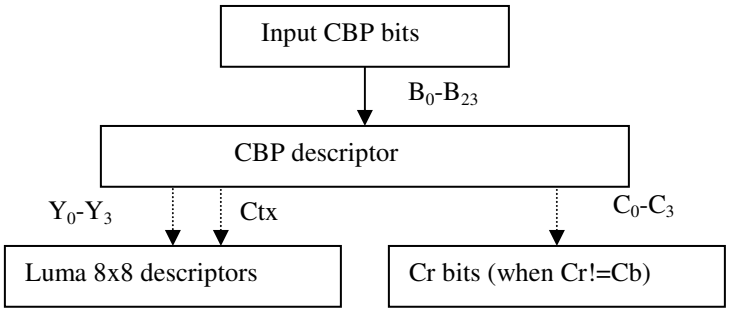
5.3.6.1 CBP length and bit order

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 code 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:

Y_0	Y_1
Y_2	Y_3

C_0	C_1
C_2	C_3

Composition rule:

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

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 descriptor 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}

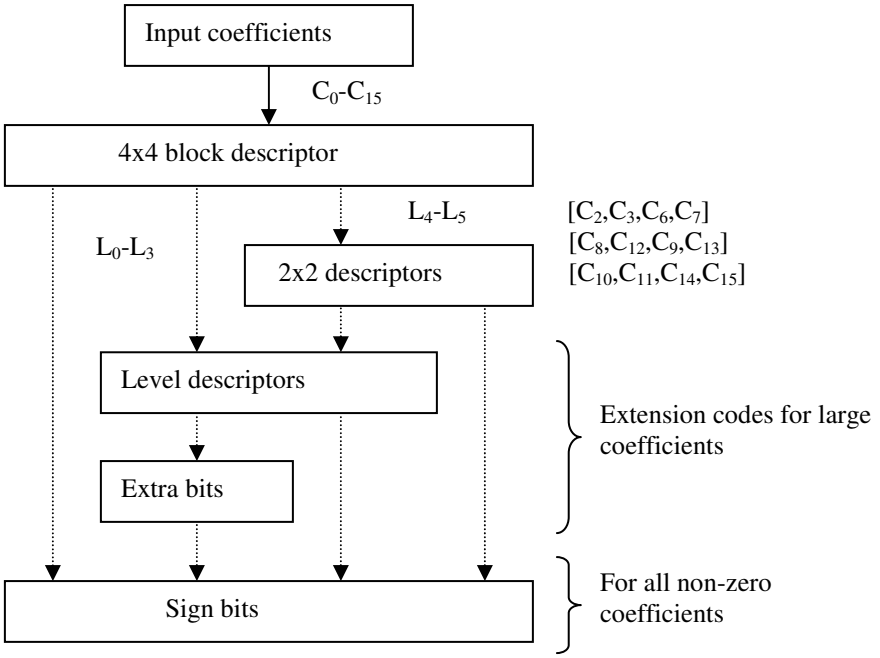
The following types of coefficients are encoded using separate groups of tables:

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_0 is always 0.

5.4.2 The structure of the code.

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



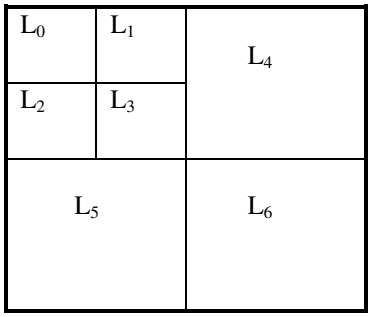
The main code object, 4x4 block descriptor is transmitted first. Based on the values of its components, subsequent code objects: 2x2 descriptors, level descriptors, and sign bits may follow.

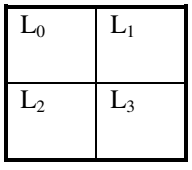
The order of these code objects follows the natural order of components in descriptors. E.g. if 4x4 descriptor indicates that there is a large DC coefficient, then, the next code object is its Level descriptor. If the level descriptor is not sufficient to represent the absolute value of this coefficient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

Below we describe each of these objects in details.

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	149-276
30	8	277-532

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 Performance Estimates

6.1 CPU Usage

The encoder/decoder complexity is asymmetric with the difference in complexity between the encoder and decoder near a factor of 3-5 times under normal (default) encoder and decoder operation.

The following systems are recommended for high quality playback of RealVideo 9 on PCs.

- **CIF or QCIF for dial-up connections –** For playback of typical content for dial-up speeds (176 x 132), a 200 MHz Pentium II (or better) is recommended.
- **Full Screen –** For playback of 640 x 480 video at full 24 fps (for film) or 30 fps (for video), a 750 MHz Pentium III (or better) is recommended.
- **HDTV –** For playback of HD-resolution content (e.g. 720p), a 2.6 GHz Pentium 4 (or better) is recommended.

Tables 6.1 – 6.3 show the CPU performance of RealVideo 9 running on popular device processors. In Tables 6.2 and 6.3, the performance is measured on actual devices running the RealVideo 9 decoder optimized for that platform.

TABLE 6.1: RealVideo 9 CPU performance, with zero-wait memory

Simulation using ADS 1.2 developer tools, with zero-wait memory					
Bitrate	Image Size	Frame Rate	Content Type	Processor	
				ARM925T	XScale
30 Kbps	QCIF	15 fps	Low Action	10.8 MHz	10.7 MHz
48 Kbps	QCIF	15 fps	Moderate Action	25.4 MHz	25.6 MHz
250 Kbps	SIF	25 fps	High Action	171.7 MHz	172.9 MHz
200 Kbps	CIF	30 fps	Moderate Action	174.9 MHz	179.4 MHz
600 Kbps	CIF	30 fps	Moderate Action	235.3 MHz	257.0 MHz

TABLE 6.2: RealVideo 9 decoder CPU usage on XScale

Machine = ARM XScale, PXA255, iPAQ H5550, 400 MHz				
Bitrate	Image Size	Frame Rate	Content Type	CPU Usage
30 Kbps	QCIF	15 fps	Low Action	22.9 MHz
48 Kbps	QCIF	15 fps	Moderate Action	39.2 MHz
250 Kbps	SIF	25 fps	High Action	231.3 MHz
200 Kbps	CIF	30 fps	Moderate Action	260.4 MHz
600 Kbps	CIF	30 fps	Moderate Action	350.2 MHz

TABLE 6.3: RealVideo 9 CPU usage on a 1.4 GHz, Pentium 4 processor

Machine = Pentium 4, 1.4 GHz				
Bitrate	Image Size	Frame Rate	Content Type	CPU Usage
48 Kbps	CIF	30 fps	Low Action	81.9 MHz
100 Kbps	CIF	30 fps	Moderate Action	150.3 MHz
300 Kbps	QVGA	30 fps	Mixed Action	158.3 MHz
800 Kbps	CIF	30 fps	Moderate Action	260.0 MHz
1.5 Mbps	CIF	30 fps	Moderate Action	307.9 MHz

6.2 Memory Usage

The memory usage of the RealVideo 9 decoder is dependent on the image size of the video being decoded. Table 6.4 presents the effective memory usage for several popular resolutions.

TABLE 6.4: RealVideo 9 memory usage

Image Size	Memory Usage		
	Code Size[1]	RAM[2]	All RAM[3]
QCIF	172 KB	147 KB	257 KB
CIF	172 KB	315 KB	868 KB

[1] *Code Size* reflects ARM-based Symbian/Series 60 platforms
[2] *RAM* usage does not include reference frame memory
[3] *All RAM* includes memory needed for reference and current frame buffers

7 QA Test Procedures

<Section to be expanded.>

8 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.

9 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;
```

10 Annex B

10.1Encoder 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.

10.2 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] (deprecated)
-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] (deprecated)
-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].