原 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

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]

```cpp
//ffmpeg simple player
//
//媒资检索系统子系统
//
//2013 雷霄骅 leixiaohua1020@126.com
//中国传媒大学/数字电视技术
//
#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[])
{
    AVFormatContext *pFormatCtx;
    int             i, videoindex;
    AVCodecContext  *pCodecCtx;
    AVCodec         *pCodec;
    char filepath[]="北京移动开发者大会茶歇视频2.flv";
    av_register_all();
    avformat_network_init();
    pFormatCtx = avformat_alloc_context();
    if(avformat_open_input(&pFormatCtx,filepath,NULL,NULL)!=0){
        printf("无法打开文件\n");
        return -1;
    }

    ......

        AVFrame *pFrame,*pFrameYUV;
        pFrame=avcodec_alloc_frame();
        pFrameYUV=avcodec_alloc_frame();
        uint8_t *out_buffer;

        out_buffer=new uint8_t[avpicture_get_size(PIX_FMT_RGB24, pCodecCtx->width, pCodecCtx->height)];
        avpicture_fill((AVPicture *)pFrameYUV, out_buffer, PIX_FMT_RGB24, pCodecCtx->width, pCodecCtx->height);
/*
        out_buffer=new uint8_t[avpicture_get_size(PIX_FMT_YUV420P, pCodecCtx->width, pCodecCtx->height)];
        avpicture_fill((AVPicture *)pFrameYUV, out_buffer, PIX_FMT_YUV420P, pCodecCtx->width, pCodecCtx->height);*/

/*
        out_buffer=new uint8_t[avpicture_get_size(PIX_FMT_UYVY422, pCodecCtx->width, pCodecCtx->height)];
        avpicture_fill((AVPicture *)pFrameYUV, out_buffer, PIX_FMT_UYVY422, pCodecCtx->width, pCodecCtx->height);
        out_buffer=new uint8_t[avpicture_get_size(PIX_FMT_YUV422P, pCodecCtx->width, pCodecCtx->height)];
        avpicture_fill((AVPicture *)pFrameYUV, out_buffer, PIX_FMT_YUV422P, pCodecCtx->width, pCodecCtx->height);*/

        ......

        FILE *output=fopen("out.rgb","wb+");
        //----------------------------
        while(av_read_frame(pFormatCtx, packet)>=0)
        {
            if(packet->stream_index==videoindex)
            {
                ret = avcodec_decode_video2(pCodecCtx, pFrame, &got_picture, packet);

                if(ret < 0)
                {
                    printf("解码错误\n");
                    return -1;
                }
                if(got_picture)
                {
                    /*img_convert_ctx = sws_getContext(pCodecCtx->width, pCodecCtx->height, pCodecCtx->pix_fmt, pCodecCtx->width, pCodecCtx->height, PIX_FMT_UYVY422, SWS_BICUBIC, NULL, NULL, NULL);
                    sws_scale(img_convert_ctx, (const uint8_t* const*)pFrame->data, pFrame->linesize, 0, pCodecCtx->height, pFrameYUV->data, pFrameYUV->linesize);
                    img_convert_ctx = sws_getContext(pCodecCtx->width, pCodecCtx->height, pCodecCtx->pix_fmt, pCodecCtx->width, pCodecCtx->height, PIX_FMT_YUV422P, SWS_BICUBIC, NULL, NULL, NULL);
                    sws_scale(img_convert_ctx, (const uint8_t* const*)pFrame->data, pFrame->linesize, 0, pCodecCtx->height, pFrameYUV->data, pFrameYUV->linesize);*/
                    //转换
                    img_convert_ctx = sws_getContext(pCodecCtx->width, pCodecCtx->height, pCodecCtx->pix_fmt, pCodecCtx->width, pCodecCtx->height, PIX_FMT_RGB24, SWS_BICUBIC, NULL, NULL, NULL);
                    sws_scale(img_convert_ctx, (const uint8_t* const*)pFrame->data, pFrame->linesize, 0, pCodecCtx->height, pFrameYU
```

```
V->data, pFrameYUV->linesize);


                    //RGB
                    fwrite(pFrameYUV->data[0],(pCodecCtx->width)*(pCodecCtx->height)*3,1,output);
                    /*
                    //UYVY
                    fwrite(pFrameYUV->data[0],(pCodecCtx->width)*(pCodecCtx->height),2,output);
                    //YUV420P
                    fwrite(pFrameYUV->data[0],(pCodecCtx->width)*(pCodecCtx->height),1,output);
                    fwrite(pFrameYUV->data[1],(pCodecCtx->width)*(pCodecCtx->height)/4,1,output);
                    fwrite(pFrameYUV->data[2],(pCodecCtx->width)*(pCodecCtx->height)/4,1,output);
                    */
                    ......

                }
            }
            av_free_packet(packet);
        }

        fclose(output);

        ......

        return 0;
}
```

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：

[cpp]

```cpp
AVFrame *pFrame,*pFrameYUV;
pFrame=avcodec_alloc_frame();
pFrameYUV=avcodec_alloc_frame();
uint8_t *out_buffer;

out_buffer=new uint8_t[avpicture_get_size(PIX_FMT_RGB24, pCodecCtx->width, pCodecCtx->height)];
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：

[cpp]

```cpp
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：

[cpp]

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

保存RGB24格式的数据，用以下代码：

[cpp]

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

保存UYVY格式的数据，用以下代码：

[cpp]

```cpp
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。

[cpp]

```cpp
/* 雷霄骅
 * 中国传媒大学/数字电视技术
 * leixiaohua1020@126.com
 *
 */
 /*
 * copyright (c) 2006 Michael Niedermayer <michaelni@gmx.at>
 *
 * This file is part of FFmpeg.
 *
 * FFmpeg is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * FFmpeg is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with FFmpeg; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
 */

#ifndef AVUTIL_PIXFMT_H
#define AVUTIL_PIXFMT_H

/**
 * @file
 * pixel format definitions
 *
 */

#include "libavutil/avconfig.h"

/**
 * Pixel format.
 *
 * @note
 * PIX_FMT_RGB32 is handled in an endian-specific manner. An RGBA
 * color is put together as:
 *  (A << 24) | (R << 16) | (G << 8) | B
 * This is stored as BGRA on little-endian CPU architectures and ARGB on
 * big-endian CPUs.
 *
 * @par
 * When the pixel format is palettized RGB (PIX_FMT_PAL8), the palettized
 * image data is stored in AVFrame.data[0]. The palette is transported in
 * AVFrame.data[1], is 1024 bytes long (256 4-byte entries) and is
 * formatted the same as in PIX_FMT_RGB32 described above (i.e., it is
 * also endian-specific). Note also that the individual RGB palette
 * components stored in AVFrame.data[1] should be in the range 0..255.
 * This is important as many custom PAL8 video codecs that were designed
 * to run on the IBM VGA graphics adapter use 6-bit palette components.
 *
 * @par
 * For all the 8bit per pixel formats, an RGB32 palette is in data[1] like
 * for pal8. This palette is filled in automatically by the function
 * allocating the picture.
 *
 * @note
 * make sure that all newly added big endian formats have pix_fmt&1==1
 * and that all newly added little endian formats have pix_fmt&1==0
 * this allows simpler detection of big vs little endian.
 */
enum PixelFormat {
    PIX_FMT_NONE= -1,
    PIX_FMT_YUV420P,   ///< planar YUV 4:2:0, 12bpp, (1 Cr & Cb sample per 2x2 Y samples)
    PIX_FMT_YUYV422,   ///< packed YUV 4:2:2, 16bpp, Y0 Cb Y1 Cr
    PIX_FMT_RGB24,     ///< packed RGB 8:8:8, 24bpp, RGBRGB...
    PIX_FMT_BGR24,     ///< packed RGB 8:8:8, 24bpp, BGRBGR...
    PIX_FMT_YUV422P,   ///< planar YUV 4:2:2, 16bpp, (1 Cr & Cb sample per 2x1 Y samples)
    PIX_FMT_YUV444P,   ///< planar YUV 4:4:4, 24bpp, (1 Cr & Cb sample per 1x1 Y samples)
    PIX_FMT_YUV410P,   ///< planar YUV 4:1:0,  9bpp, (1 Cr & Cb sample per 4x4 Y samples)
    PIX_FMT_YUV411P,   ///< planar YUV 4:1:1, 12bpp, (1 Cr & Cb sample per 4x1 Y samples)
    PIX_FMT_GRAY8,     ///<        Y        ,  8bpp
    PIX_FMT_MONOWHITE, ///<        Y        ,  1bpp, 0 is white, 1 is black, in each byte pixels are ordered from the msb to the lsb
    PIX_FMT_MONOBLACK, ///<        Y        ,  1bpp, 0 is black, 1 is white, in each byte pixels are ordered from the msb to the lsb
    PIX_FMT_PAL8,      ///< 8 bit with PIX_FMT_RGB32 palette
    PIX_FMT_YUVJ420P,  ///< planar YUV 4:2:0, 12bpp, full scale (JPEG), deprecated in favor of PIX_FMT_YUV420P and setting color_rang
    PIX_FMT_YUVJ422P,  ///< planar YUV 4:2:2, 16bpp, full scale (JPEG), deprecated in favor of PIX_FMT_YUV422P and setting color_rang
```

```
    PIX_FMT_YUVJ444P,  ///< planar YUV 4:4:4, 24bpp, full scale (JPEG), deprecated in favor of PIX_FMT_YUV444P and setting color_rang

    PIX_FMT_XVMC_MPEG2_MC,///< XVideo Motion Acceleration via common packet passing
    PIX_FMT_XVMC_MPEG2_IDCT,
    PIX_FMT_UYVY422,   ///< packed YUV 4:2:2, 16bpp, Cb Y0 Cr Y1
    PIX_FMT_UYYVYY411, ///< packed YUV 4:1:1, 12bpp, Cb Y0 Y1 Cr Y2 Y3
    PIX_FMT_BGR8,      ///< packed RGB 3:3:2,  8bpp, (msb)2B 3G 3R(lsb)
    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
yte is the one composed by the 4 msb bits
    PIX_FMT_BGR4_BYTE, ///< packed RGB 1:2:1,  8bpp, (msb)1B 2G 1R(lsb)
    PIX_FMT_RGB8,      ///< packed RGB 3:3:2,  8bpp, (msb)2R 3G 3B(lsb)
    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
yte is the one composed by the 4 msb bits
    PIX_FMT_RGB4_BYTE, ///< packed RGB 1:2:1,  8bpp, (msb)1R 2G 1B(lsb)
    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
 U and the following byte V)
    PIX_FMT_NV21,      ///< as above, but U and V bytes are swapped

    PIX_FMT_ARGB,      ///< packed ARGB 8:8:8:8, 32bpp, ARGBARGB...
    PIX_FMT_RGBA,      ///< packed RGBA 8:8:8:8, 32bpp, RGBARGBA...
    PIX_FMT_ABGR,      ///< packed ABGR 8:8:8:8, 32bpp, ABGRABGR...
    PIX_FMT_BGRA,      ///< packed BGRA 8:8:8:8, 32bpp, BGRABGRA...

    PIX_FMT_GRAY16BE,  ///<        Y        , 16bpp, big-endian
    PIX_FMT_GRAY16LE,  ///<        Y        , 16bpp, little-endian
    PIX_FMT_YUV440P,   ///< planar YUV 4:4:0 (1 Cr & Cb sample per 1x2 Y samples)
    PIX_FMT_YUVJ440P,  ///< planar YUV 4:4:0 full scale (JPEG), deprecated in favor of PIX_FMT_YUV440P and setting color_range
    PIX_FMT_YUVA420P,  ///< planar YUV 4:2:0, 20bpp, (1 Cr & Cb sample per 2x2 Y & A samples)
    PIX_FMT_VDPAU_H264,///< H.264 HW decoding with VDPAU, data[0] contains a vdpau_render_state struct which contains the bitstream o
the slices as well as various fields extracted from headers
    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
lds extracted from headers
    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
lds extracted from headers
    PIX_FMT_VDPAU_WMV3,///< WMV3 HW decoding with VDPAU, data[0] contains a vdpau_render_state struct which contains the bitstream of
he slices as well as various fields extracted from headers
    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
lds extracted from headers
    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-en
dian
    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

    PIX_FMT_RGB565BE,  ///< packed RGB 5:6:5, 16bpp, (msb)   5R 6G 5B(lsb), big-endian
    PIX_FMT_RGB565LE,  ///< packed RGB 5:6:5, 16bpp, (msb)   5R 6G 5B(lsb), little-endian
    PIX_FMT_RGB555BE,  ///< packed RGB 5:5:5, 16bpp, (msb)1A 5R 5G 5B(lsb), big-endian, most significant bit to 0
    PIX_FMT_RGB555LE,  ///< packed RGB 5:5:5, 16bpp, (msb)1A 5R 5G 5B(lsb), little-endian, most significant bit to 0

    PIX_FMT_BGR565BE,  ///< packed BGR 5:6:5, 16bpp, (msb)   5B 6G 5R(lsb), big-endian
    PIX_FMT_BGR565LE,  ///< packed BGR 5:6:5, 16bpp, (msb)   5B 6G 5R(lsb), little-endian
    PIX_FMT_BGR555BE,  ///< packed BGR 5:5:5, 16bpp, (msb)1A 5B 5G 5R(lsb), big-endian, most significant bit to 1
    PIX_FMT_BGR555LE,  ///< packed BGR 5:5:5, 16bpp, (msb)1A 5B 5G 5R(lsb), little-endian, most significant bit to 1

    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 heade

    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
    PIX_FMT_VAAPI_VLD,  ///< HW decoding through VA API, Picture.data[3] contains a vaapi_render_state struct which contains the bits
eam of the slices as well as various fields extracted from headers

    PIX_FMT_YUV420P16LE,  ///< planar YUV 4:2:0, 24bpp, (1 Cr & Cb sample per 2x2 Y samples), little-endian
    PIX_FMT_YUV420P16BE,  ///< planar YUV 4:2:0, 24bpp, (1 Cr & Cb sample per 2x2 Y samples), big-endian
    PIX_FMT_YUV422P16LE,  ///< planar YUV 4:2:2, 32bpp, (1 Cr & Cb sample per 2x1 Y samples), little-endian
    PIX_FMT_YUV422P16BE,  ///< planar YUV 4:2:2, 32bpp, (1 Cr & Cb sample per 2x1 Y samples), big-endian
    PIX_FMT_YUV444P16LE,  ///< planar YUV 4:4:4, 48bpp, (1 Cr & Cb sample per 1x1 Y samples), little-endian
    PIX_FMT_YUV444P16BE,  ///< planar YUV 4:4:4, 48bpp, (1 Cr & Cb sample per 1x1 Y samples), big-endian
    PIX_FMT_VDPAU_MPEG4,  ///< MPEG4 HW decoding with VDPAU, data[0] contains a vdpau_render_state struct which contains the bitstrea
of the slices as well as various fields extracted from headers
    PIX_FMT_DXVA2_VLD,    ///< HW decoding through DXVA2, Picture.data[3] contains a LPDIRECT3DSURFACE9 pointer

    PIX_FMT_RGB444LE,  ///< packed RGB 4:4:4, 16bpp, (msb)4A 4R 4G 4B(lsb), little-endian, most significant bits to 0
    PIX_FMT_RGB444BE,  ///< packed RGB 4:4:4, 16bpp, (msb)4A 4R 4G 4B(lsb), big-endian, most significant bits to 0
    PIX_FMT_BGR444LE,  ///< packed BGR 4:4:4, 16bpp, (msb)4A 4B 4G 4R(lsb), little-endian, most significant bits to 1
    PIX_FMT_BGR444BE,  ///< packed BGR 4:4:4, 16bpp, (msb)4A 4B 4G 4R(lsb), big-endian, most significant bits to 1
    PIX_FMT_GRAY8A,    ///< 8bit gray, 8bit alpha
    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-en
dian
    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

    //the following 10 formats have the disadvantage of needing 1 format for each bit depth, thus
    //If you want to support multiple bit depths, then using PIX_FMT_YUV420P16* with the bpp stored seperately
    //is better
    PIX_FMT_YUV420P9BE, ///< planar YUV 4:2:0, 13.5bpp, (1 Cr & Cb sample per 2x2 Y samples), big-endian
    PIX_FMT_YUV420P9LE, ///< planar YUV 4:2:0, 13.5bpp, (1 Cr & Cb sample per 2x2 Y samples), little-endian
    PIX_FMT_YUV420P10BE,///< planar YUV 4:2:0, 15bpp, (1 Cr & Cb sample per 2x2 Y samples), big-endian
    PIX_FMT_YUV420P10LE,///< planar YUV 4:2:0, 15bpp, (1 Cr & Cb sample per 2x2 Y samples), little-endian
```

```
    PIX_FMT_YUV422P10BE,///< planar YUV 4:2:2, 20bpp, (1 Cr & Cb sample per 2x1 Y samples), big-endian
    PIX_FMT_YUV422P10LE,///< planar YUV 4:2:2, 20bpp, (1 Cr & Cb sample per 2x1 Y samples), little-endian
    PIX_FMT_YUV444P9BE, ///< planar YUV 4:4:4, 27bpp, (1 Cr & Cb sample per 1x1 Y samples), big-endian
    PIX_FMT_YUV444P9LE, ///< planar YUV 4:4:4, 27bpp, (1 Cr & Cb sample per 1x1 Y samples), little-endian
    PIX_FMT_YUV444P10BE,///< planar YUV 4:4:4, 30bpp, (1 Cr & Cb sample per 1x1 Y samples), big-endian
    PIX_FMT_YUV444P10LE,///< planar YUV 4:4:4, 30bpp, (1 Cr & Cb sample per 1x1 Y samples), little-endian
    PIX_FMT_YUV422P9BE, ///< planar YUV 4:2:2, 18bpp, (1 Cr & Cb sample per 2x1 Y samples), big-endian
    PIX_FMT_YUV422P9LE, ///< planar YUV 4:2:2, 18bpp, (1 Cr & Cb sample per 2x1 Y samples), little-endian
    PIX_FMT_VDA_VLD,    ///< hardware decoding through VDA

#ifdef AV_PIX_FMT_ABI_GIT_MASTER
    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
    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
    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
    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
#endif
    PIX_FMT_GBRP,      ///< planar GBR 4:4:4 24bpp
    PIX_FMT_GBRP9BE,   ///< planar GBR 4:4:4 27bpp, big endian
    PIX_FMT_GBRP9LE,   ///< planar GBR 4:4:4 27bpp, little endian
    PIX_FMT_GBRP10BE,  ///< planar GBR 4:4:4 30bpp, big endian
    PIX_FMT_GBRP10LE,  ///< planar GBR 4:4:4 30bpp, little endian
    PIX_FMT_GBRP16BE,  ///< planar GBR 4:4:4 48bpp, big endian
    PIX_FMT_GBRP16LE,  ///< planar GBR 4:4:4 48bpp, little endian

#ifndef AV_PIX_FMT_ABI_GIT_MASTER
    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
    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
    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
    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
#endif
    PIX_FMT_0RGB=0x123+4,      ///< packed RGB 8:8:8, 32bpp, 0RGB0RGB...
    PIX_FMT_RGB0,      ///< packed RGB 8:8:8, 32bpp, RGB0RGB0...
    PIX_FMT_0BGR,      ///< packed BGR 8:8:8, 32bpp, 0BGR0BGR...
    PIX_FMT_BGR0,      ///< packed BGR 8:8:8, 32bpp, BGR0BGR0...
    PIX_FMT_YUVA444P,  ///< planar YUV 4:4:4 32bpp, (1 Cr & Cb sample per 1x1 Y & A samples)

    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
};

#define PIX_FMT_Y400A PIX_FMT_GRAY8A
#define PIX_FMT_GBR24P PIX_FMT_GBRP

#if AV_HAVE_BIGENDIAN
#   define PIX_FMT_NE(be, le) PIX_FMT_##be
#else
#   define PIX_FMT_NE(be, le) PIX_FMT_##le
#endif

#define PIX_FMT_RGB32   PIX_FMT_NE(ARGB, BGRA)
#define PIX_FMT_RGB32_1 PIX_FMT_NE(RGBA, ABGR)
#define PIX_FMT_BGR32   PIX_FMT_NE(ABGR, RGBA)
#define PIX_FMT_BGR32_1 PIX_FMT_NE(BGRA, ARGB)
#define PIX_FMT_0RGB32  PIX_FMT_NE(0RGB, BGR0)
#define PIX_FMT_0BGR32  PIX_FMT_NE(0BGR, RGB0)

#define PIX_FMT_GRAY16 PIX_FMT_NE(GRAY16BE, GRAY16LE)
#define PIX_FMT_RGB48  PIX_FMT_NE(RGB48BE,  RGB48LE)
#define PIX_FMT_RGB565 PIX_FMT_NE(RGB565BE, RGB565LE)
#define PIX_FMT_RGB555 PIX_FMT_NE(RGB555BE, RGB555LE)
#define PIX_FMT_RGB444 PIX_FMT_NE(RGB444BE, RGB444LE)
#define PIX_FMT_BGR48  PIX_FMT_NE(BGR48BE,  BGR48LE)
#define PIX_FMT_BGR565 PIX_FMT_NE(BGR565BE, BGR565LE)
#define PIX_FMT_BGR555 PIX_FMT_NE(BGR555BE, BGR555LE)
#define PIX_FMT_BGR444 PIX_FMT_NE(BGR444BE, BGR444LE)

#define PIX_FMT_YUV420P9  PIX_FMT_NE(YUV420P9BE , YUV420P9LE)
#define PIX_FMT_YUV422P9  PIX_FMT_NE(YUV422P9BE , YUV422P9LE)
#define PIX_FMT_YUV444P9  PIX_FMT_NE(YUV444P9BE , YUV444P9LE)
#define PIX_FMT_YUV420P10 PIX_FMT_NE(YUV420P10BE, YUV420P10LE)
#define PIX_FMT_YUV422P10 PIX_FMT_NE(YUV422P10BE, YUV422P10LE)
#define PIX_FMT_YUV444P10 PIX_FMT_NE(YUV444P10BE, YUV444P10LE)
#define PIX_FMT_YUV420P16 PIX_FMT_NE(YUV420P16BE, YUV420P16LE)
#define PIX_FMT_YUV422P16 PIX_FMT_NE(YUV422P16BE, YUV422P16LE)
#define PIX_FMT_YUV444P16 PIX_FMT_NE(YUV444P16BE, YUV444P16LE)

#define PIX_FMT_RGBA64 PIX_FMT_NE(RGBA64BE, RGBA64LE)
#define PIX_FMT_BGRA64 PIX_FMT_NE(BGRA64BE, BGRA64LE)
#define PIX_FMT_GBRP9     PIX_FMT_NE(GBRP9BE ,    GBRP9LE)
#define PIX_FMT_GBRP10    PIX_FMT_NE(GBRP10BE,    GBRP10LE)
#define PIX_FMT_GBRP16    PIX_FMT_NE(GBRP16BE,    GBRP16LE)

#endif /* AVUTIL_PIXFMT_H */
```

文章标签： FFMPEG YUV RGB 转换 视频

个人分类： FFMPEG

所属专栏： FFmpeg