

FFV1 Video Codec Specification

by Michael Niedermayer michaelni@gmx.at

- Introduction
- Terms and Definitions
- Terms
- Definitions
- Conventions
- Arithmetic operators
- Assignment operators
- Comparison operators
- Order of operation precedence
- Range
- Bitstream functions
- General Description
- Border
- Median predictor
- Context
- Quantization
- Colorspace
 - JPEG2000-RCT
- Coding of the sample difference
 - Range coding mode
 - Huffman coding mode
- Bitstream
- Configuration Record
 - In AVI File Format
 - In ISO/IEC 14496-12 (MP4 File Format)
 - In NUT File Format
- Frame
- Slice
- Slice Header
- Parameters
- Quantization Tables
 - Restrictions
- Changelog
- ToDo
- Bibliography
- References
- Copyright

1 Introduction

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

The latest version of this document is available at <https://raw.githubusercontent.com/FFmpeg/FFV1/master/ffv1.md>

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

2 Terms and Definitions

2.1 Terms

The key words MUST, MUST NOT, SHOULD, and SHOULD NOT in this document are to be interpreted as described in [RFC 2119](#).

For reference, below is an excerpt of RFC 2119:

“MUST”	means that the definition is an absolute requirement of the specification.
“MUST NOT”	means that the definition is an absolute prohibition of the specification.
“SHOULD”	means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
“SHOULD NOT”	means that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

2.2 Definitions

ESC	Escape symbol to indicate that the symbol to be stored 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 Color Transform, a near linear, exactly reversible integer transform that converts between RGB and YCbCr representations of a sample.
VLC	Variable length code.

3 Conventions

Note: the operators and the order of precedence are the same as used in the C programming language [ISO/IEC 9899](#).

3.1 Arithmetic operators

a + b means a plus b.

a - b means a minus b.

-a means negation of a.

a * b means a multiplied by b.

a / b means a divided by b with truncation of the result toward zero.

a % b means remainder of a divided by b.

a & b means bit-wise “and” of a and b.

a | b means bit-wise “or” of a and b.

a >> b means arithmetic right shift of two’s complement integer representation of a by b binary digits.

a << b means arithmetic left shift of two’s complement integer representation of a by b binary digits.

3.2 Assignment operators

a = b means a is assigned b.

a++ is equivalent to **a = a + 1**.

a-- is equivalent to **a = a - 1**.

a += b is equivalent to **a = a + b**.

a -= b is equivalent to **a = a - b**.

3.3 Comparison operators

a > b means a greater than b.

a >= b means a greater than or equal to b.

a < b means a less than b.

a <= b means a less than or equal b.

a == b means a equal to b.

a != b means a not equal to b.

a && b means boolean logical “and” of a and b.

a || b means boolean logical “or” of a and b.

!a means boolean logical “not”.

a ? b : c means b if a is true otherwise c.

3.4 Order of operation precedence

When order of precedence is not indicated explicitly by use of parentheses, operations are evaluated in the following order (from top to bottom, operations of same precedence being evaluated from left to right):

a++, a--
!a, -a
a * b, a / b, a % b
a + b, a - b
a << b, a >> b
a < b, a <= b, a > b, a >= b
a == b, a != b
a & b
a
a && b
a
a ? b : c
a = b, a += b, a -= b

3.5 Range

a...b means any value starting from a to b, inclusive.

3.6 Bitstream functions

remaing_bits_in_bitstream() means the count of remaining bits after the current position in the bitstream. It is computed from the NumBytes value multiplied by 8 minus the count of bits already read by the bitstream parser.

4 General 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,Alpha.

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 see [Coding of the Sample Difference](#).

4.1 Border

For the purpose of the predictor and context, samples above the coded slice are assumed to be 0; samples to the right of the coded slice are identical to the closest left sample; samples to the left of the coded slice 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

4.2 Median predictor

$\text{median}(\text{left}, \text{top}, \text{left} + \text{top} - \text{diag})$

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

Note, this is also used in [JPEG-LS and HuffYuv](#).

4.3 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.

4.4 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:

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

4.5 Colorspace

4.5.1 JPEG2000-RCT

$$Cb = b - g$$

$$Cr = r - g$$

$$Y = g + (Cb + Cr) \gg 2$$

$$g = Y - (Cb + Cr) \gg 2$$

$$r = Cr + g$$

$$b = Cb + g$$

JPEG2000

4.6 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, the term “bits” represents bits_per_raw_sample+1 for RCT or bits_per_raw_sample otherwise:

$$coder_input = [(sample_difference + 2^{bits-1}) \& (2^{bits} - 1)] - 2^{bits-1}$$

4.6.1 Range coding mode

Early experimental versions of FFV1 used the CABAC Arithmetic coder from H.264 but due to the uncertain patent/royalty situation, as well as its slightly worse performance, CABAC was replaced by a range coder based on an algorithm defined by *G. Nigel N. Martin* in 1979 RangeCoder.

4.6.1.1 Range 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.

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

$$S_{i+1,C_i} = zero_state_{S_{i,C_i}} \quad \wedge \quad l_i = L_i \quad \wedge \quad t_i = R_i - r_i \quad \Leftarrow \quad b_i = 0 \quad \Longleftrightarrow \quad L_i < R_i - r_i$$

$$S_{i+1,C_i} = one_state_{S_{i,C_i}} \quad \wedge \quad l_i = L_i - R_i + r_i \quad \wedge \quad t_i = r_i \quad \Leftarrow \quad b_i = 1 \quad \Longleftrightarrow \quad L_i \geq R_i - r_i$$

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

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

$$R_0 = 65280$$

$$L_0 = 2^8 B_0 + B_1$$

$$j_0 = 2$$

4.6.1.2 Range 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 8-bit 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
    }
}

```

4.6.1.3 Initial values for the context model At keyframes all range coder state variables are set to their initial state.

4.6.1.4 State transition table

$$\begin{aligned}
one_state_i &= default_state_transition_i + state_transition_delta_i \\
zero_state_i &= 256 - one_state_{256-i}
\end{aligned}$$

4.6.1.5 default_state_transition

0, 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, 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,

4.6.1.6 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 parameters. The reference implementation 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,

4.6.2 Huffman coding mode

This coding mode uses golomb rice codes. The VLC code is split into 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.

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

4.6.2.1 Prefix

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

4.6.2.2 Suffix

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

4.6.2.3 Examples

4.6.2.4 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.

4.6.2.5 Run length coding The run value is encoded in 2 parts, the prefix part stores the more significant part of the run as well as adjusting the `run_index` which determines the number of bits in the less significant part of the run. The 2nd part of the value stores the less significant part of the run as it is. The `run_index` is reset for each plane and slice to 0.

```
log2_run[41]={JPEGLS.
  0, 0, 0, 0, 1, 1, 1, 1,
  2, 2, 2, 2, 3, 3, 3, 3,
  4, 4, 5, 5, 6, 6, 7, 7,
  8, 9,10,11,12,13,14,15,
 16,17,18,19,20,21,22,23,
 24,
};

if (run_count == 0 && run_mode == 1) {
  if (get_bits1()) {
    run_count = 1 << log2_run[run_index];
    if (x + run_count <= w)
      run_index++;
  } else {
    if (log2_run[run_index])
      run_count = get_bits(log2_run[run_index]);
    else
      run_count = 0;
    if (run_index)
      run_index--;
    run_mode = 2;
  }
}
```

JPEGLS

4.6.2.6 Level coding Level coding is identical to the normal difference coding with the exception that the 0 value is removed as it cannot 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.

5 Bitstream

<code>u(n)</code>	unsigned big endian integer using n bits
<code>sg</code>	Golomb Rice coded signed scalar symbol coded with the method described in Huffman Coding Mode
<code>br</code>	Range coded boolean (1-bit) symbol with the method described in Range binary values
<code>ur</code>	Range coded unsigned scalar symbol coded with the method described in Range non binary values
<code>sr</code>	Range coded signed scalar symbol coded with the method described in Range non binary values

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

Default values at the decoder initialization phase:

ConfigurationRecordIsPresent is set to 0.

5.1 Configuration Record

In the case of a bitstream with version ≥ 2 , a configuration record is stored in the the underlying container, at the track header level. It contains the parameters used for all frames. The size of the configuration record, **NumBytes**, is supplied by the underlying container.

```
ConfigurationRecord( NumBytes ) {  
    ConfigurationRecordIsPresent = 1  
    Parameters( )  
    while( remaining_bits_in_bitstream( ) > 32)  
        reserved_for_future_use          u(1)  
    configuration_record_crc_parity      u(32)  
}
```

reserved_for_future_use has semantics that are reserved for future use.

Encoders conforming to this version of this specification SHALL NOT write this value.

Decoders conforming to this version of this specification SHALL ignore its value.

configuration_record_crc_parity 32 bits that are choosen so that the configuration record as a whole has a crc remainder of 0.

This is equivalent to storing the crc remainder in the 32-bit parity.

The CRC generator polynom used is the standard IEEE CRC polynom (0x104C11DB7) with initial value 0.

This configuration record can be placed in any file format supporting configuration records, fitting as much as possible with how the file format uses to store configuration records. The configuration record storage place and **NumBytes** are currently defined and supported by this version of this specification for the following container formats:

5.1.1 In AVI File Format

The Configuration Record extends the stream format chunk (“AVI “, “hdlr”, “strl”, “strf”) with the ConfigurationRecord bistream.

See [AVI](#) for more information about chunks.

NumBytes is defined as the size, in bytes, of the strf chunk indicated in the chunk header minus the size of the stream format structure.

5.1.2 In ISO/IEC 14496-12 (MP4 File Format)

The Configuration Record extends the sample description box (“moov”, “trak”, “mdia”, “minf”, “stbl”, “stds”) with a “glbl” box which contains the ConfigurationRecord bitstream.

See [ISO14496_12](#) for more information about boxes.

****NumBytes*** is defined as the size, in bytes, of the “glbl” box indicated in the box header minus the size of the box header.

5.1.3 In NUT File Format

The `codec_specific_data` element (in “stream_header” packet) contains the ConfigurationRecord bitstream. See **NUT** for more information about elements.

NumBytes is defined as the size, in bytes, of the `codec_specific_data` element as indicated in the “length” field of `codec_specific_data`

5.2 Frame

```

Frame( ) {
    keyframe
    if( keyframe && !ConfigurationRecordIsPresent)
        Parameters( )
    for( i = 0; i < slice_count; i++ )
        Slice( i )
}

```

5.3 Slice

```

Slice( i ) {
    if( version > 2 )
        SliceHeader( i )
    if( colorspace_type == 0 ) {
        for( p = 0; p < primary_color_count; p++ ) {
            Plane( p )
        }
    } else if( colorspace_type == 1 ) {
        for( y = 0; y < height; y++ )
            for( p = 0; p < primary_color_count; p++ ) {
                Line( p, y )
            }
    }
    if( i
        slice_size
    if( ec ) {
        error_status
        slice_crc_parity
    }
}

```

primary_color_count is defined as $1 + (\text{chroma_planes} ? 2 : 0) + (\text{alpha_plane} ? 1 : 0)$.

slice_size indicates the size of the slice in bytes.

Note: this allows finding the start of slices before previous slices have been fully decoded. And allows this way parallel decoding as well as error resilience.

error_status specifies the error status.

value	error status
0	no error
1	slice contains a correctable error
2	slice contains a uncorrectable error

value	error status
Other	reserved for future use

slice_crc_parity 32 bits that are choosen so that the slice as a whole has a crc remainder of 0.

This is equivalent to storing the crc remainder in the 32-bit parity.

The CRC generator polynom used is the standard IEEE CRC polynom (0x104C11DB7) with initial value 0.

5.4 Slice Header

SliceHeader(i) {	type
slice_x	ur
slice_y	ur
slice_width - 1	ur
slice_height - 1	ur
for(j = 0; j < quant_table_index_count; j++)	
quant_table_index [i][j]	ur
picture_structure	ur
sar_num	ur
sar_den	ur
if(version > 3) {	
reset_contexts	br
slice_coding_mode	ur
}	
}	

slice_x indicates the x position on the slice raster formed by num_h_slices.
Inferred to be 0 if not present.

slice_y indicates the y position on the slice raster formed by num_v_slices.
Inferred to be 0 if not present.

slice_width indicates the width on the slice raster.
Inferred to be 1 if not present.

slice_height indicates the height on the slice raster.
Inferred to be 1 if not present.

quant_table_index_count is defined as $1 + ((\text{chroma_planes} \parallel \text{version} < 4) ? 1 : 0) + (\text{alpha_plane} ? 1 : 0)$.

quant_table_index indicates the index to select the quantization table set and the initial states for the slice.

Inferred to be 0 if not present.

picture_structure specifies the picture structure.

Inferred to be 0 if not present.

value	picure structure used
0	unknown
1	top field first
2	bottom field first
3	progressive
Other	reserved for future use

sar_num specifies the sample aspect ratio numerator.

Inferred to be 0 if not present.

MUST be 0 if sample aspect ratio is unknown.

sar_den specifies the sample aspect ratio denominator.

Inferred to be 0 if not present.

MUST be 0 if sample aspect ratio is unknown.

reset_contexts indicates if slice contexts must be reset.

Inferred to be 0 if not present.

slice_coding_mode indicates the slice coding mode.

Inferred to be 0 if not present.

value	slice coding mode
0	normal Range Coding or VLC
1	raw PCM
Other	reserved for future use

5.5 Parameters

Parameters() {	type
version	ur
if(version > 2)	
micro_version	ur
coder_type	ur
if(coder_type > 1)	
for(i = 1; i < 256; i++)	
state_transition_delta[i]	sr
colorspace_type	ur
if(version > 0)	
bits_per_raw_sample	ur
chroma_planes	br
log2(h_chroma_subsample)	ur
log2(v_chroma_subsample)	ur
alpha_plane	br
if (version > 1) {	
num_h_slices - 1	ur
num_v_slices - 1	ur
quant_table_count	ur
}	
for(i = 0; i < quant_table_count; i++)	
QuantizationTable(i)	
if (version > 1) {	
for(i = 0; i < quant_table_count; i++) {	
states_coded	br
if(states_coded)	
for(j = 0; j < context_count[i]; j++)	
for(k = 0; k < CONTEXT_SIZE; k++)	
initial_state_delta[i][j][k]	sr
}	
ec	ur
intra	ur

Parameters() {	type
}	
}	

version specifies the version of the bitstream.

Each version is incompatible with others versions: decoders SHOULD reject a file due to unknown version.

Decoders SHOULD reject a file with version < 2 && ConfigurationRecordIsPresent == 1.

Decoders SHOULD reject a file with version >= 2 && ConfigurationRecordIsPresent == 0.

value	version
0	FFV1 version 0
1	FFV1 version 1
2	reserved*
3	FFV1 version 3
Other	reserved for future use

* Version 2 was never enabled in the encoder thus version 2 files SHOULD NOT exist, and this document does not describe them to keep the text simpler.

micro_version specifies the micro-version of the bitstream.

After a version is considered stable (a micro-version value is assigned to be the first stable variant of a specific version), each new micro-version after this first stable variant is compatible with the previous micro-version: decoders SHOULD NOT reject a file due to an unknown micro-version equal or above the micro-version considered as stable.

Meaning of micro_version for version 3:

value	micro_version
0...3	reserved*
4	first stable variant
Other	reserved for future use

* were development versions which may be incompatible with the stable variants.

Meaning of micro_version for version 4 (note: at the time of writting of this specification, version 4 is not considered stable so the first stable version value is to be annoned in the future):

value	micro_version
0...TBA	reserved*
TBA	first stable variant
Other	reserved for future use

* were development versions which may be incompatible with the stable variants.

coder_type specifies the coder used

value	coder used
0	Golomb Rice
1	Range Coder with default state transition table

value	coder used
2	Range Coder with custom state transition table
Other	reserved for future use

state_transition_delta specifies the range coder custom state transition table.

If **state_transition_delta** is not present in the bitstream, all range coder custom state transition table elements are assumed to be 0.

colourspace_type specifies the color space.

value	color space used
0	YCbCr
1	JPEG 2000 RCT
Other	reserved for future use

chroma_planes indicates if chroma (color) planes are present.

value	color space used
0	chroma planes are not present
1	chroma planes are present

bits_per_raw_sample indicates the number of bits for each luma and chroma sample. Inferred to be 8 if not present.

value	bits for each luma and chroma sample
0	reserved*
Other	the actual bits for each luma and chroma sample

* Encoders MUST not store **bits_per_raw_sample** = 0

Decoders SHOULD accept and interpret **bits_per_raw_sample** = 0 as 8.

h_chroma_subsample indicates the subsample factor between luma and chroma width ($chroma_width = 2^{-\log_2 h_chroma_subsample} luma_width$)

v_chroma_subsample indicates the subsample factor between luma and chroma height ($chroma_height = 2^{-\log_2 v_chroma_subsample} luma_height$)

alpha_plane indicates if a transparency plane is present.

value	color space used
0	transparency plane is not present
1	transparency plane is present

num_h_slices indicates the number of horizontal elements of the slice raster. Inferred to be 1 if not present.

num_v_slices indicates the number of vertical elements of the slice raster.

Inferred to be 1 if not present.

quant_table_count indicates the number of quantization table sets.

Inferred to be 1 if not present.

states_coded indicates if the respective quantization table set has the initial states coded.

Inferred to be 0 if not present.

value	initial states
0	initial states are not present and are assumed to be all 128
1	initial states are present

initial_state_delta [i][j][k] indicates the initial range coder state, it is encoded using k as context index and

pred = j ? initial_states[i][j - 1][k] : 128

initial_state[i][j][k] = (pred + initial_state_delta[i][j][k]) & 255

slice_count indicates the number of slices in the current frame, slice_count is 1 if it is not explicitly coded.

ec indicates the error detection/correction type.

value	error detection/correction type
0	32bit CRC on the global header
1	32bit CRC per slice and the global header
Other	reserved for future use

intra indicates the relationship between frames. Inferred to be 0 if not present.

value	relationship
0	frames are independent or dependent (key and non key frames)
1	frames are independent (key frames only)
Other	reserved for future use

5.6 Quantization Tables

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 **Range Non Binary Values**. 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

QuantizationTable(i) {	type
scale = 1	
for(j = 0; j < MAX_CONTEXT_INPUTS; j++) {	
QuantizationTablePerContext(i, j, scale)	
scale *= 2 * len_count[i][j] - 1	
}	
context_count[i] = (scale + 1) / 2	
}	

MAX_CONTEXT_INPUTS is 5.

QuantizationTablePerContext(i, j, scale) {	type
<pre> v = 0 for(k = 0; k < 128;) { len - 1 for(a = 0; a < len; a++) { quant_tables[i][j][k] = scale* v k++ } v++ } for(k = 1; k < 128; k++) { quant_tables[i][j][256 - k] = -quant_tables[i][j][k] } quant_tables[i][j][128] = -quant_tables[i][j][127] len_count[i][j] = v } </pre>	sr

quant_tables indicates the quantification table values.

context_count indicates the count of contexts.

5.6.1 Restrictions

In version 2 and later the maximum slice size in pixels is $\frac{width \cdot height}{4}$, this is to ensure that fast multithreaded decoding is possible.

6 Changelog

See <https://github.com/FFmpeg/FFV1/commits/master>

7 ToDo

- mean,k estimation for the golomb rice codes

8 Bibliography

8.1 References

RFC 2119 - Key words for use in RFCs to Indicate Requirement Levels <https://www.ietf.org/rfc/rfc2119.txt>

ISO/IEC 9899 - Programming languages - C <http://www.open-std.org/JTC1/SC22/WG14/www/standards>

JPEG-LS FCD 14495 <http://www.jpeg.org/public/fcd14495p.pdf>

H.264 Draft <http://bs.hhi.de/~wiegand/JVT-G050.pdf>

HuffYuv <http://cultact-server.novi.dk/kpo/huffyuv/huffyuv.html>

FFmpeg <http://ffmpeg.org>

JPEG2000 <http://www.jpeg.org/jpeg2000/>

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

AVI RIFF File Format <https://msdn.microsoft.com/en-us/library/windows/desktop/dd318189%28v=vs.85%29.aspx>

Information technology Coding of audio-visual objects Part 12: ISO base media file format http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=61988

NUT Open Container Format <http://www.ffmpeg.org/~michael/nut.txt>

9 Copyright

Copyright 2003-2013 Michael Niedermayer <michaelni@gmx.at>

This text can be used under the GNU Free Documentation License or GNU General Public License. See <http://www.gnu.org/licenses/fdl.txt>.