cellar M. Niedermayer
Internet-Draft D. Rice
Intended status: Standards Track J. Martinez
Expires: March 19, 2020 September 16, 2019

FFV1 Video Coding Format Version 4
draft-ietf-cellar-ffv1-v4-06

Abstract

This document defines FFV1, a lossless intra-frame video encoding format. FFV1 is designed to efficiently compress video data in a variety of pixel formats. Compared to uncompressed video, FFV1 offers storage compression, frame fixity, and self-description, which makes FFV1 useful as a preservation or intermediate video format.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document describes FFV1, a lossless video encoding format. The design of FFV1 considers the storage of image characteristics, data fixity, and the optimized use of encoding time and storage requirements. FFV1 is designed to support a wide range of lossless video applications such as long-term audiovisual preservation, scientific imaging, screen recording, and other video encoding scenarios that seek to avoid the generational loss of lossy video encodings.

This document defines a version 4 of FFV1. Prior versions of FFV1 are defined within [I-D.ietf-cellar-ffv1].

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

This document assumes familiarity with mathematical and coding concepts such as Range coding [range-coding] and YCbCr color spaces [YCbCr].

2. Notation and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2.1. Definitions

"Container": Format that encapsulates "Frames" (see the section on Frames (#frame)) and (when required) a "Configuration Record" into a bitstream.

"Sample": The smallest addressable representation of a color component or a luma component in a "Frame". Examples of "Sample" are Luma, Blue Chrominance, Red Chrominance, Transparency, Red, Green, and Blue.

"Plane": A discrete component of a static image comprised of "Samples" that represent a specific quantification of "Samples" of that image.

"Pixel": The smallest addressable representation of a color in a "Frame". It is composed of 1 or more "Samples".

"ESC": An ESCape symbol to indicate that the symbol to be stored is too large for normal storage and that an alternate storage method is used.

"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 "Pixel".

"VLC": Variable Length Code, a code that maps source symbols to a variable number of bits.

"RGB": A reference to the method of storing the value of a "Pixel" by using three numeric values that represent Red, Green, and Blue.

"YCbCr": A reference to the method of storing the value of a "Pixel" by using three numeric values that represent the luma of the "Pixel" (Y) and the chrominance of the "Pixel" (Cb and Cr). YCbCr word is used for historical reasons and currently references any color space relying on 1 luma "Sample" and 2 chrominance "Samples", e.g. YCbCr, YCgCo or ICtCp. The exact meaning of the three numeric values is unspecified.

"TBA": To Be Announced. Used in reference to the development of future iterations of the FFV1 specification.

2.2. Conventions

2.2.1. Pseudo-code

The FFV1 bitstream is described in this document using pseudo-code. Note that the pseudo-code is used for clarity in order to illustrate the structure of FFV1 and not intended to specify any particular implementation. The pseudo-code used is based upon the C programming language [ISO.9899.1990] and uses its "if/else", "while" and "for" functions as well as functions defined within this document.

2.2.2. Arithmetic Operators

Note: the operators and the order of precedence are the same as used in the C programming language [ISO.9899.1990].

"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.

"a ^ b" means a raised to the b-th power.

"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.

2.2.3. Assignment Operators

"a = b" means a is assigned b.

"a++" is equivalent to a is assigned a + 1.

"a--" is equivalent to a is assigned a - 1.

"a += b" is equivalent to a is assigned a + b.

"a -= b" is equivalent to a is assigned a - b.

"a *= b" is equivalent to a is assigned a * b.

2.2.4. Comparison Operators

"a > b" means a is greater than b.

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

"a < b" means a is less than b.

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

"a == b" means a is equal to b.

"a != b" means a is 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" of a.

"a ? b : c" if a is true, then b, otherwise c.

2.2.5. Mathematical Functions

floor(a) the largest integer less than or equal to a

ceil(a) the smallest integer greater than or equal to a

sign(a) extracts the sign of a number, i.e. if a < 0 then -1, else if a > 0 then 1, else 0

abs(a) the absolute value of a, i.e. abs(a) = sign(a)*a

log2(a) the base-two logarithm of a

min(a,b) the smallest of two values a and b

max(a,b) the largest of two values a and b

median(a,b,c) the numerical middle value in a data set of a, b, and c, i.e. a+b+c-min(a,b,c)-max(a,b,c)

a_(b) the b-th value of a sequence of a

a~b,c. the 'b,c'-th value of a sequence of a

2.2.6. 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). This order of operations is based on the order of operations used in Standard C.

```
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 & b
a | b
a && b
a || b
a ? b : c
a = b, a += b, a -= b, a *= b
```

2.2.7. Range

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

2.2.8. NumBytes

"NumBytes" is a non-negative integer that expresses the size in 8-bit octets of a particular FFV1 "Configuration Record" or "Frame". FFV1 relies on its "Container" to store the "NumBytes" values, see the section on the Mapping FFV1 into Containers (#mapping-ffv1-into-containers).

2.2.9. Bitstream Functions

2.2.9.1. remaining_bits_in_bitstream

"remaining_bits_in_bitstream( )" means the count of remaining bits after the pointer in that "Configuration Record" or "Frame". It is computed from the "NumBytes" value multiplied by 8 minus the count of bits of that "Configuration Record" or "Frame" already read by the bitstream parser.

2.2.9.2. remaining_symbols_in_syntax

"remaining_symbols_in_syntax( )" is true as long as the RangeCoder has not consumed all the given input bytes.

2.2.9.3. byte_aligned

"byte_aligned( )" is true if "remaining_bits_in_bitstream(NumBytes)" is a multiple of 8, otherwise false.

2.2.9.4. get_bits

"get_bits(i)" is the action to read the next "i" bits in the bitstream, from most significant bit to least significant bit, and to return the corresponding value. The pointer is increased by "i".

3. Sample Coding

For each "Slice" (as described in the section on Slices (#slice)) of a "Frame", the "Planes", "Lines", and "Samples" are coded in an order determined by the "Color Space" (see the section on Color Space (#color-spaces)). Each "Sample" is predicted by the median predictor as described in the section of the Median Predictor (#median-predictor) from other "Samples" within the same "Plane" and the difference is stored using the method described in Coding of the Sample Difference (#coding-of-the-sample-difference).

3.1. Border

A border is assumed for each coded "Slice" for the purpose of the median predictor and context according to the following rules:

* one column of "Samples" to the left of the coded slice is assumed as identical to the "Samples" of the leftmost column of the coded slice shifted down by one row. The value of the topmost "Sample" of the column of "Samples" to the left of the coded slice is assumed to be "0"

* one column of "Samples" to the right of the coded slice is assumed as identical to the "Samples" of the rightmost column of the coded slice

* an additional column of "Samples" to the left of the coded slice and two rows of "Samples" above the coded slice are assumed to be "0"

The following table depicts a slice of 9 "Samples" "a,b,c,d,e,f,g,h,i" in a 3x3 arrangement along with its assumed border.

```
+---+---+---+---+---+---+---+---+
| 0 | 0 |   | 0 | 0 | 0 |   | 0 |
+---+---+---+---+---+---+---+---+
| 0 | 0 |   | 0 | 0 | 0 |   | 0 |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
| 0 | 0 |   | a | b | c |   | c |
+---+---+---+---+---+---+---+---+
| 0 | a |   | d | e | f |   | f |
+---+---+---+---+---+---+---+---+
| 0 | d |   | g | h | i |   | i |
+---+---+---+---+---+---+---+---+
```

3.2. Samples

Relative to any "Sample" "X", six other relatively positioned "Samples" from the coded "Samples" and presumed border are identified according to the labels used in the following diagram. The labels for these relatively positioned "Samples" are used within the median predictor and context.

```
+---+---+---+---+
|   |   | T |   |
+---+---+---+---+
|   |tl | t |tr |
+---+---+---+---+
| L | l | X |   |
+---+---+---+---+
```

The labels for these relative "Samples" are made of the first letters of the words Top, Left and Right.

3.3. Median Predictor

The prediction for any "Sample" value at position "X" may be computed based upon the relative neighboring values of "l", "t", and "tl" via this equation:

"median(l, t, l + t - tl)".

Note, this prediction template is also used in [ISO.14495-1.1999] and [HuffYUV].

Exception for the median predictor: if "colorspace_type == 0 && bits_per_raw_sample == 16 && (coder_type == 1 || coder_type == 2)", the following median predictor MUST be used:

"median(left16s, top16s, left16s + top16s - diag16s)"

where:

```
left16s = l  >= 32768 ? ( l  - 65536 ) : l
top16s  = t  >= 32768 ? ( t  - 65536 ) : t
diag16s = tl >= 32768 ? ( tl - 65536 ) : tl
```

Background: a two's complement signed 16-bit signed integer was used for storing "Sample" values in all known implementations of FFV1 bitstream. So in some circumstances, the most significant bit was wrongly interpreted (used as a sign bit instead of the 16th bit of an unsigned integer). Note that when the issue is discovered, the only configuration of all known implementations being impacted is 16-bit YCbCr with no Pixel transformation with Range Coder coder, as other potentially impacted configurations (e.g. 15/16-bit JPEG2000-RCT with Range Coder coder, or 16-bit content with Golomb Rice coder) were implemented nowhere [ISO.15444-1.2016]. In the meanwhile, 16-bit JPEG2000-RCT with Range Coder coder was implemented without this issue in one implementation and validated by one conformance checker. It is expected (to be confirmed) to remove this exception for the median predictor in the next version of the FFV1 bitstream.

3.4. Context

Relative to any "Sample" "X", the Quantized Sample Differences "L-l", "l-tl", "tl-t", "T-t", and "t-tr" are used as context:

```
context = Q_{0}[l - tl] +
          Q_{1}[tl - t] +
          Q_{2}[t - tr] +
          Q_{3}[L - l]  +
          Q_{4}[T - t]
```

Figure 1

If "context >= 0" then "context" is used and the difference between the "Sample" and its predicted value is encoded as is, else "-context" is used and the difference between the "Sample" and its predicted value is encoded with a flipped sign.

3.5. Quantization Table Sets

The FFV1 bitstream contains 1 or more Quantization Table Sets. Each Quantization Table Set contains exactly 5 Quantization Tables with each Quantization Table corresponding to 1 of the 5 Quantized Sample Differences. For each Quantization Table, both the number of quantization steps and their distribution are stored in the FFV1 bitstream; each Quantization Table has exactly 256 entries, and the 8 least significant bits of the Quantized Sample Difference are used as index:

```
Q_{j}[k] = quant_tables[i][j][k&255]
```

Figure 2

In this formula, "i" is the Quantization Table Set index, "j" is the Quantized Table index, "k" the Quantized Sample Difference.

3.6. Quantization Table Set Indexes

For each "Plane" of each slice, a Quantization Table Set is selected from an index:

* For Y "Plane", "quant_table_set_index[0]" index is used

* For Cb and Cr "Planes", "quant_table_set_index[1]" index is used

* For extra "Plane", "quant_table_set_index[(version <= 3 || chroma_planes) ? 2 : 1]" index is used

Background: in first implementations of FFV1 bitstream, the index for Cb and Cr "Planes" was stored even if it is not used (chroma_planes set to 0), this index is kept for version <= 3 in order to keep compatibility with FFV1 bitstreams in the wild.

3.7. Color spaces

FFV1 supports several color spaces. The count of allowed coded planes and the meaning of the extra "Plane" are determined by the selected color space.

The FFV1 bitstream interleaves data in an order determined by the color space. In YCbCr for each "Plane", each "Line" is coded from top to bottom and for each "Line", each "Sample" is coded from left to right. In JPEG2000-RCT for each "Line" from top to bottom, each "Plane" is coded and for each "Plane", each "Sample" is encoded from left to right.

3.7.1. YCbCr

This color space allows 1 to 4 "Planes".

The Cb and Cr "Planes" are optional, but if used then MUST be used together. Omitting the Cb and Cr "Planes" codes the frames in grayscale without color data.

An optional transparency "Plane" can be used to code transparency data.

An FFV1 "Frame" using YCbCr MUST use one of the following arrangements:

* Y

* Y, Transparency

* Y, Cb, Cr

* Y, Cb, Cr, Transparency

The Y "Plane" MUST be coded first. If the Cb and Cr "Planes" are used then they MUST be coded after the Y "Plane". If a transparency "Plane" is used, then it MUST be coded last.

3.7.2. RGB

This color space allows 3 or 4 "Planes".

An optional transparency "Plane" can be used to code transparency data.

JPEG2000-RCT is a Reversible Color Transform that codes RGB (red, green, blue) "Planes" losslessly in a modified YCbCr color space [ISO.15444-1.2016]. Reversible Pixel transformations between YCbCr and RGB use the following formulae.

```
Cb=b-g
Cr=r-g
Y=g+(Cb+Cr)>>2
g=Y-(Cb+Cr)>>2
r=Cr+g
b=Cb+g
```

Figure 3

Exception for the JPEG2000-RCT conversion: if bits_per_raw_sample is between 9 and 15 inclusive and extra_plane is 0, the following formulae for reversible conversions between YCbCr and RGB MUST be used instead of the ones above:

```
Cb=g-b
Cr=r-b
Y=b+(Cb+Cr)>>2
b=Y-(Cb+Cr)>>2
r=Cr+b
g=Cb+b
```

Figure 4

Background: At the time of this writing, in all known implementations of FFV1 bitstream, when bits_per_raw_sample was between 9 and 15 inclusive and extra_plane is 0, GBR "Planes" were used as BGR "Planes" during both encoding and decoding. In the meanwhile, 16-bit JPEG2000-RCT was implemented without this issue in one implementation and validated by one conformance checker. Methods to address this exception for the transform are under consideration for the next version of the FFV1 bitstream.

When FFV1 uses the JPEG2000-RCT, the horizontal "Lines" are interleaved to improve caching efficiency since it is most likely that the JPEG2000-RCT will immediately be converted to RGB during decoding. The interleaved coding order is also Y, then Cb, then Cr, and then if used transparency.

As an example, a "Frame" that is two "Pixels" wide and two "Pixels" high, could be comprised of the following structure:

```
+------------------------+------------------------+
| Pixel(1,1)             | Pixel(2,1)             |
| Y(1,1) Cb(1,1) Cr(1,1) | Y(2,1) Cb(2,1) Cr(2,1) |
+------------------------+------------------------+
| Pixel(1,2)             | Pixel(2,2)             |
| Y(1,2) Cb(1,2) Cr(1,2) | Y(2,2) Cb(2,2) Cr(2,2) |
+------------------------+------------------------+
```

In JPEG2000-RCT, the coding order would be left to right and then top to bottom, with values interleaved by "Lines" and stored in this order:

Y(1,1) Y(2,1) Cb(1,1) Cb(2,1) Cr(1,1) Cr(2,1) Y(1,2) Y(2,2) Cb(1,2) Cb(2,2) Cr(1,2) Cr(2,2)

3.8. 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 JPEG2000-RCT), only the n (or n+1, in the case of JPEG2000-RCT) 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 JPEG2000-RCT or bits_per_raw_sample otherwise:

```
coder_input =
    [(sample_difference + 2^(bits-1)) & (2^bits - 1)] - 2^(bits-1)
```

Figure 5

3.8.1. Range Coding Mode

Early experimental versions of FFV1 used the CABAC Arithmetic coder from H.264 as defined in [ISO.14496-10.2014] 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 and N. Martin in 1979 [range-coding].

3.8.1.1. 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. The length of the bytestream encoding n binary symbols is "j~n~" bytes.

```
r_{i} = floor( ( R_{i} * S_{i,C_{i}} ) / 2^8 )
```

Figure 6

```
S_{i+1,C_{i}} =  zero_state_{S_{i,C_{i}}} XOR
          l_i =  L_i                      XOR
          t_i =  R_i - r_i                <==
          b_i =  0                        <==>
          L_i <  R_i - r_i

S_{i+1,C_{i}} =  one_state_{S_{i,C_{i}}}  XOR
          l_i =  L_i - R_i + r_i          XOR
          t_i =  r_i                      <==
          b_i =  1                        <==>
          L_i >= R_i - r_i
```

Figure 7

```
S_{i+1,k} = S_{i,k} <== C_i != k
```

Figure 8

```
R_{i+1} =  2^8 * t_{i}                    XOR
L_{i+1} =  2^8 * l_{i} + B_{j_{i}}        XOR
j_{i+1} =  j_{i} + 1                      <==
t_{i}   <  2^8

R_{i+1} =  t_{i}                          XOR
L_{i+1} =  l_{i}                          XOR
j_{i+1} =  j_{i}                          <==
t_{i}   >= 2^8
```

Figure 9

```
R_{0} = 65280
```

Figure 10

```
L_{0} = 2^8 * B_{0} + B_{1}
```

Figure 11

```
j_{0} = 2
```

Figure 12

3.8.1.1.1. Termination

The range coder can be used in 3 modes.

* In "Open mode" when decoding, every symbol the reader attempts to read is available. In this mode arbitrary data can have been appended without affecting the range coder output. This mode is not used in FFV1.

* In "Closed mode" the length in bytes of the bytestream is provided to the range decoder. Bytes beyond the length are read as 0 by the range decoder. This is generally 1 byte shorter than the open mode.

* In "Sentinel mode" the exact length in bytes is not known and thus the range decoder MAY read into the data that follows the range coded bytestream by one byte. In "Sentinel mode", the end of the range coded bytestream is a binary symbol with state 129, which value SHALL be discarded. After reading this symbol, the range decoder will have read one byte beyond the end of the range coded

bytestream. This way the byte position of the end can be determined. Bytestreams written in "Sentinel mode" can be read in "Closed mode" if the length can be determined, in this case the last (sentinel) symbol will be read non-corrupted and be of value 0.

Above describes the range decoding, encoding is defined as any process which produces a decodable bytestream.

There are 3 places where range coder termination is needed in FFV1. First is in the "Configuration Record", in this case the size of the range coded bytestream is known and handled as "Closed mode". Second is the switch from the "Slice Header" which is range coded to Golomb coded slices as "Sentinel mode". Third is the end of range coded Slices which need to terminate before the CRC at their end. This can be handled as "Sentinel mode" or as "Closed mode" if the CRC position has been determined.

3.8.1.2. 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 that 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.

```
pseudo-code                                                   | type
--------------------------------------------------------------|-----
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|
        }                                                     |
    }                                                         |
}                                                             |
```

3.8.1.3. Initial Values for the Context Model

At keyframes all Range Coder state variables are set to their initial state.

3.8.1.4. State Transition Table

```
one_state_{i} =
       default_state_transition_{i} + state_transition_delta_{i}
```

Figure 13

```
zero_state_{i} = 256 - one_state_{256-i}
```

Figure 14

3.8.1.5. 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,
```

3.8.1.6. Alternative State Transition Table

The alternative state transition table has been built using iterative minimization of frame sizes and generally performs better than the default. To use it, the coder_type (see the section on coder_type (#codertype)) MUST be set to 2 and the difference to the default MUST be stored in the "Parameters", see the section on Parameters (#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,
```

3.8.2. Golomb Rice Mode

The end of the bitstream of the "Frame" is filled with 0-bits until that the bitstream contains a multiple of 8 bits.

3.8.2.1. Signed Golomb Rice Codes

This coding mode uses Golomb Rice codes. The VLC is split into 2 parts, the prefix stores the most significant bits and the suffix stores the k least significant bits or stores the whole number in the ESC case.

```
pseudo-code | type --------------------------------------------------------------|----- int get_ur_golomb(k) { | for (prefix = 0; prefix < 12; prefix++) { | if (get_bits(1)) { | return get_bits(k) + (prefix << k) | } | } | return get_bits(bits) + 11 | } | | int get_sr_golomb(k) { | v = get_ur_golomb(k); | if (v & 1) return - (v >> 1) - 1; | else      return   (v >> 1); | }
```

3.8.2.1.1. Prefix

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

Table 1

3.8.2.1.2. 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

Table 2

difference On a small set of test "Samples" the use of prediction slightly improved the compression rate.

3.8.2.3. Scalar Mode

Each difference is coded with the per context mean prediction removed and a per context value for k.

```
get_vlc_symbol(state) { i = state->count; k = 0; while (i < state->error_sum) { k++; i += i; } v = get_sr_golomb(k); if (2 * state->drift < -state->count) { v = -1 - v; } ret = sign_extend(v + state->bias, bits); state->error_sum += abs(v); state->drift     += v; if (state->count == 128) { state->count     >>= 1; state->drift     >>= 1; state->error_sum >>= 1; } state->count++; if (state->drift <= -state->count) { state->bias = max(state->bias - 1, -128); state->drift = max(state->drift + state->count, -state->count + 1); } else if (state->drift > 0) { state->bias = min(state->bias + 1, 127); state->drift = min(state->drift - state->count, 0); } return ret; }
```

3.8.2.4. Initial Values for the VLC context state

At keyframes all coder state variables are set to their initial state.

```
drift     = 0; error_sum = 4; bias      = 0; count     = 1;
```

4. Bitstream

An FFV1 bitstream is composed of a series of 1 or more "Frames" and (when required) a "Configuration Record".

Within the following sub-sections, pseudo-code is used to explain the structure of each FFV1 bitstream component, as described in the section on Pseudo-Code (#pseudocode). The following table lists symbols used to annotate that pseudo-code in order to define the storage of the data referenced in that line of pseudo-code.

4.1.1. version

"version" specifies the version of the FFV1 bitstream.

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

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

Decoders SHOULD reject a file with version >= 3 && ConfigurationRecordIsPresent == 0.

value	version
0	FFV1 version 0
1	FFV1 version 1
2	reserved*
3	FFV1 version 3
4	FFV1 version 4
Other	reserved for future use

Table 5

* 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.

4.1.2. micro_version

"micro_version" specifies the micro-version of the FFV1 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

Table 6

* development versions may be incompatible with the stable variants.

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

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

Table 7

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

4.1.3. coder_type

"coder_type" specifies the coder used.

value	coder used
0	Golomb Rice
1	Range Coder with default state transition table
2	Range Coder with custom state transition table
Other	reserved for future use

Table 8

4.1.4. state_transition_delta

"state_transition_delta" specifies the Range coder custom state transition table.

If state_transition_delta is not present in the FFV1 bitstream, all Range coder custom state transition table elements are assumed to be 0.

4.1.5. colorspace_type

"colorspace_type" specifies the color space encoded, the pixel transformation used by the encoder, the extra plane content, as well as interleave method.

value	color space encoded	pixel transformation	extra plane content	interleave method
0	YCbCr	None	Transparency	"Plane" then "Line"
1	RGB	JPEG2000-RCT	Transparency	"Line" then "Plane"
Other	reserved for future use	reserved for future use	reserved for future use	reserved for future use

Table 9

Restrictions:

If "colorspace_type" is 1, then "chroma_planes" MUST be 1, "log2_h_chroma_subsample" MUST be 0, and "log2_v_chroma_subsample" MUST be 0.

4.1.6. chroma_planes

"chroma_planes" indicates if chroma (color) "Planes" are present.

value	presence
0	chroma "Planes" are not present
1	chroma "Planes" are present

Table 10

4.1.7. bits_per_raw_sample

"bits_per_raw_sample" indicates the number of bits for each "Sample". Inferred to be 8 if not present.

value	bits for each sample
0	reserved*
Other	the actual bits for each "Sample"

Table 11

* Encoders MUST NOT store bits_per_raw_sample = 0 Decoders SHOULD accept and interpret bits_per_raw_sample = 0 as 8.

4.1.8. log2_h_chroma_subsample

"log2_h_chroma_subsample" indicates the subsample factor, stored in powers to which the number 2 must be raised, between luma and chroma width ("chroma_width = 2^-log2_h_chroma_subsample^ * luma_width").

4.1.9. log2_v_chroma_subsample

"log2_v_chroma_subsample" indicates the subsample factor, stored in powers to which the number 2 must be raised, between luma and chroma height ("chroma_height=2^-log2_v_chroma_subsample^ * luma_height").

4.1.10. extra_plane

"extra_plane" indicates if an extra "Plane" is present.

value	presence
0	extra "Plane" is not present
1	extra "Plane" is present

Table 12

4.1.11. num_h_slices

"num_h_slices" indicates the number of horizontal elements of the slice raster.

Inferred to be 1 if not present.

4.1.12. num_v_slices

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

Inferred to be 1 if not present.

4.1.13. quant_table_set_count

"quant_table_set_count" indicates the number of Quantization Table Sets.

Inferred to be 1 if not present.

MUST NOT be 0.

4.1.14. states_coded

"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

Table 13

4.1.15. initial_state_delta

"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 ]
```

Figure 15

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

Figure 16

4.1.16. ec

"ec" indicates the error detection/correction type.

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

Table 14

4.1.17. intra

"intra" indicates the relationship between the instances of "Frame".

Inferred to be 0 if not present.

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

Table 15

4.2. Configuration Record

In the case of a FFV1 bitstream with "version >= 3", a "Configuration Record" is stored in the underlying "Container", at the track header level. It contains the "Parameters" used for all instances of "Frame". The size of the "Configuration Record", "NumBytes", is supplied by the underlying "Container".

```
pseudo-code                                                 | type --------------------------------------------------------------|----- ConfigurationRecord( NumBytes ) {                         | ConfigurationRecordIsPresent = 1                        | Parameters( )                                       | while (remaining_symbols_in_syntax(NumBytes - 4)) {    | reserved_for_future_use                              | br/ur/sr }                                                 | configuration_record_crc_parity                      | u(32) }                                                  |
```

4.2.1. reserved_for_future_use

"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.

4.2.2. configuration_record_crc_parity

"configuration_record_crc_parity" 32 bits that are chosen 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 polynomial used is the standard IEEE CRC polynomial (0x104C11DB7) with initial value 0.

4.2.3. Mapping FFV1 into Containers

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 formats:

4.2.3.1. AVI File Format

The "Configuration Record" extends the stream format chunk ("AVI ", "hdlr", "strl", "strf") with the ConfigurationRecord bitstream.

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.

4.2.3.2. ISO Base Media File Format

The "Configuration Record" extends the sample description box ("moov", "trak", "mdia", "minf", "stbl", "stsd") with a "glbl" box that contains the ConfigurationRecord bitstream. See [ISO.14496-12.2015] 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.

4.2.3.3. 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

4.2.3.4. Matroska File Format

FFV1 SHOULD use "V_FFV1" as the Matroska "Codec ID". For FFV1 versions 2 or less, the Matroska "CodecPrivate" Element SHOULD NOT be used. For FFV1 versions 3 or greater, the Matroska "CodecPrivate" Element MUST contain the FFV1 "Configuration Record" structure and no other data. See [Matroska] for more information about elements.

"NumBytes" is defined as the "Element Data Size" of the "CodecPrivate" Element.

4.3. Frame

A "Frame" is an encoded representation of a complete static image. The whole "Frame" is provided by the underlaying container.

A "Frame" consists of the keyframe field, "Parameters" (if version <=1), and a sequence of independent slices. The pseudo-code below describes the contents of a "Frame".

```
pseudo-code                                    | type --------------------------------------------------------------|----- Frame( NumBytes ) {                    | keyframe                    | br if (keyframe && !ConfigurationRecordIsPresent {        | Parameters( )                    | }                    | while (remaining_bits_in_bitstream( NumBytes )) {       | Slice( )                    | }                    | }                    |
```

Architecture overview of slices in a "Frame":

```
+-----------------------------------------------------------------+
+=================================================================+
| first slice header                                              |
+-----------------------------------------------------------------+
| first slice content                                             |
+-----------------------------------------------------------------+
| first slice footer                                              |
+-----------------------------------------------------------------+
| --------------------------------------------------------------- |
+-----------------------------------------------------------------+
| second slice header                                             |
+-----------------------------------------------------------------+
| second slice content                                            |
+-----------------------------------------------------------------+
| second slice footer                                             |
+-----------------------------------------------------------------+
| --------------------------------------------------------------- |
+-----------------------------------------------------------------+
| ...                                                             |
+-----------------------------------------------------------------+
| --------------------------------------------------------------- |
+-----------------------------------------------------------------+
| last slice header                                               |
+-----------------------------------------------------------------+
| last slice content                                              |
+-----------------------------------------------------------------+
| last slice footer                                               |
+-----------------------------------------------------------------+
```

Table 16

4.5. Slice

A "Slice" is an independent spatial sub-section of a "Frame" that is encoded separately from an other region of the same "Frame". The use of more than one "Slice" per "Frame" can be useful for taking advantage of the opportunities of multithreaded encoding and decoding.

A "Slice" consists of a "Slice Header" (when relevant), a "Slice Content", and a "Slice Footer" (when relevant). The pseudo-code below describes the contents of a "Slice".

```
pseudo-code                                                   | type
--------------------------------------------------------------|-----
Slice( ) {                                                    |
    if (version >= 3) {                                       |
        SliceHeader( )                                        |
    }                                                         |
    SliceContent( )                                           |
    if (coder_type == 0) {                                    |
        while (!byte_aligned()) {                             |
            padding                                           | u(1)
        }                                                     |
    }                                                         |
    if (version <= 1) {                                       |
        while (remaining_bits_in_bitstream( NumBytes ) != 0) {| reserved
                                                              | u(1)
        }                                                     |
    }                                                         |
    if (version >= 3) {                                       |
        SliceFooter( )                                        |
    }                                                         |
}                                                             |
```

"padding" specifies a bit without any significance and used only for byte alignment. "padding" MUST be 0.

"reserved" specifies a bit without any significance in this revision of the specification and may have a significance in a later revision of this specification.

Encoders SHOULD NOT fill these bits.

Decoders SHOULD ignore these bits.

Note in case these bits are used in a later revision of this specification: any revision of this specification SHOULD care about avoiding to add 40 bits of content after "SliceContent" for version 0 and 1 of the bitstream. Background: due to some non conforming encoders, some bitstreams where found with 40 extra bits corresponding to "error_status" and "slice_crc_parity", a decoder conforming to the revised specification could not do the difference between a revised bitstream and a buggy bitstream.

4.6. Slice Header

A "Slice Header" provides information about the decoding configuration of the "Slice", such as its spatial position, size, and aspect ratio. The pseudo-code below describes the contents of the "Slice Header".

```
pseudo-code                                                   | type
--------------------------------------------------------------|-----
SliceHeader( ) {                                              |
    slice_x                                                   | ur
    slice_y                                                   | ur
    slice_width - 1                                           | ur
    slice_height - 1                                          | ur
    for (i = 0; i < quant_table_set_index_count; i++) {       |
        quant_table_set_index[ i ]                            | ur
    }                                                         |
    picture_structure                                         | ur
    sar_num                                                   | ur
    sar_den                                                   | ur
    if (version >= 4) {                                       |
        reset_contexts                                        | br
        slice_coding_mode                                     | ur
    }                                                         |
}                                                             |
```

4.6.1. slice_x

"slice_x" indicates the x position on the slice raster formed by

Inferred to be 0 if not present.

4.5.2. slice_y

"slice_y" indicates the y position on the slice raster formed by num_v_slices.

Inferred to be 0 if not present.

4.5.3. slice_width

"slice_width" indicates the width on the slice raster formed by num_h_slices.

Inferred to be 1 if not present.

4.5.4. slice_height

"slice_height" indicates the height on the slice raster formed by num_v_slices.

Inferred to be 1 if not present.

4.5.5. quant_table_set_index_count

"quant_table_set_index_count" is defined as "1 + ((chroma_planes || version <= 3) ? 1 : 0) + (extra_plane ? 1 : 0)".

4.5.6. quant_table_set_index

"quant_table_set_index" indicates the Quantization Table Set index to select the Quantization Table Set and the initial states for the slice.

Inferred to be 0 if not present.

4.5.7. picture_structure

"picture_structure" specifies the temporal and spatial relationship of each "Line" of the "Frame".

Inferred to be 0 if not present.

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

Table 17

4.5.8. sar_num

"sar_num" specifies the "Sample" aspect ratio numerator.

Inferred to be 0 if not present.

A value of 0 means that aspect ratio is unknown.

Encoders MUST write 0 if "Sample" aspect ratio is unknown.

If "sar_den" is 0, decoders SHOULD ignore the encoded value and consider that "sar_num" is 0.

4.5.9. sar_den

"sar_den" specifies the "Sample" aspect ratio denominator.

Inferred to be 0 if not present.

A value of 0 means that aspect ratio is unknown.

Encoders MUST write 0 if "Sample" aspect ratio is unknown.

If "sar_num" is 0, decoders SHOULD ignore the encoded value and consider that "sar_den" is 0.

4.5.10. reset_contexts

"reset_contexts" indicates if slice contexts must be reset.

Inferred to be 0 if not present.

4.5.11. slice_coding_mode

"slice_coding_mode" indicates the slice coding mode.

Inferred to be 0 if not present.

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

Table 18

4.6. Slice Content

A "Slice Content" contains all "Line" elements part of the "Slice".

Depending on the configuration, "Line" elements are ordered by "Plane" then by row (YCbCr) or by row then by "Plane" (RGB).

```
pseudo-code | type --------------------------------------------------------------|----- SliceContent( ) { | if (colorspace_type == 0) { | for (p = 0; p < primary_color_count; p++) { | for (y = 0; y < plane_pixel_height[ p ]; y++) { | Line( p, y ) | } | } | } else if (colorspace_type == 1) { | for (y = 0; y < slice_pixel_height; y++) { | for (p = 0; p < primary_color_count; p++) { | Line( p, y ) | } | } | } | } |
```

4.6.1. primary_color_count

"primary_color_count" is defined as "1 + (chroma_planes ? 2 : 0) + (extra_plane ? 1 : 0)".

4.6.2. plane_pixel_height

"plane_pixel_height[p]" is the height in pixels of plane p of the slice.

"plane_pixel_height[0]" and "plane_pixel_height[1 + (chroma_planes ? 2 : 0)]" value is "slice_pixel_height".

If "chroma_planes" is set to 1, "plane_pixel_height[1]" and "plane_pixel_height[2]" value is "ceil(slice_pixel_height / log2_v_chroma_subsample)".

4.6.3. slice_pixel_height

"slice_pixel_height" is the height in pixels of the slice.

Its value is "floor((slice_y + slice_height) * slice_pixel_height / num_v_slices) - slice_pixel_y".

4.6.4. slice_pixel_y

"slice_pixel_y" is the slice vertical position in pixels.

Its value is "floor(slice_y * frame_pixel_height / num_v_slices)".

4.7. Line

A "Line" is a list of the sample differences (relative to the predictor) of primary color components. The pseudo-code below describes the contents of the "Line".

```
pseudo-code                                         | type --------------------------------------------------------------|----- Line( p, y ) { | if (colorspace_type == 0) { | for (x = 0; x < plane_pixel_width[ p ]; x++) { | sample_difference[ p ][ y ][ x ] | } | } else if (colorspace_type == 1) { | for (x = 0; x < slice_pixel_width; x++) { | sample_difference[ p ][ y ][ x ] | } | } | } |
```

4.7.1. plane_pixel_width

"plane_pixel_width[p]" is the width in "Pixels" of "Plane" p of the slice.

"plane_pixel_width[0]" and "plane_pixel_width[1 + (chroma_planes ? 2 : 0)]" value is "slice_pixel_width".

If "chroma_planes" is set to 1, "plane_pixel_width[1]" and "plane_pixel_width[2]" value is "ceil(slice_pixel_width / (1 << log2_h_chroma_subsample))".

4.7.2. slice_pixel_width

"slice_pixel_width" is the width in "Pixels" of the slice.

Its value is "floor((slice_x + slice_width) * slice_pixel_width / num_h_slices) - slice_pixel_x".

4.7.3. slice_pixel_x

"slice_pixel_x" is the slice horizontal position in "Pixels".

Its value is "floor(slice_x * frame_pixel_width / num_h_slices)".

4.7.4. sample_difference

"sample_difference[p][y][x]" is the sample difference for "Sample" at "Plane" "p", y position "y", and x position "x". The "Sample" value is computed based on median predictor and context described in the section on Samples (#samples).

4.8. Slice Footer

A "Slice Footer" provides information about slice size and (optionally) parity. The pseudo-code below describes the contents of the "Slice Footer".

Note: "Slice Footer" is always byte aligned.

```
pseudo-code                                                   | type --------------------------------------------------------------|----- SliceFooter( ) {                                        | slice_size                                    | u(24) if (ec) {                                    | error_status                                    | u(8) slice_crc_parity                                    | u(32) }                                    | }                                    |
```

4.8.1. slice_size

"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 parallel decoding as well as error resilience.

4.8.2. error_status

"error_status" specifies the error status.

value	error status
0	no error
1	slice contains a correctable error
2	slice contains a uncorrectable error
Other	reserved for future use

Table 19

4.8.3. slice_crc_parity

"slice_crc_parity" 32 bits that are chosen 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 polynomial used is the standard IEEE CRC polynomial (0x104C11DB7) with initial value 0.

4.9. Quantization Table Set

The Quantization Table Sets are stored by storing the number of equal entries -1 of the first half of the table (represented as "len - 1" in the pseudo-code below) using the method described in Range Non Binary Values (#range-non-binary-values). The second half doesn't need to be stored as it is identical to the first with flipped sign. "scale" and "len_count[i][j]" are temporary values used for the computing of "context_count[i]" and are not used outside Quantization Table Set pseudo-code.

Example:

Table: 0 0 1 1 1 1 2 2 -2 -2 -2 -1 -1 -1 -1 0

Stored values: 1, 3, 1

```
pseudo-code | type --------------------------------------------------------------|------ QuantizationTableSet( i ) { | scale = 1 | for (j = 0; j < MAX_CONTEXT_INPUTS; j++) { | QuantizationTable( i, j, scale ) | scale *= 2 * len_count[ i ][ j ] - 1 | } | context_count[ i ] = ceil( scale / 2 ) | } |
```

MAX_CONTEXT_INPUTS is 5.

```
pseudo-code | type --------------------------------------------------------------|------ QuantizationTable(i, j, scale) { | v = 0 | for (k = 0; k < 128;) { | len - 1 | ur 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 | } |
```

4.9.1. quant_tables

"quant_tables[i][j][k]" indicates the quantification table value of the Quantized Sample Difference "k" of the Quantization Table "j" of the Set Quantization Table Set "i".

4.9.2. context_count

"context_count[i]" indicates the count of contexts for Quantization Table Set "i".

5. Restrictions

To ensure that fast multithreaded decoding is possible, starting with version 3 and if "frame_pixel_width * frame_pixel_height" is more than 101376, "slice_width * slice_height" MUST be less or equal to "num_h_slices * num_v_slices / 4". Note: 101376 is the frame size in "Pixels" of a 352x288 frame also known as CIF ("Common Intermediate Format") frame size format.

For each "Frame", each position in the slice raster MUST be filled by one and only one slice of the "Frame" (no missing slice position, no slice overlapping).

For each "Frame" with keyframe value of 0, each slice MUST have the same value of "slice_x, slice_y, slice_width, slice_height" as a slice in the previous "Frame", except if "reset_contexts" is 1.

6. Security Considerations

Like any other codec, (such as [RFC6716]), FFV1 should not be used with insecure ciphers or cipher-modes that are vulnerable to known plaintext attacks. Some of the header bits as well as the padding are easily predictable.

Implementations of the FFV1 codec need to take appropriate security considerations into account, as outlined in [RFC4732]. It is extremely important for the decoder to be robust against malicious payloads. Malicious payloads must not cause the decoder to overrun its allocated memory or to take an excessive amount of resources to decode. The same applies to the encoder, even though problems in encoders are typically rarer. Malicious video streams must not cause the encoder to misbehave because this would allow an attacker to attack transcoding gateways. A frequent security problem in image and video codecs is also to not check for integer overflows in "Pixel" count computations, that is to allocate width * height without considering that the multiplication result may have overflowed the arithmetic types range. The range coder could, if implemented naively, read one byte over the end. The implementation must ensure that no read outside allocated and initialized memory occurs.

The reference implementation [REFIMPL] contains no known buffer overflow or cases where a specially crafted packet or video segment could cause a significant increase in CPU load.

The reference implementation [REFIMPL] was validated in the following conditions:

* Sending the decoder valid packets generated by the reference encoder and verifying that the decoder’s output matches the encoder’s input.

* Sending the decoder packets generated by the reference encoder and then subjected to random corruption.

* Sending the decoder random packets that are not FFV1.

In all of the conditions above, the decoder and encoder was run inside the [VALGRIND] memory debugger as well as clangs address sanitizer [Address-Sanitizer], which track reads and writes to invalid memory regions as well as the use of uninitialized memory. There were no errors reported on any of the tested conditions.

7. Media Type Definition

This registration is done using the template defined in [RFC6838] and following [RFC4855].

Type name: video

Subtype name: FFV1

Required parameters: None.

Optional parameters:

This parameter is used to signal the capabilities of a receiver implementation. This parameter MUST NOT be used for any other purpose.

version: The version of the FFV1 encoding as defined by the section on version (#version).

micro_version: The micro_version of the FFV1 encoding as defined by the section on micro_version (#micro-version).

coder_type: The coder_type of the FFV1 encoding as defined by the section on coder_type (#coder-type).

colorspace_type: The colorspace_type of the FFV1 encoding as defined by the section on colorspace_type (#colorspace-type).

bits_per_raw_sample: The bits_per_raw_sample of the FFV1 encoding as defined by the section on bits_per_raw_sample (#bits-per-raw-sample).

max-slices: The value of max-slices is an integer indicating the maximum count of slices with a frames of the FFV1 encoding.

Encoding considerations:

This media type is defined for encapsulation in several audiovisual container formats and contains binary data; see the section on "Mapping FFV1 into Containers" (#mapping-ffv1-into-containers). This media type is framed binary data Section 4.8 of [RFC6838].

Security considerations:

See the "Security Considerations" section (#security-considerations) of this document.

Interoperability considerations: None.

Published specification:

[I-D.ietf-cellar-ffv1] and RFC XXXX.

[RFC Editor: Upon publication as an RFC, please replace "XXXX" with the number assigned to this document and remove this note.]

Applications which use this media type:

Any application that requires the transport of lossless video can use this media type. Some examples are, but not limited to screen recording, scientific imaging, and digital video preservation.

Fragment identifier considerations: N/A.

Additional information: None.

Person & email address to contact for further information: Michael Niedermayer michael@niedermayer.cc (mailto:michael@niedermayer.cc)

Intended usage: COMMON

Restrictions on usage: None.

Author: Dave Rice dave@dericed.com (mailto:dave@dericed.com)

Change controller: IETF cellar working group delegated from the IESG.

8. IANA Considerations

The IANA is requested to register the following values:

* Media type registration as described in Media Type Definition (#media-type-definition).

9. Appendixes

9.1. Decoder implementation suggestions

9.1.1. Multi-threading Support and Independence of Slices

The FFV1 bitstream is parsable in two ways: in sequential order as described in this document or with the pre-analysis of the footer of each slice. Each slice footer contains a slice_size field so the boundary of each slice is computable without having to parse the slice content. That allows multi-threading as well as independence of slice content (a bitstream error in a slice header or slice content has no impact on the decoding of the other slices).

After having checked keyframe field, a decoder SHOULD parse slice_size fields, from slice_size of the last slice at the end of the "Frame" up to slice_size of the first slice at the beginning of the "Frame", before parsing slices, in order to have slices boundaries. A decoder MAY fallback on sequential order e.g. in case of a corrupted "Frame" (frame size unknown, slice_size of slices not coherent...) or if there is no possibility of seeking into the stream.

10. Changelog

See https://github.com/FFmpeg/FFV1/commits/master (https://github.com/FFmpeg/FFV1/commits/master)

11. Normative References

[I-D.ietf-cellar-ffv1]
Niedermayer, M., Rice, D., and J. Martinez, "FFV1 Video Coding Format Version 0, 1, and 3", Internet Draft, draft-ietf-cellar-ffv1-08, August 13, 2019, <https://www.ietf.org/archive/id/draft-ietf-cellar-ffv1-08>.

[ISO.15444-1.2016]
International Organization for Standardization, "Information technology -- JPEG 2000 image coding system: Core coding system", October 2016.

[ISO.9899.1990]
International Organization for Standardization, "Programming languages - C", 1990.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC4732] Handley, M., Ed., Rescorla, E., Ed., and IAB, "Internet Denial-of-Service Considerations", RFC 4732, DOI 10.17487/RFC4732, December 2006, <https://www.rfc-editor.org/info/rfc4732>.

[RFC4855] Casner, S., "Media Type Registration of RTP Payload Formats", RFC 4855, DOI 10.17487/RFC4855, February 2007, <https://www.rfc-editor.org/info/rfc4855>.

[RFC6716] Valin, JM., Vos, K., and T. Terriberry, "Definition of the Opus Audio Codec", RFC 6716, DOI 10.17487/RFC6716, September 2012, <https://www.rfc-editor.org/info/rfc6716>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

12. Informative References

[Address-Sanitizer] The Clang Team, "ASAN AddressSanitizer website", September 2019, <https://clang.llvm.org/docs/AddressSanitizer.html>.

[AVI] Microsoft, "AVI RIFF File Reference", September 2019, <https://msdn.microsoft.com/en-us/library/windows/desktop/dd318189%28v=vs.85%29.aspx>.

[HuffYUV] Rudiak-Gould, B., "HuffYUV", December 2003, <https://web.archive.org/web/20040402121343/http://cultact-server.novi.dk/kpo/huffyuv/huffyuv.html>.

[ISO.14495-1.1999] International Organization for Standardization, "Information technology -- Lossless and near-lossless compression of continuous-tone still images: Baseline", December 1999.

[ISO.14496-10.2014] International Organization for Standardization, "Information technology -- Coding of audio-visual objects -- Part 10: Advanced Video Coding", September 2014.

[ISO.14496-12.2015] International Organization for Standardization, "Information technology -- Coding of audio-visual objects -- Part 12: ISO base media file format", December 2015.

[Matroska] IETF, "Matroska", 2016, <https://datatracker.ietf.org/doc/draft-lhomme-cellar-matroska/>.

[NUT] Niedermayer, M., "NUT Open Container Format", December 2013, <https://ffmpeg.org/~michael/nut.txt>.

[range-coding] Nigel, G. and N. Martin, "Range encoding: an algorithm for removing redundancy from a digitised message.", July 1979.

[REFIMPL] Niedermayer, M., "The reference FFV1 implementation / the FFV1 codec in FFmpeg", September 2019, <https://ffmpeg.org>.

[VALGRIND] Valgrind Developers, "Valgrind website", September 2019, <https://valgrind.org/>.

[YCbCr] Wikipedia, "YCbCr", September 2019, <https://en.wikipedia.org/w/index.php?title=YCbCr>.

Authors' Addresses

Michael Niedermayer

Email: michael@niedermayer.cc

Dave Rice

Email: dave@dericed.com

Jerome Martinez

Email: jerome@mediaarea.net