

## FFMPEG 实现 YUV, RGB各种图像原始数据之间的转换 (swscale)

2013年11月06日 15:37:09 阅读数：88034

FFMPEG中的 swscale提供了视频原始数据 (YUV420, YUV422, YUV444, RGB24...) 之间的转换, 分辨率变换等操作, 使用起来十分方便, 在这里记录一下它的用法。

swscale主要用于在2个AVFrame之间进行转换。

下面来看一个视频解码的简单例子, 这个程序完成了对“北京移动开发者大会茶歇视频2.flv” (其实就是优酷上的一个普通视频) 的解码工作, 并将解码后的数据保存为原始数据文件 (例如YUV420, YUV422, RGB24等等)。其中略去了很多的代码。

注：完整代码在文章：[100行代码实现最简单的基于FFMPEG+SDL的视频播放器](#)

```
[cpp]
1. //ffmpeg simple player
2. //
3. //媒资检索系统子系统
4. //
5. //2013 雷霄骅 leixiaohua1020@126.com
6. //中国传媒大学/数字电视技术
7. //
8. #include "stdafx.h"
9.
10. int _tmain(int argc, _TCHAR* argv[])
11. {
12.     AVFormatContext *pFormatCtx;
13.     int i, videoindex;
14.     AVCodecContext *pCodecCtx;
15.     AVCodec *pCodec;
16.     char filepath[]="北京移动开发者大会茶歇视频2.flv";
17.     av_register_all();
18.     avformat_network_init();
19.     pFormatCtx = avformat_alloc_context();
20.     if(avformat_open_input(&pFormatCtx, filepath, NULL, NULL) != 0){
21.         printf("无法打开文件\n");
22.         return -1;
23.     }
24.
25.     .....
26.
27.     AVFrame *pFrame, *pFrameYUV;
28.     pFrame=avcodec_alloc_frame();
29.     pFrameYUV=avcodec_alloc_frame();
30.     uint8_t *out_buffer;
31.
32.     out_buffer=new uint8_t[avpicture_get_size(PIX_FMT_RGB24, pCodecCtx->width, pCodecCtx->height)];
33.     avpicture_fill((AVPicture *)pFrameYUV, out_buffer, PIX_FMT_RGB24, pCodecCtx->width, pCodecCtx->height);
34.
35.     /*
36.     out_buffer=new uint8_t[avpicture_get_size(PIX_FMT_YUV420P, pCodecCtx->width, pCodecCtx->height)];
37.     avpicture_fill((AVPicture *)pFrameYUV, out_buffer, PIX_FMT_YUV420P, pCodecCtx->width, pCodecCtx->height);*/
38.
39.     /*
40.     out_buffer=new uint8_t[avpicture_get_size(PIX_FMT_UYVY422, pCodecCtx->width, pCodecCtx->height)];
41.     avpicture_fill((AVPicture *)pFrameYUV, out_buffer, PIX_FMT_UYVY422, pCodecCtx->width, pCodecCtx->height);
42.     out_buffer=new uint8_t[avpicture_get_size(PIX_FMT_YUV422P, pCodecCtx->width, pCodecCtx->height)];
43.     avpicture_fill((AVPicture *)pFrameYUV, out_buffer, PIX_FMT_YUV422P, pCodecCtx->width, pCodecCtx->height);*/
44.
45.     .....
46.
47.     FILE *output=fopen("out.rgb", "wb+");
48.     //-----
49.     while(av_read_frame(pFormatCtx, packet)>=0)
50.     {
51.         if(packet->stream_index==videoindex)
52.         {
53.             ret = avcodec_decode_video2(pCodecCtx, pFrame, &got_picture, packet);
54.
55.             if(ret < 0)
56.             {
57.                 printf("解码错误\n");
58.                 return -1;
59.             }
60.             if(got_picture)
61.             {
62.                 /*img_convert_ctx = sws_getContext(pCodecCtx->width, pCodecCtx->height, pCodecCtx->pix_fmt, pCodecCtx->width, pCodecCtx->height, PIX_FMT_UYVY422, SWS_BICUBIC, NULL, NULL, NULL);
63.                 sws_scale(img_convert_ctx, (const uint8_t* const*)pFrame->data, pFrame->linesize, 0, pCodecCtx->height, pFrameYUV->data, pFrameYUV->linesize);
64.                 img_convert_ctx = sws_getContext(pCodecCtx->width, pCodecCtx->height, pCodecCtx->pix_fmt, pCodecCtx->width, pCodecCtx->height, PIX_FMT_YUV422P, SWS_BICUBIC, NULL, NULL, NULL);
65.                 sws_scale(img_convert_ctx, (const uint8_t* const*)pFrame->data, pFrame->linesize, 0, pCodecCtx->height, pFrameYUV->data, pFrameYUV->linesize);*/
66.                 //转换
67.                 img_convert_ctx = sws_getContext(pCodecCtx->width, pCodecCtx->height, pCodecCtx->pix_fmt, pCodecCtx->width, pCodecCtx->height, PIX_FMT_RGB24, SWS_BICUBIC, NULL, NULL, NULL);
68.                 sws_scale(img_convert_ctx, (const uint8_t* const*)pFrame->data, pFrame->linesize, 0, pCodecCtx->height, pFrameYUV->data, pFrameYUV->linesize);
```

```

V->data, pFrameYUV->linesize);
68.
69.
70.         //RGB
71.         fwrite(pFrameYUV->data[0], (pCodecCtx->width)*(pCodecCtx->height)*3, 1, output);
72.         /*
73.         //UYVY
74.         fwrite(pFrameYUV->data[0], (pCodecCtx->width)*(pCodecCtx->height), 2, output);
75.         //YUV420P
76.         fwrite(pFrameYUV->data[0], (pCodecCtx->width)*(pCodecCtx->height), 1, output);
77.         fwrite(pFrameYUV->data[1], (pCodecCtx->width)*(pCodecCtx->height)/4, 1, output);
78.         fwrite(pFrameYUV->data[2], (pCodecCtx->width)*(pCodecCtx->height)/4, 1, output);
79.         */
80.         .....
81.
82.     }
83. }
84.     av_free_packet(packet);
85. }
86.
87.     fclose(output);
88.
89.     .....
90.
91.     return 0;
92. }

```

从代码中可以看出，解码后的视频帧数据保存在pFrame变量中，然后经过swscale函数转换后，将视频帧数据保存在pFrameYUV变量中。最后将pFrameYUV中的数据写入成文件。

在本代码中，将数据保存成了RGB24的格式。如果想保存成其他格式，比如YUV420，YUV422等，需要做2个步骤：

### 1.初始化pFrameYUV的时候，设定想要转换的格式：

```

1.  AVFrame *pFrame,*pFrameYUV;
2.  pFrame=avcodec_alloc_frame();
3.  pFrameYUV=avcodec_alloc_frame();
4.  uint8_t *out_buffer;
5.
6.  out_buffer=new uint8_t[avpicture_get_size(PIX_FMT_RGB24, pCodecCtx->width, pCodecCtx->height)];
7.  avpicture_fill((AVPicture *)pFrameYUV, out_buffer, PIX_FMT_RGB24, pCodecCtx->width, pCodecCtx->height);

```

只需要把PIX\_FMT\_\*\*\*改了就可以了

### 2.在sws\_getContext()中更改想要转换的格式：

```

1.  img_convert_ctx = sws_getContext(pCodecCtx->width, pCodecCtx->height, pCodecCtx->pix_fmt, pCodecCtx->width, pCodecCtx->height, PIX_FMT_RGB24, SWS_BICUBIC, NULL, NULL, NULL);

```

也是把PIX\_FMT\_\*\*\*改了就可以了

最后，如果想将转换后的原始数据存成文件，只需要将pFrameYUV的data指针指向的数据写入文件就可以了。

例如，保存YUV420P格式的数据，用以下代码：

```

1.  fwrite(pFrameYUV->data[0], (pCodecCtx->width)*(pCodecCtx->height), 1, output);
2.  fwrite(pFrameYUV->data[1], (pCodecCtx->width)*(pCodecCtx->height)/4, 1, output);
3.  fwrite(pFrameYUV->data[2], (pCodecCtx->width)*(pCodecCtx->height)/4, 1, output);

```

保存RGB24格式的数据，用以下代码：

```

1.  fwrite(pFrameYUV->data[0], (pCodecCtx->width)*(pCodecCtx->height)*3, 1, output);

```

保存UYVY格式的数据，用以下代码：

```

1.  fwrite(pFrameYUV->data[0], (pCodecCtx->width)*(pCodecCtx->height), 2, output);

```

在这里又有一个问题，YUV420P格式需要写入data[0]，data[1]，data[2]；而RGB24，UYVY格式却仅仅是写入data[0]，他们的区别到底是什么呢？经过研究发现，在FFMPEG中，图像原始数据包括两种：planar和packed。planar就是将几个分量分开存，比如YUV420中，data[0]专门存Y，data[1]专门存U，data[2]专门存V。而packed则是打包存，所有数据都存在data[0]中。

具体哪个格式是planar，哪个格式是packed，可以查看pixfmt.h文件。注：有些格式名称后面是LE或BE，分别对应little-endian或big-endian。另外名字后面有P的是planar格式。

```

1.  /*  雷霄骅
2.  *  中国传媒大学/数字电视技术
3.  *  leixiaohua1020@126.com
4.  *
5.  */
6.  /*
7.  *  copyright (c) 2006 Michael Niedermayer <michaelni@gmx.at>
8.  *
9.  *  This file is part of Ffmpeg.
10. *
11. *  Ffmpeg is free software; you can redistribute it and/or
12. *  modify it under the terms of the GNU Lesser General Public
13. *  License as published by the Free Software Foundation; either
14. *  version 2.1 of the License, or (at your option) any later version.
15. *
16. *  Ffmpeg is distributed in the hope that it will be useful,
17. *  but WITHOUT ANY WARRANTY; without even the implied warranty of
18. *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19. *  Lesser General Public License for more details.
20. *
21. *  You should have received a copy of the GNU Lesser General Public
22. *  License along with Ffmpeg; if not, write to the Free Software
23. *  Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
24. */
25.
26. #ifndef AVUTIL_PIXFMT_H
27. #define AVUTIL_PIXFMT_H
28.
29. /**
30.  * @file
31.  * pixel format definitions
32.  *
33.  */
34.
35. #include "libavutil/avconfig.h"
36.
37. /**
38.  * Pixel format.
39.  *
40.  * @note
41.  * PIX_FMT_RGB32 is handled in an endian-specific manner. An RGBA
42.  * color is put together as:
43.  * (A << 24) | (R << 16) | (G << 8) | B
44.  * This is stored as BGRA on little-endian CPU architectures and ARGB on
45.  * big-endian CPUs.
46.  *
47.  * @par
48.  * When the pixel format is palettized RGB (PIX_FMT_PALETTE8), the palettized
49.  * image data is stored in AVFrame.data[0]. The palette is transported in
50.  * AVFrame.data[1], is 1024 bytes long (256 4-byte entries) and is
51.  * formatted the same as in PIX_FMT_RGB32 described above (i.e., it is
52.  * also endian-specific). Note also that the individual RGB palette
53.  * components stored in AVFrame.data[1] should be in the range 0..255.
54.  * This is important as many custom PAL8 video codecs that were designed
55.  * to run on the IBM VGA graphics adapter use 6-bit palette components.
56.  *
57.  * @par
58.  * For all the 8bit per pixel formats, an RGB32 palette is in data[1] like
59.  * for pal8. This palette is filled in automatically by the function
60.  * allocating the picture.
61.  *
62.  * @note
63.  * make sure that all newly added big endian formats have pix_fmt&1==1
64.  * and that all newly added little endian formats have pix_fmt&1==0
65.  * this allows simpler detection of big vs little endian.
66.  */
67. enum PixelFormat {
68.     PIX_FMT_NONE= -1,
69.     PIX_FMT_YUV420P,    ///< planar YUV 4:2:0, 12bpp, (1 Cr & Cb sample per 2x2 Y samples)
70.     PIX_FMT_YUV422P,    ///< packed YUV 4:2:2, 16bpp, Y0 Cb Y1 Cr
71.     PIX_FMT_RGB24,      ///< packed RGB 8:8:8, 24bpp, RGBRGB...
72.     PIX_FMT_BGR24,      ///< packed RGB 8:8:8, 24bpp, BGRBGR...
73.     PIX_FMT_YUV422P,    ///< planar YUV 4:2:2, 16bpp, (1 Cr & Cb sample per 2x1 Y samples)
74.     PIX_FMT_YUV444P,    ///< planar YUV 4:4:4, 24bpp, (1 Cr & Cb sample per 1x1 Y samples)
75.     PIX_FMT_YUV410P,    ///< planar YUV 4:1:0, 9bpp, (1 Cr & Cb sample per 4x4 Y samples)
76.     PIX_FMT_YUV411P,    ///< planar YUV 4:1:1, 12bpp, (1 Cr & Cb sample per 4x1 Y samples)
77.     PIX_FMT_GRAY8,      ///<          Y          , 8bpp
78.     PIX_FMT_MONOWHITE,  ///<          Y          , 1bpp, 0 is white, 1 is black, in each byte pixels are ordered from the msb to the lsb
79.
80.     PIX_FMT_MONOBLACK,  ///<          Y          , 1bpp, 0 is black, 1 is white, in each byte pixels are ordered from the msb to the lsb
81.
82.     PIX_FMT_PALETTE8,   ///< 8 bit with PIX_FMT_RGB32 palette
83.     PIX_FMT_YUVJ420P,   ///< planar YUV 4:2:0, 12bpp, full scale (JPEG), deprecated in favor of PIX_FMT_YUV420P and setting color_range
84.
85.     PIX_FMT_YUVJ422P,   ///< planar YUV 4:2:2, 16bpp, full scale (JPEG), deprecated in favor of PIX_FMT_YUV422P and setting color_range
86.
87.     PIX_FMT_YUVJ444P,   ///< planar YUV 4:4:4, 24bpp, full scale (JPEG), deprecated in favor of PIX_FMT_YUV444P and setting color_range
88.
89.     PIX_FMT_MJPEG,      ///< MJPEG image in plane 0
90.     PIX_FMT_JPEG,       ///< JPEG image in plane 0
91.     PIX_FMT_H264,       ///< H.264 image in plane 0
92.     PIX_FMT_H264_MV,    ///< H.264 image in plane 0
93.     PIX_FMT_H263,       ///< H.263 image in plane 0
94.     PIX_FMT_H263_MV,    ///< H.263 image in plane 0
95.     PIX_FMT_H261,       ///< H.261 image in plane 0
96.     PIX_FMT_H261_MV,    ///< H.261 image in plane 0
97.     PIX_FMT_H264_I,     ///< H.264 image in plane 0
98.     PIX_FMT_H264_MV_I,  ///< H.264 image in plane 0
99.     PIX_FMT_H263_I,     ///< H.263 image in plane 0
100.    PIX_FMT_H263_MV_I,   ///< H.263 image in plane 0
101.    PIX_FMT_H261_I,     ///< H.261 image in plane 0
102.    PIX_FMT_H261_MV_I,   ///< H.261 image in plane 0
103.    PIX_FMT_H264_I16,    ///< H.264 image in plane 0
104.    PIX_FMT_H264_MV_I16, ///< H.264 image in plane 0
105.    PIX_FMT_H263_I16,    ///< H.263 image in plane 0
106.    PIX_FMT_H263_MV_I16, ///< H.263 image in plane 0
107.    PIX_FMT_H261_I16,    ///< H.261 image in plane 0
108.    PIX_FMT_H261_MV_I16, ///< H.261 image in plane 0
109.    PIX_FMT_H264_I422,   ///< H.264 image in plane 0
110.    PIX_FMT_H264_MV_I422, ///< H.264 image in plane 0
111.    PIX_FMT_H263_I422,   ///< H.263 image in plane 0
112.    PIX_FMT_H263_MV_I422, ///< H.263 image in plane 0
113.    PIX_FMT_H261_I422,   ///< H.261 image in plane 0
114.    PIX_FMT_H261_MV_I422, ///< H.261 image in plane 0
115.    PIX_FMT_H264_I420,   ///< H.264 image in plane 0
116.    PIX_FMT_H264_MV_I420, ///< H.264 image in plane 0
117.    PIX_FMT_H263_I420,   ///< H.263 image in plane 0
118.    PIX_FMT_H263_MV_I420, ///< H.263 image in plane 0
119.    PIX_FMT_H261_I420,   ///< H.261 image in plane 0
120.    PIX_FMT_H261_MV_I420, ///< H.261 image in plane 0
121.    PIX_FMT_H264_I420_16, ///< H.264 image in plane 0
122.    PIX_FMT_H264_MV_I420_16, ///< H.264 image in plane 0
123.    PIX_FMT_H263_I420_16, ///< H.263 image in plane 0
124.    PIX_FMT_H263_MV_I420_16, ///< H.263 image in plane 0
125.    PIX_FMT_H261_I420_16, ///< H.261 image in plane 0
126.    PIX_FMT_H261_MV_I420_16, ///< H.261 image in plane 0
127.    PIX_FMT_H264_I420_16_1, ///< H.264 image in plane 0
128.    PIX_FMT_H264_MV_I420_16_1, ///< H.264 image in plane 0
129.    PIX_FMT_H263_I420_16_1, ///< H.263 image in plane 0
130.    PIX_FMT_H263_MV_I420_16_1, ///< H.263 image in plane 0
131.    PIX_FMT_H261_I420_16_1, ///< H.261 image in plane 0
132.    PIX_FMT_H261_MV_I420_16_1, ///< H.261 image in plane 0
133.    PIX_FMT_H264_I420_16_1_1, ///< H.264 image in plane 0
134.    PIX_FMT_H264_MV_I420_16_1_1, ///< H.264 image in plane 0
135.    PIX_FMT_H263_I420_16_1_1, ///< H.263 image in plane 0
136.    PIX_FMT_H263_MV_I420_16_1_1, ///< H.263 image in plane 0
137.    PIX_FMT_H261_I420_16_1_1, ///< H.261 image in plane 0
138.    PIX_FMT_H261_MV_I420_16_1_1, ///< H.261 image in plane 0
139.    PIX_FMT_H264_I420_16_1_1_1, ///< H.264 image in plane 0
140.    PIX_FMT_H264_MV_I420_16_1_1_1, ///< H.264 image in plane 0
141.    PIX_FMT_H263_I420_16_1_1_1, ///< H.263 image in plane 0
142.    PIX_FMT_H263_MV_I420_16_1_1_1, ///< H.263 image in plane 0
143.    PIX_FMT_H261_I420_16_1_1_1, ///< H.261 image in plane 0
144.    PIX_FMT_H261_MV_I420_16_1_1_1, ///< H.261 image in plane 0
145.    PIX_FMT_H264_I420_16_1_1_1_1, ///< H.264 image in plane 0
146.    PIX_FMT_H264_MV_I420_16_1_1_1_1, ///< H.264 image in plane 0
147.    PIX_FMT_H263_I420_16_1_1_1_1, ///< H.263 image in plane 0
148.    PIX_FMT_H263_MV_I420_16_1_1_1_1, ///< H.263 image in plane 0
149.    PIX_FMT_H261_I420_16_1_1_1_1, ///< H.261 image in plane 0
150.    PIX_FMT_H261_MV_I420_16_1_1_1_1, ///< H.261 image in plane 0
151.    PIX_FMT_H264_I420_16_1_1_1_1_1, ///< H.264 image in plane 0
152.    PIX_FMT_H264_MV_I420_16_1_1_1_1_1, ///< H.264 image in plane 0
153.    PIX_FMT_H263_I420_16_1_1_1_1_1, ///< H.263 image in plane 0
154.    PIX_FMT_H263_MV_I420_16_1_1_1_1_1, ///< H.263 image in plane 0
155.    PIX_FMT_H261_I420_16_1_1_1_1_1, ///< H.261 image in plane 0
156.    PIX_FMT_H261_MV_I420_16_1_1_1_1_1, ///< H.261 image in plane 0
157.    PIX_FMT_H264_I420_16_1_1_1_1_1_1, ///< H.264 image in plane 0
158.    PIX_FMT_H264_MV_I420_16_1_1_1_1_1_1, ///< H.264 image in plane 0
159.    PIX_FMT_H263_I420_16_1_1_1_1_1_1, ///< H.263 image in plane 0
160.    PIX_FMT_H263_MV_I420_16_1_1_1_1_1_1, ///< H.263 image in plane 0
161.    PIX_FMT_H261_I420_16_1_1_1_1_1_1, ///< H.261 image in plane 0
162.    PIX_FMT_H261_MV_I420_16_1_1_1_1_1_1, ///< H.261 image in plane 0
163.    PIX_FMT_H264_I420_16_1_1_1_1_1_1_1, ///<
```

83.	PIX_FMT_YUVJ444P, ///< planar YUV 4:4:4, 24bpp, full scale (JPEG), deprecated in favor of PIX_FMT_YUV444P and setting color_range
84.	PIX_FMT_XVMC_MPEG2_MC, ///< XVideo Motion Acceleration via common packet passing
85.	PIX_FMT_XVMC_MPEG2_IDCT,
86.	PIX_FMT_UYVY422, ///< packed YUV 4:2:2, 16bpp, Cb Y0 Cr Y1
87.	PIX_FMT_UYVYY411, ///< packed YUV 4:1:1, 12bpp, Cb Y0 Y1 Cr Y2 Y3
88.	PIX_FMT_BGR8, ///< packed RGB 3:3:2, 8bpp, (msb)2B 3G 3R(lsb)
89.	PIX_FMT_BGR4, ///< packed RGB 1:2:1 bitstream, 4bpp, (msb)1B 2G 1R(lsb), a byte contains two pixels, the first pixel in the
90.	PIX_FMT_BGR4_BYTE, ///< packed RGB 1:2:1, 8bpp, (msb)1B 2G 1R(lsb)
91.	PIX_FMT_RGB8, ///< packed RGB 3:3:2, 8bpp, (msb)2R 3G 3B(lsb)
92.	PIX_FMT_RGB4, ///< packed RGB 1:2:1 bitstream, 4bpp, (msb)1R 2G 1B(lsb), a byte contains two pixels, the first pixel in the
93.	PIX_FMT_RGB4_BYTE, ///< packed RGB 1:2:1, 8bpp, (msb)1R 2G 1B(lsb)
94.	PIX_FMT_NV12, ///< planar YUV 4:2:0, 12bpp, 1 plane for Y and 1 plane for the UV components, which are interleaved (first by
95.	PIX_FMT_NV21, ///< as above, but U and V bytes are swapped
96.	
97.	PIX_FMT_ARGB, ///< packed ARGB 8:8:8:8, 32bpp, ARGBARGB...
98.	PIX_FMT_RGBA, ///< packed RGBA 8:8:8:8, 32bpp, RGBARGBA...
99.	PIX_FMT_ABGR, ///< packed ABGR 8:8:8:8, 32bpp, ABGRABGR...
100.	PIX_FMT_BGRA, ///< packed BGRA 8:8:8:8, 32bpp, BGRABGRA...
101.	
102.	PIX_FMT_GRAY16BE, ///< Y, 16bpp, big-endian
103.	PIX_FMT_GRAY16LE, ///< Y, 16bpp, little-endian
104.	PIX_FMT_YUV440P, ///< planar YUV 4:4:0 (1 Cr & Cb sample per 1x2 Y samples)
105.	PIX_FMT_YUVJ440P, ///< planar YUV 4:4:0 full scale (JPEG), deprecated in favor of PIX_FMT_YUV440P and setting color_range
106.	PIX_FMT_YUV420P, ///< planar YUV 4:2:0, 20bpp, (1 Cr & Cb sample per 2x2 Y & A samples)
107.	PIX_FMT_VDPAU_H264, ///< H.264 HW decoding with VDPAU, data[0] contains a vdpau_render_state struct which contains the bitstream of
108.	PIX_FMT_VDPAU_MPEG1, ///< MPEG-1 HW decoding with VDPAU, data[0] contains a vdpau_render_state struct which contains the bitstream of the slices as well as various
109.	PIX_FMT_VDPAU_MPEG2, ///< MPEG-2 HW decoding with VDPAU, data[0] contains a vdpau_render_state struct which contains the bitstream of the slices as well as various
110.	PIX_FMT_VDPAU_WMV3, ///< WMV3 HW decoding with VDPAU, data[0] contains a vdpau_render_state struct which contains the bitstream of the
111.	PIX_FMT_VDPAU_VC1, ///< VC-1 HW decoding with VDPAU, data[0] contains a vdpau_render_state struct which contains the bitstream of the slices as well as various
112.	PIX_FMT_RGB48BE, ///< packed RGB 16:16:16, 48bpp, 16R, 16G, 16B, the 2-byte value for each R/G/B component is stored as big-endian
113.	PIX_FMT_RGB48LE, ///< packed RGB 16:16:16, 48bpp, 16R, 16G, 16B, the 2-byte value for each R/G/B component is stored as little-endian
114.	
115.	PIX_FMT_RGB565BE, ///< packed RGB 5:6:5, 16bpp, (msb) 5R 6G 5B(lsb), big-endian
116.	PIX_FMT_RGB565LE, ///< packed RGB 5:6:5, 16bpp, (msb) 5R 6G 5B(lsb), little-endian
117.	PIX_FMT_RGB555BE, ///< packed RGB 5:5:5, 16bpp, (msb)1A 5R 5G 5B(lsb), big-endian, most significant bit to 0
118.	PIX_FMT_RGB555LE, ///< packed RGB 5:5:5, 16bpp, (msb)1A 5R 5G 5B(lsb), little-endian, most significant bit to 0
119.	
120.	PIX_FMT_BGR565BE, ///< packed BGR 5:6:5, 16bpp, (msb) 5B 6G 5R(lsb), big-endian
121.	PIX_FMT_BGR565LE, ///< packed BGR 5:6:5, 16bpp, (msb) 5B 6G 5R(lsb), little-endian
122.	PIX_FMT_BGR555BE, ///< packed BGR 5:5:5, 16bpp, (msb)1A 5B 5G 5R(lsb), big-endian, most significant bit to 1
123.	PIX_FMT_BGR555LE, ///< packed BGR 5:5:5, 16bpp, (msb)1A 5B 5G 5R(lsb), little-endian, most significant bit to 1
124.	
125.	PIX_FMT_VAAPI_MOCO, ///< HW acceleration through VA API at motion compensation entry-point, Picture.data[3] contains a vaapi_render_state struct which contains macroblocks as well as various fields extracted from headers
126.	PIX_FMT_VAAPI_IDCT, ///< HW acceleration through VA API at IDCT entry-point, Picture.data[3] contains a vaapi_render_state struct which contains fields extracted from headers
127.	PIX_FMT_VAAPI_VLD, ///< HW decoding through VA API, Picture.data[3] contains a vaapi_render_state struct which contains the bits of the slices as well as various fields extracted from headers
128.	
129.	PIX_FMT_YUV420P16LE, ///< planar YUV 4:2:0, 24bpp, (1 Cr & Cb sample per 2x2 Y samples), little-endian
130.	PIX_FMT_YUV420P16BE, ///< planar YUV 4:2:0, 24bpp, (1 Cr & Cb sample per 2x2 Y samples), big-endian
131.	PIX_FMT_YUV422P16LE, ///< planar YUV 4:2:2, 32bpp, (1 Cr & Cb sample per 2x1 Y samples), little-endian
132.	PIX_FMT_YUV422P16BE, ///< planar YUV 4:2:2, 32bpp, (1 Cr & Cb sample per 2x1 Y samples), big-endian
133.	PIX_FMT_YUV444P16LE, ///< planar YUV 4:4:4, 48bpp, (1 Cr & Cb sample per 1x1 Y samples), little-endian
134.	PIX_FMT_YUV444P16BE, ///< planar YUV 4:4:4, 48bpp, (1 Cr & Cb sample per 1x1 Y samples), big-endian
135.	PIX_FMT_VDPAU_MPEG4, ///< MPEG4 HW decoding with VDPAU, data[0] contains a vdpau_render_state struct which contains the bitstream of the slices as well as various fields extracted from headers
136.	PIX_FMT_DXVA2_VLD, ///< HW decoding through DXVA2, Picture.data[3] contains a LPDIRECT3DSURFACE9 pointer
137.	
138.	PIX_FMT_RGB444LE, ///< packed RGB 4:4:4, 16bpp, (msb)4A 4R 4G 4B(lsb), little-endian, most significant bits to 0
139.	PIX_FMT_RGB444BE, ///< packed RGB 4:4:4, 16bpp, (msb)4A 4R 4G 4B(lsb), big-endian, most significant bits to 0
140.	PIX_FMT_BGR444LE, ///< packed BGR 4:4:4, 16bpp, (msb)4A 4B 4G 4R(lsb), little-endian, most significant bits to 1
141.	PIX_FMT_BGR444BE, ///< packed BGR 4:4:4, 16bpp, (msb)4A 4B 4G 4R(lsb), big-endian, most significant bits to 1
142.	PIX_FMT_GRAY8A, ///< 8bit gray, 8bit alpha
143.	PIX_FMT_BGR48BE, ///< packed RGB 16:16:16, 48bpp, 16B, 16G, 16R, the 2-byte value for each R/G/B component is stored as big-endian
144.	PIX_FMT_BGR48LE, ///< packed RGB 16:16:16, 48bpp, 16B, 16G, 16R, the 2-byte value for each R/G/B component is stored as little-endian
145.	
146.	//the following 10 formats have the disadvantage of needing 1 format for each bit depth, thus
147.	//If you want to support multiple bit depths, then using PIX_FMT_YUV420P16* with the bpp stored separately
148.	//is better
149.	PIX_FMT_YUV420P9BE, ///< planar YUV 4:2:0, 13.5bpp, (1 Cr & Cb sample per 2x2 Y samples), big-endian
150.	PIX_FMT_YUV420P9LE, ///< planar YUV 4:2:0, 13.5bpp, (1 Cr & Cb sample per 2x2 Y samples), little-endian
151.	PIX_FMT_YUV420P10BE, ///< planar YUV 4:2:0, 15bpp, (1 Cr & Cb sample per 2x2 Y samples), big-endian
152.	PIX_FMT_YUV420P10LE, ///< planar YUV 4:2:0, 15bpp, (1 Cr & Cb sample per 2x2 Y samples), little-endian

```

153.     PIX_FMT_YUV422P10BE, ///< planar YUV 4:2:2, 20bpp, (1 Cr & Cb sample per 2x1 Y samples), big-endian
154.     PIX_FMT_YUV422P10LE, ///< planar YUV 4:2:2, 20bpp, (1 Cr & Cb sample per 2x1 Y samples), little-endian
155.     PIX_FMT_YUV444P9BE,  ///< planar YUV 4:4:4, 27bpp, (1 Cr & Cb sample per 1x1 Y samples), big-endian
156.     PIX_FMT_YUV444P9LE,  ///< planar YUV 4:4:4, 27bpp, (1 Cr & Cb sample per 1x1 Y samples), little-endian
157.     PIX_FMT_YUV444P10BE, ///< planar YUV 4:4:4, 30bpp, (1 Cr & Cb sample per 1x1 Y samples), big-endian
158.     PIX_FMT_YUV444P10LE, ///< planar YUV 4:4:4, 30bpp, (1 Cr & Cb sample per 1x1 Y samples), little-endian
159.     PIX_FMT_YUV422P9BE,  ///< planar YUV 4:2:2, 18bpp, (1 Cr & Cb sample per 2x1 Y samples), big-endian
160.     PIX_FMT_YUV422P9LE,  ///< planar YUV 4:2:2, 18bpp, (1 Cr & Cb sample per 2x1 Y samples), little-endian
161.     PIX_FMT_VDA_VLD,     ///< hardware decoding through VDA
162.
163. #ifdef AV_PIX_FMT_ABI_GIT_MASTER
164.     PIX_FMT_RGBA64BE,    ///< packed RGBA 16:16:16:16, 64bpp, 16R, 16G, 16B, 16A, the 2-
byte value for each R/G/B/A component is stored as big-endian
165.     PIX_FMT_RGBA64LE,    ///< packed RGBA 16:16:16:16, 64bpp, 16R, 16G, 16B, 16A, the 2-
byte value for each R/G/B/A component is stored as little-endian
166.     PIX_FMT_BGRA64BE,    ///< packed RGBA 16:16:16:16, 64bpp, 16B, 16G, 16R, 16A, the 2-
byte value for each R/G/B/A component is stored as big-endian
167.     PIX_FMT_BGRA64LE,    ///< packed RGBA 16:16:16:16, 64bpp, 16B, 16G, 16R, 16A, the 2-
byte value for each R/G/B/A component is stored as little-endian
168. #endif
169.     PIX_FMT_GBRP,        ///< planar GBR 4:4:4 24bpp
170.     PIX_FMT_GBRP9BE,     ///< planar GBR 4:4:4 27bpp, big endian
171.     PIX_FMT_GBRP9LE,     ///< planar GBR 4:4:4 27bpp, little endian
172.     PIX_FMT_GBRP10BE,    ///< planar GBR 4:4:4 30bpp, big endian
173.     PIX_FMT_GBRP10LE,    ///< planar GBR 4:4:4 30bpp, little endian
174.     PIX_FMT_GBRP16BE,    ///< planar GBR 4:4:4 48bpp, big endian
175.     PIX_FMT_GBRP16LE,    ///< planar GBR 4:4:4 48bpp, little endian
176.
177. #ifndef AV_PIX_FMT_ABI_GIT_MASTER
178.     PIX_FMT_RGBA64BE=0x123, ///< packed RGBA 16:16:16:16, 64bpp, 16R, 16G, 16B, 16A, the 2-
byte value for each R/G/B/A component is stored as big-endian
179.     PIX_FMT_RGBA64LE,    ///< packed RGBA 16:16:16:16, 64bpp, 16R, 16G, 16B, 16A, the 2-
byte value for each R/G/B/A component is stored as little-endian
180.     PIX_FMT_BGRA64BE,    ///< packed RGBA 16:16:16:16, 64bpp, 16B, 16G, 16R, 16A, the 2-
byte value for each R/G/B/A component is stored as big-endian
181.     PIX_FMT_BGRA64LE,    ///< packed RGBA 16:16:16:16, 64bpp, 16B, 16G, 16R, 16A, the 2-
byte value for each R/G/B/A component is stored as little-endian
182. #endif
183.     PIX_FMT_0RGB=0x123+4,    ///< packed RGB 8:8:8, 32bpp, 0RGB0RGB...
184.     PIX_FMT_RGB0,           ///< packed RGB 8:8:8, 32bpp, RGB0RGB0...
185.     PIX_FMT_0BGR,          ///< packed BGR 8:8:8, 32bpp, 0BGR0BGR...
186.     PIX_FMT_BGR0,           ///< packed BGR 8:8:8, 32bpp, BGR0BGR0...
187.     PIX_FMT_YUVA444P,      ///< planar YUV 4:4:4 32bpp, (1 Cr & Cb sample per 1x1 Y & A samples)
188.
189.     PIX_FMT_NB,            ///< number of pixel formats, DO NOT USE THIS if you want to link with shared libav* because the number of for
ts might differ between versions
190. };
191.
192. #define PIX_FMT_Y400A PIX_FMT_GRAY8A
193. #define PIX_FMT_GBR24P PIX_FMT_GBRP
194.
195. #if AV_HAVE_BIGENDIAN
196. #   define PIX_FMT_NE(be, le) PIX_FMT_##be
197. #else
198. #   define PIX_FMT_NE(be, le) PIX_FMT_##le
199. #endif
200.
201. #define PIX_FMT_RGB32    PIX_FMT_NE(ARGB, BGRA)
202. #define PIX_FMT_RGB32_1 PIX_FMT_NE(RGBA, ABGR)
203. #define PIX_FMT_BGR32    PIX_FMT_NE(ABGR, RGBA)
204. #define PIX_FMT_BGR32_1 PIX_FMT_NE(BGRA, ARGB)
205. #define PIX_FMT_0RGB32   PIX_FMT_NE(0RGB, BGR0)
206. #define PIX_FMT_0BGR32   PIX_FMT_NE(0BGR, RGB0)
207.
208. #define PIX_FMT_GRAY16   PIX_FMT_NE(GRAY16BE, GRAY16LE)
209. #define PIX_FMT_RGB48     PIX_FMT_NE(RGB48BE,  RGB48LE)
210. #define PIX_FMT_RGB565    PIX_FMT_NE(RGB565BE, RGB565LE)
211. #define PIX_FMT_RGB555    PIX_FMT_NE(RGB555BE, RGB555LE)
212. #define PIX_FMT_RGB444    PIX_FMT_NE(RGB444BE, RGB444LE)
213. #define PIX_FMT_BGR48     PIX_FMT_NE(BGR48BE,  BGR48LE)
214. #define PIX_FMT_BGR565    PIX_FMT_NE(BGR565BE, BGR565LE)
215. #define PIX_FMT_BGR555    PIX_FMT_NE(BGR555BE, BGR555LE)
216. #define PIX_FMT_BGR444    PIX_FMT_NE(BGR444BE, BGR444LE)
217.
218. #define PIX_FMT_YUV420P9   PIX_FMT_NE(YUV420P9BE , YUV420P9LE)
219. #define PIX_FMT_YUV422P9   PIX_FMT_NE(YUV422P9BE , YUV422P9LE)
220. #define PIX_FMT_YUV444P9   PIX_FMT_NE(YUV444P9BE , YUV444P9LE)
221. #define PIX_FMT_YUV420P10  PIX_FMT_NE(YUV420P10BE, YUV420P10LE)
222. #define PIX_FMT_YUV422P10  PIX_FMT_NE(YUV422P10BE, YUV422P10LE)
223. #define PIX_FMT_YUV444P10  PIX_FMT_NE(YUV444P10BE, YUV444P10LE)
224. #define PIX_FMT_YUV420P16  PIX_FMT_NE(YUV420P16BE, YUV420P16LE)
225. #define PIX_FMT_YUV422P16  PIX_FMT_NE(YUV422P16BE, YUV422P16LE)
226. #define PIX_FMT_YUV444P16  PIX_FMT_NE(YUV444P16BE, YUV444P16LE)
227.
228. #define PIX_FMT_RGBA64     PIX_FMT_NE(RGBA64BE, RGBA64LE)
229. #define PIX_FMT_BGRA64     PIX_FMT_NE(BGRA64BE, BGRA64LE)
230. #define PIX_FMT_GBRP9      PIX_FMT_NE(GBRP9BE ,   GBRP9LE)
231. #define PIX_FMT_GBRP10     PIX_FMT_NE(GBRP10BE,   GBRP10LE)
232. #define PIX_FMT_GBRP16     PIX_FMT_NE(GBRP16BE,   GBRP16LE)
233.
234. #endif /* AVUTIL_PIXFMT_H */

```



版权声明：本文为博主原创文章，未经博主允许不得转载。<https://blog.csdn.net/leixiaohua1020/article/details/14215391>

文章标签：[FFMPEG](#) [YUV](#) [RGB](#) [转换](#) [视频](#)

个人分类：[FFMPEG](#)

所属专栏：[FFmpeg](#)

此PDF由[spygg](#)生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com