RealNetworks

RealVideo 8
RealVideo 9

Combo Decoder Summary

Video and Audio Technologies
Codec Group
RealNetworks, Inc

October 23, 2002

Version 1.0

Summary

This document a summary of the RealVideo 8 and RealVideo 9 Codec. RealVideo 8 and 9 achieves new levels of compression performance at low as well as high data rates. The improvements are due in part to 1/3 and 1/4 pixel interpolation for motion estimation, the addition of 16x16, 16x8, 8x16 and 8x8 pixel motion compensated blocks, 4x4 pixel block transforms, 16x16 "double" transforms, use of smart in-loop filters, efficient coding of 4x4 intra prediction modes, run length coding of MB-Types, and more efficient variable length coding of residual transform coefficients. RealVideo 9 doesn't need a post filter, while RealVideo 8 does take advantage of one. RealVideo 9 adds new Interlace capability to the Codec. The *RealVideo 8+9 Combo Decoder* is able to decode both RealVideo 8 and RealVideo 9 content. This decoder has built in CPU scalability to ensure best possible video experience various hardware configurations.

Revision History:

Revision	Date	Comment
1.0	10/23/02	Initial Decoder Summary

Table of Content

1 High Level Overview

RealVideo 8 and RealVideo 9 represents major advances in compression performance. The *RealVideo 8+9 Combo Decoder* is a singe decoder implementation able to decode both RealVideo 8 and RealVideo 9 content. RealVideo 8 and RealVideo 9 achieves new levels of compression performance at low as well as high data rates. The improvements are due in part to

- 4x4 pixel block transforms
- 16x16 "double" transforms
- 1/3 and 1/4 pixel interpolation for motion compensation
- the addition of 16x16, 16x8, 8x16 and 8x8 pixel motion compensated blocks
- 4x4 and 16x16 spatial prediction for intra-coded macroblocks
- smart in-loop deblocking filter
- run length coding of some macroblock types
- more efficient variable length coding of residual transform coefficients

RealVideo 9 doesn't need a post filter, while RealVideo 8 does take advantage of one. RealVideo 9 adds new Interlace capability to the Codec. The *RealVideo 8+9 Combo 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). HDTV resolutions (e.g. 1080i, 720p) are also supported, and there is no maximum supported frame size.

Video quality requirements: A noticeable improvement in video quality over standards-based codecs at comparable data rates.

3 Algorithm Descriptions

RealVideo 8 and RealVideo 9 consist of a set of coding tools including:

- 4x4 pixel block transforms
- 16x16 "double" transforms for intra-coded macroblocks
- sub-pixel accurate interpolation for motion compensation
- 16x16 and 8x8 pixel motion compensated blocks
- in-loop deblocking filter
- 4x4 and 16x16 spatial prediction for intra-coded macroblocks
- more efficient variable length coding of residual transform coefficients

In addition RealVideo 9 adds the following additional coding tools:

- Improved Intra mode coding
- Advanced Deblocking Filter
- 1/4 Pel Motion Estimation (includes the Funny position)
- Addition 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 coding
- Run length encoding of Skipped macroblocks
- Better B Frame motion vector prediction
- Bidirectional MB Type for B frames
- Interlaced Coding

3.1 Overview

RealVideo 8 and 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 3.1 provides a high-level block diagram of the algorithm.

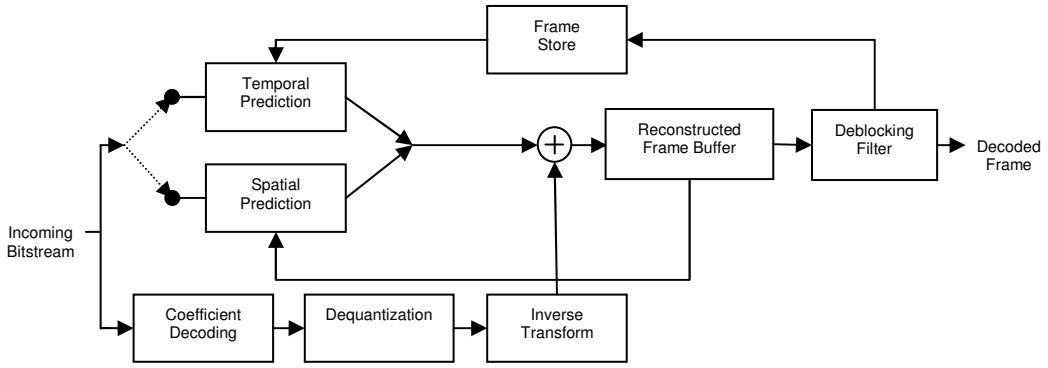

Figure 3.1: Block diagram of the RealVideo 8 and 9 decoder algorithm

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 8 and 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.

3.1.1 Picture Types

There are 3 picture types in RealVideo 8 and 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 3.2 provides an example of display and bitstream ordering of I, P and B Pictures.

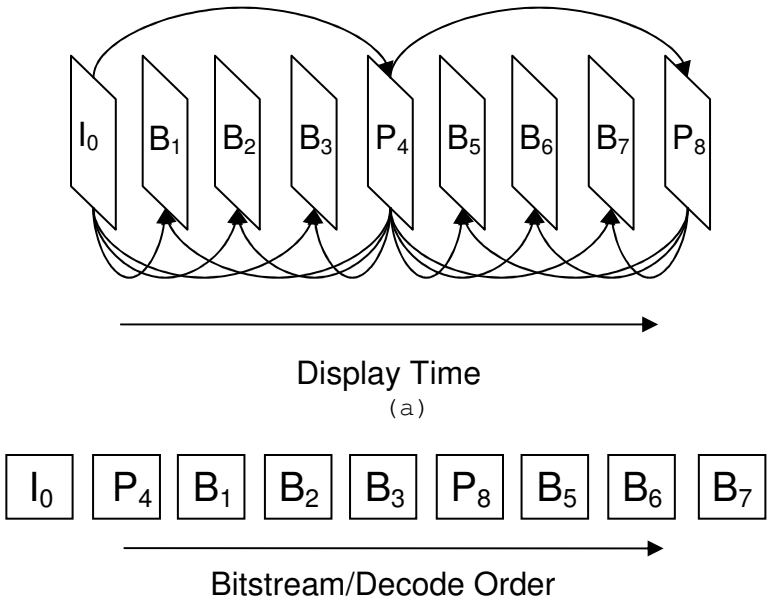

Figure 3.2: (a) Display Order. (b) Bitstream and Decode Order

3.1.2 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 3.3.

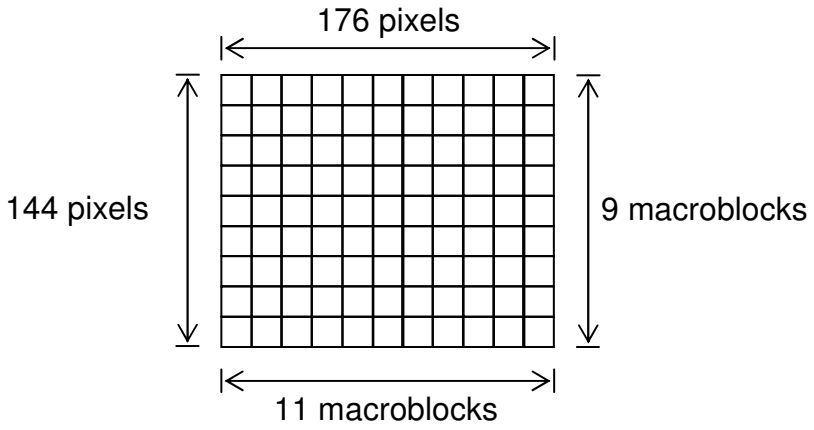

Figure 3.3: A picture with 11 x 9 macroblocks (QCIF picture)

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.

3.1.3 Macroblock Structure

The basic transform used for residual coding is a 4x4 2-D transform. Figure 3.4 below indicate how a macroblock is divided into 4x4 regions and the scanning order of these regions.

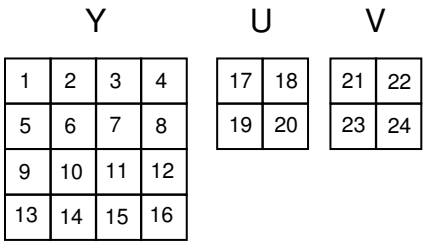

Figure 3.4: Macroblock scanning order of 4x4 blocks

3.2 Core Compression Algorithm

3.2.1 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 3.1.

TABLE 3.1: 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

3.2.2 Block sizes for Motion Compensated 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

exist for RV9. A valid MV is defined such that the interpolation of that MV is possible within the padded image.

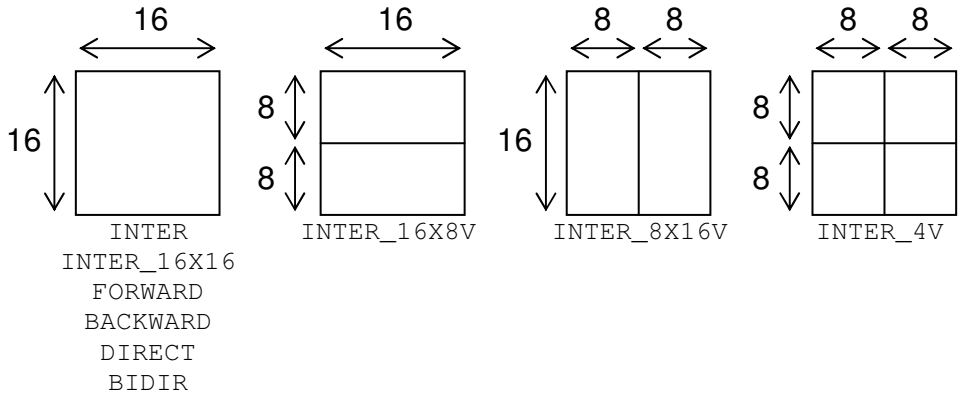

Figure 3.5: Motion compensation block sizes for Inter macroblocks

3.2.3 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 3.2.

TABLE 3.2: 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 3.3.

TABLE 3.3: 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.

3.2.4 1/4 sub-pel prediction

Motion vectors in RealVideo 9 are transmitted in 1/4 pixel units. 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).
```

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 3.4.

TABLE 3.4: 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)	$t_0 = p_0$ $v_0 = h_0$
(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)	$t_0 = p_0$ $v_0 = (t_{-2} - 5t_{-1} + 52t_0 + 52t_1 - 5t_2 + t_3 + 32) >> 6$
(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)	$t_0 = (p_{-2} - 5p_{-1} + 52p_0 + 52p_1 - 5p_2 + p_3 + 32) >> 6$ $v_0 = (t_{-2} - 5t_{-1} + 20t_0 + 52t_1 - 5t_2 + t_3 + 32) >> 6$
(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)	$t_0 = (p_{-2} - 5p_{-1} + 52p_0 + 20p_1 - 5p_2 + p_3 + 32) >> 6$ $v_0 = t_0$
(3,1)	$t_0 = (p_{-2} - 5p_{-1} + 52p_0 + 20p_1 - 5p_2 + p_3 + 32) >> 6$ $v_0 = (t_{-2} - 5t_{-1} + 52t_0 + 20t_1 - 5t_2 + t_3 + 32) >> 6$
(3,2)	$t_0 = (p_{-2} - 5p_{-1} + 52p_0 + 20p_1 - 5p_2 + p_3 + 32) >> 6$ $v_0 = (t_{-2} - 5t_{-1} + 20t_0 + 20t_1 - 5t_2 + t_3 + 16) >> 5$
(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 3.5.

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 3.5: 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.

3.2.5 4x4 Intra Prediction

Spatial prediction for intra-coded macroblocks is used. This prediction is 4x4 block based 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.

3.2.6 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.

3.2.7 4x4 Transform

3.2.7.1 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 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.

3.2.7.2 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.

3.2.8 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.

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.

3.2.9 RealVideo 8 Deblocking filter

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.

3.2.10 RealVideo 9 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)

After the reconstruction of a entire picture a conditional filtering of this picture takes place, that effects the *boundaries* of the 4x4 block structure. RealVideo 9 deblocking filter is designed to provide PSNR improvement as well high visual quality. Thus there is no smoothing post-filter required for RealVideo 9.

Since RealVideo 9 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.

3.3 B Frames

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.

3.4 Interlaced Mode

<incomplete>

3.5 Reference Picture Resampling (RPR)

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 with 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.

3.6 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.