Description of the 4XM Video Codec

by Michael Niedermayer <michaelni@gmx.at>

8th June 2003

Contents

1 Introduction

The 4XM video codec is a mixture between a very simplified JPEG scheme and rectangular block based fullpel motion compensation with DC difference coding. The codec uses 4:2:0 YCbCr colorspace for the JPEG part but converts it to RGB16 before using it for motion compensation.

The latest version of this document is available at http://www.mplayerhq.hu/~michael/4xm.{lyx,txt,html,ps}

4XM video is normally encapsulated in the proprietary 4XM format `http://www.pcisys.net/~melanson/codecs/4xm-format.txt`.

This document assumes familiarity with mathematical and coding concepts such as the discrete cosine transform, quantization, YCbCr colorspaces, macroblocks, and variable length codes (VLCs). A familiarity with the standard JPEG coding method is also helpful.

2 Terms and Definitions

AC — Any DCT coefficient for which the frequency in one or both dimensions is non-zero.

DC — The DCT coefficient for which the frequency is zero in both dimensions

(I)DCT — (Inverse) Discrete Cosine Transform

VLC — Variable Length Code

AAN IDCT — IDCT algorithm by Arai, Agui, and Nakajima

JPEG — Joint Photographic Expert Group

3 High-level Description

The 4XM video coding method embodies 3 types of frames: I-frames, P-frames, and C-frames. I-frames are intraframes and stand on their own. P and C-frames are Interframes.

3.1 I-Frame

I-frames are practically the same as JPEG images. Differences include just a single Huffman table, different headers, and a bitstream split into 2 partitions with one partition written in 32-bit byteswapped order. There are also no parameters for rate or quality control.

The picture is split into macroblocks which are coded left->right, top->bottom.

3.1.1 Macroblock

16x16 luma + 8x8 chroma as 4 8x8 luma blocks and 2 8x8 chroma blocks :

Y:

0	1
2	3

Cb: $\boxed{4}$ Cr: $\boxed{5}$

3.1.2 DC Prediction

DC values are predicted from the last coded block. The initial prediction value used for the first top left luma block is 0. No special handling is done between luma and chroma blocks or at the right border, so the DC value of the rightmost 8x8 Cr block of the first row will be used as the predictor for the first/top-left 8x8 luma block of the second MB row.

3.1.3 Dequantization and IDCT

4XM uses an AAN IDCT with the premultiply table merged with the quantization table. The quantization table is the default luma table used in JPEG.

default luma quantization table used in JPEG

```
16,  11,  10,  16,  24,  40,  51,  61,
12,  12,  14,  19,  26,  58,  60,  55,
14,  13,  16,  24,  40,  57,  69,  56,
14,  17,  22,  29,  51,  87,  80,  62,
18,  22,  37,  56,  68, 109, 103,  77,
24,  35,  55,  64,  81, 104, 113,  92,
49,  64,  78,  87, 103, 121, 120, 101,
72,  92,  95,  98, 112, 100, 103,  99
```

AAN premultiply table

```
16384, 22725, 21407, 19266, 16384, 12873,  8867,  4520,
22725, 31521, 29692, 26722, 22725, 17855, 12299,  6270,
21407, 29692, 27969, 25172, 21407, 16819, 11585,  5906,
19266, 26722, 25172, 22654, 19266, 15137, 10426,  5315,
16384, 22725, 21407, 19266, 16384, 12873,  8867,  4520,
12873, 17855, 16819, 15137, 12873, 10114,  6967,  3552,
8867 , 12299, 11585, 10426,  8867,  6967,  4799,  2446,
4520 ,  6270,  5906,  5315,  4520,  3552,  2446,  1247
```

merged table used in 4XM

```
16, 15, 13, 19, 24, 31, 28, 17,
17, 23, 25, 31, 36, 63, 45, 21,
18, 24, 27, 37, 52, 59, 49, 20,
16, 28, 34, 40, 60, 80, 51, 20,
18, 31, 48, 66, 68, 86, 56, 21,
19, 38, 56, 59, 64, 64, 48, 20,
27, 48, 55, 55, 56, 51, 35, 15,
20, 35, 34, 32, 31, 22, 15,  8,
```

This is simply the element-wise product of the quantization table and the AAN table divided by 2^{16}.

4XM's AAN IDCT uses (a*const)>>16 to approximate multiplications and simply shifts the transformed result 16 bits to the right. The scaled constants are:

exact	scaled constant
1.082392200...	70936
1.414213562...	92682
1.847759065...	121095
2.613125930...	171254

which are simply the exact constants multiplied by 2^{16} and rounded to the nearest integer.

see 4xm.c `http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/ffmpeg/ffmpeg/libavcodec/4xm.c?rev=HEAD&content-type=text/vnd.viewcvs-markup`

3.1.4 YCbCr 4:2:0 -> RGB565 colorspace transform

Chroma is first upsampled by sample replication / nearest neighbor scaling, so that the same Cb and Cr samples are used for each 2x2 Y samples

R= (Y + Cr + 128)>>3

G= (Y - ((Cb+Cr)> >1) + 128)> >2

B= (Y + 2Cb + 128)> >3

There is no check or protection against overflow, so values will wrap around if they are too large or small.

3.2 P-Frame

A P-frame picture is split into blocks which are coded left->right, top->bottom. Each block contains 8x8 samples in RGB565 format (5 bits for red, 6 bits for green, 5 bits for blue). Each block can be recursively split into 2, down to 2x1/1x2 sized blocks.

A P-frame block can be coded using 1 of 7 methods:

1. motion compensated with 1 vector
2. horizontally split in the middle
3. vertically split in the middle
4. skipped (block data copied from the frame before the last)
   Example: Intraframe, Interframe1, Interframe2, Interframe3
   a skipped block in Interframe3 will use the data from Interframe1; skipped blocks in Interframe1 are dissallowed as there is no source frame
5. motion vector + DC difference (the 16-bit words of the DC and the source block are simply added, there is no special handling of overflows)
6. DC only, the whole block is filled with the DC color
7. hardcoded pixel values (left->right, top->bottom)

Block splitting is only available if the resulting blocks are larger than 1x2/2x1. Hardcoded pixel values are only available for 1x2/2x1 sized blocks

Motion compensation assumes that the number of words (16-bit RGB565 pixels) per line is equal to the width, so that motion vectors which point right or left outside of the picture use the pixels from the other side. There is no subpel motion compensation or filtering which means that motion compensation can simply be done by copying the pixels from the motion block.

3.2.1 Motion Vector table

See mv[256][2] at 4xm.c `http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/ffmpeg/ffmpeg/libavcodec/4xm.c?rev=HEAD&content-type=text/vnd.viewcvs-markup`.

3.3 C-Frame

A C-frame is essentially a partial P frame. It has all the same coding options but a different header.

4 Bitstream

All 32-bit values are in little endian byte order.

4.1 I-Frame

32bit	'ifrm'
32bit	chunk length
32bit	0 (unknown)
32bit	bitstream size
n byte	bitstream
32bit	prefixstream size / 4
32bit	token_count
n byte	prefixstream

4.1.1 Prefix stream

```
start            8bit
end              8bit
do{
  for(i=start; i<=end; i++)
    frequency[i]  8bit
  start          8bit
  if(start==0) break;
  end            8bit
}
while(not 32bit aligned)
  0              8bit
for(i=0; i<token_count; i++)
  prefix[i]      prefix_vlc
256              prefix_vlc
```

Note: The prefix_vlc are stored so that each aligned 32-bit word is stored in byteswapped order. This byteswapping is not done to the bitstream, just the prefix stream

Frequency values which are not explicitly set are 0 except that frequency[256]=1. This is the "end of picture" code.

4.1.2 Macroblock

A macroblock bitstream is simply a bitstream of 6 blocks.

4.1.3 Block

```
dc_prefix        prefix_vlc         prefix stream
dc_suffix        dc_prefix bits     bitstream
i=1;
while(i<64){
  ac_prefix      prefix_vlc         prefix_stream
  if(ac_prefix == 0xF0)
    i+=16;
  else if(ac_prefix == 0x00)
    break;
  else{
    i+= ac_prefix>>4;
    level_prefix= ac_prefix&0xF;
    level_suffix level_prefix bits  bitstream
    block[ zigzag[i] ]= level;
    i++;
  }
}
```

4.2 P-Frame

32bit	'pfrm'
32bit	chunk size
32bit	0 (unknown)
32bit	unknown, perhaps a checksum
32bit	unknown
32bit	bitstream size
32bit	wordstream size

32bit bytestream size

n bytes bitstream, stored in byteswapped 32-bit words

n bytes RGB16 wordstream, stored in little endian order

n bytes bytestream

4.2.1 Block

```
block(){
  mode    vlc     bitstream
  if(mode==h_split || mode==v_split){
    block()
    block()
  }
  if(mode==mc || mode==mcdc)
    mv    8bit    bytestream
  if(mode==dc || mode==mcdc)
    dc    16bit   wordstream
  if(mode==esc){
    col1  16bit   wordstream
    col2  16bit   wordstream
  }
}
```

4.3 C-Frame

32bit 'cfrm'

32bit chunk size

32bit 0 (unknown)

32bit frame number / frame id, this is the frame number where the frame will be shown, it is also the frame number at which the last cframe part of this frame will be; note, all parts of the same cframe contain the same id here

32bit whole frame size

* p frame, this is (unk, unk, bitstream size, wordstream size, ...) for the first c frame chunk of a c frame

5 VLC Codes

5.1 prefix_vlc in I frames

The prefix_vlc table is generated from the frequencies stored in the prefix stream. Additionally, the element 256 is added with an implicit frequency of 1. For the exact algorithm see libavcodec/4xm.c read_huffman_tables().

5.2 level vlc in I Frames

Identical to JPEG

prefix	vlc	level
0		0
1	0/1	-1/1
2	0X/1X	-3..-2/2..3
3	0XX/1XX	-7..-4/4..7
...	...	...

One way to decode this is:

```
if(prefix){
  v= get_bits(prefix);
  if((v & (1<<(prefix-1))) == 0)
    v= (-1 <<prefix)|(v+1);
}else
  v= 0;
```

5.3 Block Mode Codes in P frames

For blocks 8x8, 8x4, 8x2, 4x8, 4x4, 4x2, 2x8, 2x4, 2x2

0	mc
10	h_split
110	v_split
1110	skip
11110	mcdc
11111	dc

For blocks 8x1, 4x1

0	mc
10	h_split
110	skip
1110	mcdc
1111	dc

For blocks 1x8, 1x4

0	mc
10	v_split
110	skip
1110	mcdc
1111	dc

For blocks 2x1, 1x2

0	mc
10	skip
110	mcdc
1110	dc
1111	esc

6 Applications and Platforms

The 4XM video codec is intended for gaming applications. It is known to operate on these computing platforms:

- PC/Microsoft Windows
- Apple Macintosh
- Sega Dreamcast
- Nintendo Gameboy Advance

While the Dreamcast and the targeted PC/Mac platforms have quite a bit of computing power (at least 200 MHz), the GBA has an ARM RISC CPU running at 16-17 MHz.

The 4XM coding method seems a little odd in its mixture of YCbCr and RGB colorspaces. In the end, all of the output data is RGB565. It is useful to note that many video game consoles can efficiently manuipulate this colorspace with video hardware. By contrast, many video consoles have no, or very limited, facilites for direct YCbCr rendering, particularly planar YCbCr modes.

One more note about interframe block addition: One possible approach to implementing this part of the method on console hardware, at least the Sega Dreamcast, would be to fill a texture with all zero values, skip all blocks that are not coded, and fill the coded blocks with the coded RGB565 difference. Then, the final texture could be added to the current frame. This also has the implicit side effect of saturating the addition so that the resulting pixels do not wrap around.

7 Changelog

0.01	2003-06-01 initial version by Michael Niedermayer
0.02	2003-06-07 minor changes
0.03	2003-06-08 peer review, grammar/spelling/punctuation fixes and "Applications and Platforms" section by Mike Melanson minor changes by Michael
0.04	2003-06-08 minor changes by Mike Melanson and Michael Niedermayer

8 Copyright