FFV1 Video Codec Specification

by Michael Niedermayer <michaelni@gmx.at>

April 7, 2012

Contents

1 Introduction

The FFV1 video codec is a simple and efficient lossless intra only codec.

The latest version of this document is available at `https://raw.github.com/FFmpeg/FFV1/master/ffv1.lyx`

This document assumes familiarity with mathematical and coding concepts such as Range coding and YCbCr colorspaces.

2 Terms and Definitions

ESC	Escape symbol to indicate that the to be stored symbol is too large for normal storage and a different method is used to store it.
MSB	Most significant bit, the bit that can cause the largest change in magnitude of the symbol
RCT	Reversible component transform
VLC	Variable length code

3 High-level Description

Each frame is split in 1 to 4 planes (Y, Cb, Cr, Alpha). In the case of the normal YCbCr colorspace the Y plane is coded first followed by the Cb and Cr planes, if an Alpha/transparency plane exists, it is coded last. In the case of the JPEG2000-RCT colorspace the lines are interleaved to improve caching efficiency since it is most likely that the RCT will immediately be converted to RGB during decoding; the interleaved coding order is also Y,Cb,Cr.

Samples within a plane are coded in raster scan order (left->right, top->bottom). Each sample is predicted by the median predictor from samples in the same plane and the difference is stored see3.6.

3.1 Border

For the purpose of the predictior and context samples above the coded picture are assumed to be 0; samples to the right of the coded picture are identical to the closest left sample; samples to the left of the coded picture are identical to the top right sample (if there is one), otherwise 0.

0	0	0	0	0	0
0	0	0	0	0	0
0	0	a	b	c	c
0	a	d		e	e
0	d	f	g	h	h

3.2 Median predictor

median(left, top, left + top - diag)

left, top, diag are the left, top and lefttop samples

Note, this is also used in [JPEGLS, HUFFYUV]

3.3 Context

		T	
	tl	t	tr
L	l	X	

The quantized sample differences L-l, l-tl, tl-t, t-T, t-tr are used as context

$context = Q_0[l - tl] + |Q_0|\,(Q_1[tl - t] + |Q_1|\,(Q_2[t - tr] + |Q_2|\,(Q_3[L - l] + |Q_3|\,Q_4[T - t])))$

If the context is smaller than 0 then -context is used and the difference between the sample and its predicted value is encoded with a flipped sign

3.4 Quantization

There are 5 quantization tables for the 5 sample differences, both the number of quantization steps and their distribution are stored in the bitstream. Each quantization table has exactly 256 entries, and the 8 least significant bits of the sample difference are used as index

$Q_i[a - b] = Table_i[(a - b)\&255]$

3.5 Colorspace

3.5.1 JPEG2000-RCT

$Cb = b - g$

$Cr = r - g$

$Y = g + (Cb + Cr) >> 2$

$g = Y - (Cb + Cr) >> 2$

$r = Cr + g$

$b = Cb + g$

[JPEG2000]

3.6 Coding of the sample difference

Instead of coding the n+1 bits of the sample difference with huffman or range coding (or n+2 bits, in the case of RCT), only the n (or n+1) least significant bits are used, since this is sufficient to recover the original sample. In the equation below, bits represents bits_per_raw_sample+1 for RCT or bits_per_raw_sample otherwise.

$coder\_input = \left[\left(sample\_difference + 2^{bits-1}\right) \,\&\, \left(2^{bits} - 1\right)\right] - 2^{bits-1}$

3.6.1 Range coding mode

Early experimental versions of FFV1 used the CABAC Arithmetic coder from H.264[H264], but due to the uncertain patent/royality situation as well as its slightly worse performance CABAC was replaced by a range coder based on [RANGECODER].

Binary values To encode binary digits efficiently a range coder is used. $C_i$is the i-th Context. $B_i$is the i-th byte of the bytestream. $b_i$is the i-th range coded binary value, $S_{0,i}$is the i-th initial state, which is 128. The length of the bytestream encoding n binary symbols is $j_n$bytes.

$r_i = \left\lfloor \frac{R_i S_{i,C_i}}{2^8} \right\rfloor$

$$\begin{array}{rcccccccl} S_{i+1,C_i} = zero\_state_{S_{i,C_i}} & \wedge & l_i = L_i & \wedge & t_i = R_i - r_i & \Longleftarrow & b_i = 0 & \Longleftrightarrow & L_i < R_i - r_i \\ S_{i+1,C_i} = one\_state_{S_{i,C_i}} & \wedge & l_i = L_i - R_i + r_i & \wedge & t_i = r_i & \Longleftarrow & b_i = 1 & \Longleftrightarrow & L_i \geq R_i - r_i \end{array}$$

$S_{i+1,k} = S_{i,k} \quad \Longleftarrow \quad C_i \neq k$

$$\begin{array}{rcccccl} R_{i+1} = 2^8 t_i & \wedge & L_{i+1} = 2^8 l_i + B_{j_i} & \wedge & j_{i+1} = j_i + 1 & \Longleftarrow & t_i < 2^8 \\ R_{i+1} = t_i & \wedge & L_{i+1} = l_i & \wedge & j_{i+1} = j_i & \Longleftarrow & t_i \geq 2^8 \end{array}$$

$R_0 = 65280$

$L_0 = 2^8 B_0 + B_1$

$j_0 = 2$

Non binary values To encode scalar integers it would be possible to encode each bit separately and use the past bits as context, however that would mean 255 contexts per 8bit symbol which is not only a waste of memory but also requires more past data to reach a reasonably good estimate of the probabilities. Alternatively assuming a laplacian distribution and only dealing with its variance and mean (as in huffman coding) would also be possible, however, for maximum flexibility and simplicity, the chosen method uses a single symbol to encode if a number is 0 and if not encodes the number using its exponent, mantissa and sign. The exact contexts used are best described by the following code, followed by some comments.

```
void put_symbol(RangeCoder *c, uint8_t *state, int v, int is_signed) {
    int i;
    put_rac(c, state+0, !v);
    if (v) {
        int a= ABS(v);
        int e= log2(a);

        for (i=0; i<e; i++)
            put_rac(c, state+1+MIN(i,9), 1);  //1..10
        put_rac(c, state+1+MIN(i,9), 0);
        for (i=e-1; i>=0; i--)
            put_rac(c, state+22+MIN(i,9), (a>>i)&1); //22..31
        if (is_signed)
            put_rac(c, state+11 + MIN(e, 10), v < 0); //11..21
    }
}
```

Initial values for the context model At keyframes all range coder state variables are set to 128

State transition table
$one\_state_i = default\_state\_transition_i + state\_transition\_delta_i$
$zero\_state_i = 256 - one\_state_{256-i}$

default_state_transition

```
0,  0,  0,  0,  0,  0,  0,  0, 20, 21, 22, 23, 24, 25, 26, 27,
 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 37, 38, 39, 40, 41, 42,
 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 56, 57,
 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73,
 74, 75, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88,
 89, 90, 91, 92, 93, 94, 94, 95, 96, 97, 98, 99,100,101,102,103,
104,105,106,107,108,109,110,111,112,113,114,114,115,116,117,118,
119,120,121,122,123,124,125,126,127,128,129,130,131,132,133,133,
134,135,136,137,138,139,140,141,142,143,144,145,146,147,148,149,
150,151,152,152,153,154,155,156,157,158,159,160,161,162,163,164,
165,166,167,168,169,170,171,171,172,173,174,175,176,177,178,179,
180,181,182,183,184,185,186,187,188,189,190,190,191,192,194,194,
195,196,197,198,199,200,201,202,202,204,205,206,207,208,209,209,
210,211,212,213,215,215,216,217,218,219,220,220,222,223,224,225,
226,227,227,229,229,230,231,232,234,234,235,236,237,238,239,240,
241,242,243,244,245,246,247,248,248,  0,  0,  0,  0,  0,  0,  0,
```

alternative state transition table The alternative state transition table has been build using iterative minimization of frame sizes and generally performs better than the default. To use it, the coder_type has to be set to 2 and the difference to the default has to be stored in the header. The reference implemenation of FFV1 in FFmpeg uses this table by default at the time of this writing when range coding is used.

```
0, 10, 10, 10, 10, 16, 16, 16, 28, 16, 16, 29, 42, 49, 20, 49,
 59, 25, 26, 26, 27, 31, 33, 33, 33, 34, 34, 37, 67, 38, 39, 39,
 40, 40, 41, 79, 43, 44, 45, 45, 48, 48, 64, 50, 51, 52, 88, 52,
 53, 74, 55, 57, 58, 58, 74, 60,101, 61, 62, 84, 66, 66, 68, 69,
 87, 82, 71, 97, 73, 73, 82, 75,111, 77, 94, 78, 87, 81, 83, 97,
 85, 83, 94, 86, 99, 89, 90, 99,111, 92, 93,134, 95, 98,105, 98,
105,110,102,108,102,118,103,106,106,113,109,112,114,112,116,125,
115,116,117,117,126,119,125,121,121,123,145,124,126,131,127,129,
165,130,132,138,133,135,145,136,137,139,146,141,143,142,144,148,
147,155,151,149,151,150,152,157,153,154,156,168,158,162,161,160,
172,163,169,164,166,184,167,170,177,174,171,173,182,176,180,178,
175,189,179,181,186,183,192,185,200,187,191,188,190,197,193,196,
197,194,195,196,198,202,199,201,210,203,207,204,205,206,208,214,
209,211,221,212,213,215,224,216,217,218,219,220,222,228,223,225,
226,224,227,229,240,230,231,232,233,234,235,236,238,239,237,242,
241,243,242,244,245,246,247,248,249,250,251,252,252,253,254,255,
```

3.6.2 Huffman coding mode

Uses golomb rice codes. The VLC code is split in 2 parts, the prefix stores the most significant bits, the suffix stores the k least significant bits or stores the whole number in the ESC case. The end of the bitstream (of the frame) is filled with 0 bits so that the bitstream contains a multiple of 8 bits.

Prefix

bits	value
1	0
01	1
...	...
0000 0000 0001	11
0000 0000 0000	ESC

Suffix

non ESC the k least significant bits MSB first

ESC the value - 11, in MSB first order, ESC may only be used if the value cannot be coded as non ESC

Examples

k	bits	value
0	1	0
0	001	2
2	1 00	0
2	1 10	2
2	01 01	5
any	000000000000 10000000	139

Run mode Run mode is entered when the context is 0, and left as soon as a non 0 difference is found, the level is identical to the predicted one, the run and the first different level is coded

Run length coding

Level coding is identical to the normal difference coding with the exception that the 0 value is removed as it cant occur

```
if(diff>0) diff--;
encode(diff);
```

Note, this is different from JPEG-LS, which doesn't use prediction in run mode and uses a different encoding and context model for the last difference On a small set of test samples the use of prediction slightly improved the compression rate.

4 Bitstream

b Range Coded 1-bit symbol

v unsigned scalar symbol coded with the method described in 3.6.1

s signed scalar symbol coded with the method described in 3.6.1

The same context which is initialized to 128 is used for all fields in the header

4.1 Frame

Frame {	type
keyframe	b
if (keyframe)	
Header	
if (colorspace_type == 1) {	
for (y=0; y<height; y++) {	
LumaLine[y]	
CbLine[y]	
CrLine[y]	
if (alpha_plane)	
AlphaLine[y]	
}	
} else {	
LumaPlane	
if (chroma_planes) {	
CbPlane	
CrPlane	
}	
if (alpha_plane)	
AlphaPlane	
}	
}	

4.2 Header

Header {	type
version	v
coder_type	v
if(coder_type>1)	
for(i=1; i<256; i++)	
state_transition_delta[i]	s
colorspace_type	v
chroma_planes	b
if(version>0)	
bits_per_raw_sample	v
if(chroma_planes) {	
log2(h_chroma_subsample)	v
log2(v_chroma_subsample)	v
}	
alpha_plane	b
QuantizationTables	
}	

version	0 or 1
coder_type	Coder used, 0 (Golomb Rice), 1 (Range coder), 2 (Range coder with custom state transition table)
state_transition_delta	The range coder custom state transition table. If it is not coded, all its elements are assumed to be 0.
colorspace_type	0 (YCbCr), 1 (JPEG2000_RCT)
chroma_planes	1 for color, 0 for grayscale
bits_per_raw_sample	The number of bits for each sample, commonly 8, 9, 10 or 16
h_chroma_subsample	The subsample factor between luma and chroma width ($chroma\_width = 2^{-log2\_h\_chroma\_subsample} luma\_width$)
v_chroma_subsample	The subsample factor between luma and chroma height ($chroma\_height = 2^{-log2\_v\_chroma\_subsample} luma\_height$)
alpha_plane	1 if a transparency plane is stored, 0 otherwise

4.3 Quant Table

The quantization tables are stored by storing the number of equal entries -1 of the first half of the table using the method described in 3.6.1. The second half doesn't need to be stored as it is identical to the first with flipped sign

example:
Table: 0 0 1 1 1 1 2 2-2-2-2-1-1-1-1 0
Stored values: 1, 3, 1

5 Changelog

See `https://github.com/FFmpeg/FFV1/commits/master`

6 ToDo

- mean,k estimation for the golomb rice codes
- spelling errors

7 Bibliography

References

[JPEGLS] JPEG-LS FCD 14495 `http://www.jpeg.org/public/fcd14495p.pdf`

[H264] H.264 Draft `http://bs.hhi.de/~wiegand/JVT-G050.pdf`

[HUFFYUV] Huffyuv `http://cultact-server.novi.dk/kpo/huffyuv/huffyuv.html`

[FFMPEG] FFmpeg `http://ffmpeg.org`

[JPEG2000] JPEG2000 `http://www.jpeg.org/jpeg2000/`

[RANGECODER] "Range encoding: an algorithm for removing redundancy from a digitised message. G. N. N. Martin Presented in March 1979 to the Video & Data Recording Conference, IBM UK Scientific Center held in Southampton July 24-27 1979."

8 Copyright