cellar
Internet-Draft
Intended status: Informational
Expires: December 2, 2018

M. Niedermayer

D. Rice

J. Martinez

May 31, 2018

FFV1 Video Coding Format Version 0, 1, and 3
draft-ietf-cellar-ffv1-03

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

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 2, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 4
2. Notation and Conventions 4
2.1. Definitions 4
2.2. Conventions 5
2.2.1. Arithmetic operators 5
2.2.2. Assignment operators 6
2.2.3. Comparison operators 6
2.2.4. Mathematical functions 7
2.2.5. Order of operation precedence 7
2.2.6. Pseudo-code 8
2.2.7. Range 8
2.2.8. NumBytes 8
2.2.9. Bitstream functions 8
3. General Description 8
3.1. Border 9
3.2. Samples 9
3.3. Median predictor 10
3.4. Context 10
3.5. Quantization Table Sets 11
3.6. Quantization Table Set indexes 11
3.7. Color spaces 12
3.7.1. YCbCr 12
3.7.2. RGB 12
3.8. Coding of the Sample Difference 14
3.8.1. Range coding mode 14
3.8.2. Golomb Rice mode 17
4. Bitstream 19
4.1. Configuration Record 20
4.1.1. reserved_for_future_use 21
4.1.2. configuration_record_crc_parity 21
4.1.3. Mapping FFV1 into Containers 21
4.2. Frame 22
4.3. Slice 22
4.4. Slice Header 23
4.4.1. slice_x 23
4.4.2. slice_y 23
4.4.3. slice_width 24
4.4.4. slice_height 24
4.4.5. quant_table_set_index_count 24
4.4.6. quant_table_set_index 24
4.4.7. picture_structure 24
4.4.8. sar_num 24
4.4.9. sar_den 25

4.5. Slice Content 25
4.5.1. primary_color_count 25
4.5.2. plane_pixel_height 25
4.5.3. slice_pixel_height 25
4.5.4. slice_pixel_y 25
4.6. Line 26
4.6.1. plane_pixel_width 26
4.6.2. slice_pixel_width 26
4.6.3. slice_pixel_x 26
4.6.4. sample_difference 26
4.7. Slice Footer 26
4.7.1. slice_size 27
4.7.2. error_status 27
4.7.3. slice_crc_parity 27
4.8. Parameters 27
4.8.1. version 28
4.8.2. micro_version 29
4.8.3. coder_type 29
4.8.4. state_transition_delta 30
4.8.5. colorspace_type 30
4.8.6. chroma_planes 30
4.8.7. bits_per_raw_sample 30
4.8.8. log2_h_chroma_subsample 31
4.8.9. log2_v_chroma_subsample 31
4.8.10. alpha_plane 31
4.8.11. num_h_slices 31
4.8.12. num_v_slices 31
4.8.13. quant_table_set_count 32
4.8.14. states_coded 32
4.8.15. initial_state_delta 32
4.8.16. ec 32
4.8.17. intra 32
4.9. Quantization Table Set 33
4.9.1. quant_tables 34
4.9.2. context_count 34
5. Restrictions 34
6. Security Considerations 35
7. Media Type Definition 35
8. IANA Considerations 37
9. Appendixes 37
9.1. Decoder implementation suggestions 37
9.1.1. Multi-threading support and independence of slices . 37
10. Changelog 38
11. ToDo 38
12. References 38
12.1. Normative References 38
12.2. Informative References 39
Authors' Addresses 40

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 0, 1, and 3 of FFV1. The distinctions of the versions are provided throughout the document, but in summary:

- Version 0 of FFV1 was the original implementation of FFV1 and has been in non-experimental use since April 14, 2006 [FFV1_V0].

- Version 1 of FFV1 adds support of more video bit depths and has been in use since April 24, 2009 [FFV1_V1].

- Version 2 of FFV1 only existed in experimental form and is not described by this document.

- Version 3 of FFV1 adds several features such as increased description of the characteristics of the encoding images and embedded CRC data to support fixity verification of the encoding. Version 3 has been in non-experimental use since August 17, 2013 [FFV1_V3].

The latest version of this document is available at <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" 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, Alpha, Red, Green, and Blue.

"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.

"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. 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

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

2.2.1. 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.

"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.

2.2.2. 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.

2.2.3. 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.

2.2.4. Mathematical functions

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

ceil(a) the largest integer less 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

2.2.5. 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 : c
a = b, a += b, a -= b, a *= b
```

2.2.6. 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] as uses its "if/else", "while" and "for" functions as well as functions defined within this document.

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 particular FFV1 "Configuration Record" or "Frame". FFV1 relies on its "Container" to store the "NumBytes" values, see Section 4.1.3.

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. byte_aligned

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

2.2.9.3. 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. General Description

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 Section 3.8.

3.1. Border

A border is assumed for each coded slice for the purpose of the 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

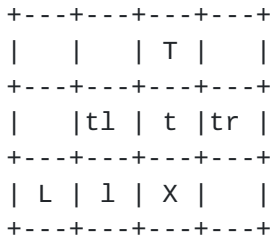
- an additional column of samples to the left of the coded slice and two rows of samples above the coded slice are assumed to be "0"

The following table depicts a slice of samples "a,b,c,d,e,f,g,h,i" along with its assumed border.



3.2. Samples

Positions used for context and median predictor are:



"X" is the current processed Sample. The identifiers 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 media predictor: if "colorspace_type == 0 && bits_per_raw_sample == 16 && (coder_type == 1 || coder_type == 2)", the following media 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. 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 media 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]
```

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, 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]
```

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 Alpha 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 two color spaces: YCbCr and RGB. Both color spaces allow an optional Alpha plane that can be used to code transparency data.

3.7.1. YCbCr

In YCbCr color space, 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 FFV1 "Frame" using YCbCr MUST use one of the following arrangements:

- Y
- Y, Alpha
- Y, Cb, Cr
- Y, Cb, Cr, Alpha

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 an Alpha (transparency) plane is used, then it MUST be coded last.

3.7.2. RGB

JPEG2000-RCT is a Reversible Color Transform that codes RGB (red, green, blue) planes losslessly in a modified YCbCr color space. 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
```

Exception for the JPEG2000-RCT conversion: if bits_per_raw_sample is between 9 and 15 inclusive and alpha_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

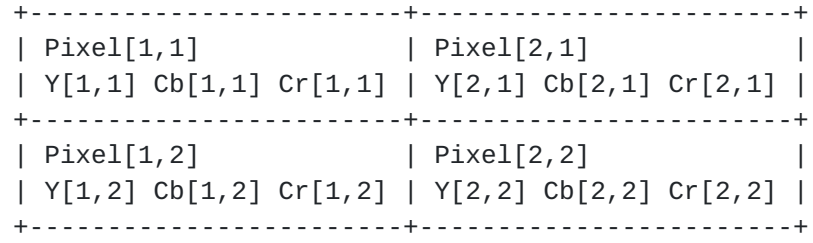
g=Cb+b
```

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 alpha_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.

[ISO.15444-1.2016]

When FFV1 uses the JPEG2000-RCT, the horizontal 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, then Cb, then Cr, and then if used Alpha.

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



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 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)
```

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, which is 128. The length of the bytestream encoding n binary symbols is "j_{n}" bytes.

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

 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

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

 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
```

$$R_{0} = 65280$$

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

$$j_{0} = 2$$

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}

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

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 MUST be set to 2 and the difference to the default MUST 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,
```

3.8.2. Golomb Rice mode

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. 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. Prefix

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

3.8.2.2. 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

3.8.2.3. 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

3.8.2.4. 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 are coded.

3.8.2.5. 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 that 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.

```
pseudo-code                                                    | type
---------------------------------------------------------------|-----
log2_run[41]={                                                 |
 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_bits(1)) {                                         |
        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;                                          |
    }                                                          |
}                                                              |
```

The log2_run function is also used within [ISO.14495-1.1999].

3.8.2.6. 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.

4. Bitstream

Symbol	Definition
u(n)	unsigned big endian integer using n bits
sg	Golomb Rice coded signed scalar symbol coded with the method described in Section 3.8.2
br	Range coded Boolean (1-bit) symbol with the method described in Section 3.8.1.1
ur	Range coded unsigned scalar symbol coded with the method described in Section 3.8.1.2
sr	Range coded signed scalar symbol coded with the method described in Section 3.8.1.2

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

The following MUST be provided by external means during initialization of the decoder:

"frame_pixel_width" is defined as "Frame" width in pixels.

"frame_pixel_height" is defined as "Frame" height in pixels.

Default values at the decoder initialization phase:

"ConfigurationRecordIsPresent" is set to 0.

4.1. 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_bits_in_bitstream(NumBytes) > 32)	
reserved_for_future_use	u(1)
configuration_record_crc_parity	u(32)
}	

4.1.1. 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.

4.1.2. 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.

4.1.3. 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:

4.1.3.1. 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.

4.1.3.2. 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.

4.1.3.3. 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

4.1.3.4. 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.

4.2. Frame

A "Frame" consists of the keyframe field, parameters (if version <=1), and a sequence of independent slices.

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

4.3. Slice

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

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

4.4. 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
}                                                              |
```

4.4.1. slice_x

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

4.4.2. slice_y

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

4.4.3. slice_width

"slice_width" indicates the width on the slice raster formed by num_h_slices.
Inferred to be 1 if not present.

4.4.4. slice_height

"slice_height" indicates the height on the slice raster formed by num_v_slices.
Inferred to be 1 if not present.

4.4.5. quant_table_set_index_count

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

4.4.6. 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.

4.4.7. 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

4.4.8. sar_num

"sar_num" specifies the sample aspect ratio numerator.
Inferred to be 0 if not present.
MUST be 0 if sample aspect ratio is unknown.

4.4.9. sar_den

"sar_den" specifies the sample aspect ratio numerator.
Inferred to be 0 if not present.
MUST be 0 if sample aspect ratio is unknown.

4.5. Slice Content

```
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 )                                   |
    }                                                          |
}                                                              |
```

4.5.1. primary_color_count

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

4.5.2. 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)".

4.5.3. 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".

4.5.4. 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)".

4.6. 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 ]                   |
    }                                                          |
}                                                              |
```

4.6.1. 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))".

4.6.2. 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".

4.6.3. 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)".

4.6.4. 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 prediction and context described in Section 3.2.

4.7. 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)
    }                                                        |
}                                                            |
```

4.7.1. 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 this way parallel decoding as well as error resilience.

4.7.2. 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

4.7.3. 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.

4.8. Parameters

```
pseudo-code                                                     | type
----------------------------------------------------------------|-----
Parameters( ) {                                                 |
    version                                                     | ur
    if (version >= 3)                                           |
        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 >= 1)                                           |
        bits_per_raw_sample                                     | ur
    chroma_planes                                               | br
    log2_h_chroma_subsample                                     | ur
    log2_v_chroma_subsample                                     | ur
    alpha_plane                                                 | br
    if (version >= 3) {                                         |
        num_h_slices - 1                                        | ur
        num_v_slices - 1                                        | ur
        quant_table_set_count                                   | ur
    }                                                           |
    for( i = 0; i < quant_table_set_count; i++ )                |
        QuantizationTableSet( i )                               |
    if (version >= 3) {                                         |
        for( i = 0; i < quant_table_set_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
    }                                                           |
}                                                               |
```

4.8.1. version

"version" specifies the version of the FFV1 bitstream.
Each version is incompatible with others versions: decoders SHOULD reject a file due to 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
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.

4.8.2. 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

* development versions may be incompatible with the stable variants.

4.8.3. 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

4.8.4. 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.

4.8.5. colorspace_type

"colorspace_type" specifies color space losslessly encoded, Pixel transformation used by the encoder, as well as interleave method.

value	color space losslessly encoded	transformation	interleave method
0	YCbCr	No Pixel transformation	plane then line
1	RGB	JPEG2000-RCT	line then plane
Other	reserved for future use	reserved for future use	reserved for future use

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.

4.8.6. chroma_planes

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

value	presence
0	chroma planes are not present
1	chroma planes are present

4.8.7. 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

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

4.8.8. 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").

4.8.9. 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").

4.8.10. alpha_plane

"alpha_plane" indicates if a transparency plane is present.

value	presence
0	transparency plane is not present
1	transparency plane is present

4.8.11. num_h_slices

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

4.8.12. num_v_slices

"num_v_slices" indicates the number of vertical elements of the slice raster.
Inferred to be 1 if not present.

4.8.13. 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.

4.8.14. 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

4.8.15. 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 ] : 128

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

4.8.16. 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

4.8.17. 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

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 Section 3.8.1.2. 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 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".

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. Although problems in encoders are typically rarer, the same applies to the encoder. 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 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 Section 4.8.1.

micro_version: The micro_version of the FFV1 encoding as defined by Section 4.8.2.

coder_type: The coder_type of the FFV1 encoding as defined by Section 4.8.3.

colorspace_type: The colorspace_type of the FFV1 encoding as defined by Section 4.8.5.

bits_per_raw_sample: The version of the FFV1 encoding as defined by Section 4.8.7.

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 Section 4.1.3. This media type is framed binary data Section 4.8 of [RFC4288].

Security considerations:

See Section 6 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 <mailto:michael@niedermayer.cc>

Intended usage: COMMON

Restrictions on usage: None.

Author: Dave Rice <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:

o Media type registration as described in Section 7.

9. Appendixes

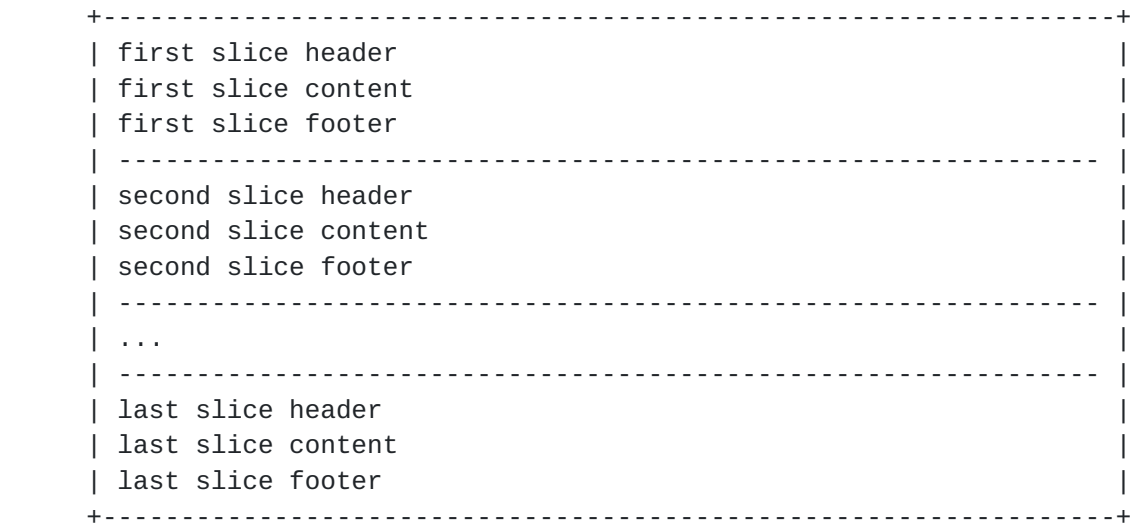
9.1. Decoder implementation suggestions

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 seek into the stream.

Architecture overview of slices in a "Frame":



10. Changelog

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

11. ToDo

o mean,k estimation for the Golomb Rice codes

12. References

12.1. Normative References

[I-D.ietf-cellar-ffv1]
Niedermayer, M., Rice, D., and J. Martinez, "FFV1 Video Coding Format Version 0, 1, and 3", draft-ietf-cellar-ffv1-02 (work in progress), April 2018.

[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", ISO Standard 9899, 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>.

[RFC4288] Freed, N. and J. Klensin, "Media Type Specifications and Registration Procedures", RFC 4288, DOI 10.17487/RFC4288, December 2005, <https://www.rfc-editor.org/info/rfc4288>.

[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>.

[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>.

12.2. Informative References

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

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

[FFV1_V0] Niedermayer, M., "Commit to mark FFV1 version 0 as non-experimental", April 2006, <https://git.videolan.org/?p=ffmpeg.git;a=commit;h=b548f2b91b701e1235608ac882ea6df915167c7e>.

[FFV1_V1] Niedermayer, M., "Commit to release FFV1 version 1", April 2009, <https://git.videolan.org/?p=ffmpeg.git;a=commit;h=68f8d33becbd73b4d0aa277f472a6e8e72ea6849>.

[FFV1_V3] Niedermayer, M., "Commit to mark FFV1 version 3 as non-experimental", August 2013, <https://git.videolan.org/?p=ffmpeg.git;a=commit;h=abe76b851c05eea8743f6c899cbe5f7409b0f301>.

[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.", Proc. Institution of Electronic and Radio Engineers International Conference on Video and Data Recording , July 1979.

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

[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>.

[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>.

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

[YCbCr] Wikipedia, "YCbCr", undated, <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