

## 最简单的基于FFmpeg的libswscale的示例（YUV转RGB）

2014年12月28日 00:46:53 阅读数：38287

最简单的基于FFmpeg的libswscale的示例系列文章列表：

[最简单的基于FFmpeg的libswscale的示例（YUV转RGB）](#)

[最简单的基于FFmpeg的libswscale的示例附件：测试图片生成工具](#)

本文记录一个基于FFmpeg的libswscale的示例。Libswscale里面实现了各种图像像素格式的转换，例如YUV与RGB之间的转换；以及图像大小缩放（例如640x360拉伸为1280x720）功能。而且libswscale还做了相应指令集的优化，因此它的转换效率比自己写的C语言的转换效率高很多。

本文记录的程序将像素格式为YUV420P，分辨率为480x272的视频转换为像素格式为RGB24，分辨率为1280x720的视频。

## 流程

### 简单的初始化方法

Libswscale使用起来很方便，最主要的函数只有3个：

- (1) `sws_getContext()`：使用参数初始化SwsContext结构体。
- (2) `sws_scale()`：转换一帧图像。
- (3) `sws_freeContext()`：释放SwsContext结构体。

其中`sws_getContext()`也可以用另一个接口函数`sws_getCachedContext()`取代。

### 复杂但是更灵活的初始化方法

初始化SwsContext除了调用`sws_getContext()`之外还有另一种方法，更加灵活，可以配置更多的参数。该方法调用的函数如下所示。

- (1) `sws_alloc_context()`：为SwsContext结构体分配内存。
- (2) `av_opt_set_XXX()`：通过`av_opt_set_int()`，`av_opt_set()`...等等一系列方法设置SwsContext结构体的值。在这里需要注意，SwsContext结构体的定义看不到，所以不能对其中的成员变量直接进行赋值，必须通过`av_opt_set()`这类的API才能对其进行赋值。
- (3) `sws_init_context()`：初始化SwsContext结构体。

这种复杂的方法可以配置一些`sws_getContext()`配置不了的参数。比如说设置图像的YUV像素的取值范围是JPEG标准（Y、U、V取值范围都是0-255）还是MPEG标准（Y取值范围是16-235，U、V的取值范围是16-240）。

## 几个知识点

下文记录几个图像像素数据处理过程中的几个知识点：像素格式，图像拉伸，YUV像素取值范围，色域。

### 像素格式

像素格式的知识此前已经记录过，不再重复。在这里记录一下FFmpeg支持的像素格式。有几点注意事项：

- (1) 所有的像素格式的名称都是以“AV\_PIX\_FMT\_”开头
- (2) 像素格式名称后面有“P”的，代表是planar格式，否则就是packed格式。Planar格式不同的分量分别存储在不同的数组中，例如AV\_PIX\_FMT\_YUV420P存储方式如下：

```
data[0]: Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8.....
data[1]: U1, U2, U3, U4.....
data[2]: V1, V2, V3, V4.....
```

Packed格式的数据都存储在同一个数组中，例如AV\_PIX\_FMT\_RGB24存储方式如下：

```
data[0]: R1, G1, B1, R2, G2, B2, R3, G3, B3, R4, G4, B4.....
```

- (3) 像素格式名称后面有“BE”的，代表是Big Endian格式；名称后面有“LE”的，代表是Little Endian格式。

FFmpeg支持的像素格式的定义位于`libavutil/pixfmt.h`，是一个名称为AVPixelFormat的枚举类型，如下所示。

```
[cpp]
1.  /**
2.   * Pixel format.
3.   *
4.   * @note
5.   * AV_PIX_FMT_RGB32 is handled in an endian-specific manner. An RGBA
6.   * color is put together as:
7.   * (A << 24) | (R << 16) | (G << 8) | B
8.   * This is stored as BGRA on little-endian CPU architectures and ARGB on
9.   * big-endian CPUs.
```

```

9.  * big-endian CPUs.
10. *
11. * @par
12. * When the pixel format is palettized RGB (AV_PIX_FMT_PAL8), the palettized
13. * image data is stored in AVFrame.data[0]. The palette is transported in
14. * AVFrame.data[1], is 1024 bytes long (256 4-byte entries) and is
15. * formatted the same as in AV_PIX_FMT_RGB32 described above (i.e., it is
16. * also endian-specific). Note also that the individual RGB palette
17. * components stored in AVFrame.data[1] should be in the range 0..255.
18. * This is important as many custom PAL8 video codecs that were designed
19. * to run on the IBM VGA graphics adapter use 6-bit palette components.
20. *
21. * @par
22. * For all the 8bit per pixel formats, an RGB32 palette is in data[1] like
23. * for pal8. This palette is filled in automatically by the function
24. * allocating the picture.
25. *
26. * @note
27. * Make sure that all newly added big-endian formats have (pix_fmt & 1) == 1
28. * and that all newly added little-endian formats have (pix_fmt & 1) == 0.
29. * This allows simpler detection of big vs little-endian.
30. */
31. enum AVPixelFormat {
32.     AV_PIX_FMT_NONE = -1,
33.     AV_PIX_FMT_YUV420P, ///< planar YUV 4:2:0, 12bpp, (1 Cr & Cb sample per 2x2 Y samples)
34.     AV_PIX_FMT_YUV422,  ///< packed YUV 4:2:2, 16bpp, Y0 Cb Y1 Cr
35.     AV_PIX_FMT_RGB24,    ///< packed RGB 8:8:8, 24bpp, RGBRGB...
36.     AV_PIX_FMT_BGR24,    ///< packed RGB 8:8:8, 24bpp, BGRBGR...
37.     AV_PIX_FMT_YUV422P,  ///< planar YUV 4:2:2, 16bpp, (1 Cr & Cb sample per 2x1 Y samples)
38.     AV_PIX_FMT_YUV444P,  ///< planar YUV 4:4:4, 24bpp, (1 Cr & Cb sample per 1x1 Y samples)
39.     AV_PIX_FMT_YUV410P,  ///< planar YUV 4:1:0, 9bpp, (1 Cr & Cb sample per 4x4 Y samples)
40.     AV_PIX_FMT_YUV411P,  ///< planar YUV 4:1:1, 12bpp, (1 Cr & Cb sample per 4x1 Y samples)
41.     AV_PIX_FMT_GRAY8,    ///<      Y      , 8bpp
42.     AV_PIX_FMT_MONOWHITE, ///<      Y      , 1bpp, 0 is white, 1 is black, in each byte pixels are ordered from the msb to the l
43.
44.     AV_PIX_FMT_MONOBLACK, ///<      Y      , 1bpp, 0 is black, 1 is white, in each byte pixels are ordered from the msb to the l
45.
46.     AV_PIX_FMT_PAL8,     ///< 8 bit with PIX_FMT_RGB32 palette
47.     AV_PIX_FMT_YUVJ420P, ///< planar YUV 4:2:0, 12bpp, full scale (JPEG), deprecated in favor of PIX_FMT_YUV420P and setting color_r
48.     AV_PIX_FMT_YUVJ422P, ///< planar YUV 4:2:2, 16bpp, full scale (JPEG), deprecated in favor of PIX_FMT_YUV422P and setting color_r
49.     AV_PIX_FMT_YUVJ444P, ///< planar YUV 4:4:4, 24bpp, full scale (JPEG), deprecated in favor of PIX_FMT_YUV444P and setting color_r
50.
51. #if FF_API_XVMC
52.     AV_PIX_FMT_XVMC_MPEG2_MC, ///< XVideo Motion Acceleration via common packet passing
53.     AV_PIX_FMT_XVMC_MPEG2_IDCT,
54. #define AV_PIX_FMT_XVMC AV_PIX_FMT_XVMC_MPEG2_IDCT
55. #endif /* FF_API_XVMC */
56.     AV_PIX_FMT_UYVY422,  ///< packed YUV 4:2:2, 16bpp, Cb Y0 Cr Y1
57.     AV_PIX_FMT_UYVYY411, ///< packed YUV 4:1:1, 12bpp, Cb Y0 Y1 Cr Y2 Y3
58.     AV_PIX_FMT_BGR8,      ///< packed RGB 3:3:2, 8bpp, (msb)2B 3G 3R(lsb)
59.     AV_PIX_FMT_BGR4,      ///< packed RGB 1:2:1 bitstream, 4bpp, (msb)1B 2G 1R(lsb), a byte contains two pixels, the first pixel in
60.     e byte is the one composed by the 4 msb bits
61.     AV_PIX_FMT_BGR4_BYTE,  ///< packed RGB 1:2:1, 8bpp, (msb)1B 2G 1R(lsb)
62.     AV_PIX_FMT_RGB8,      ///< packed RGB 3:3:2, 8bpp, (msb)2R 3G 3B(lsb)
63.     AV_PIX_FMT_RGB4,      ///< packed RGB 1:2:1 bitstream, 4bpp, (msb)1R 2G 1B(lsb), a byte contains two pixels, the first pixel in
64.     e byte is the one composed by the 4 msb bits
65.     AV_PIX_FMT_RGB4_BYTE,  ///< packed RGB 1:2:1, 8bpp, (msb)1R 2G 1B(lsb)
66.     AV_PIX_FMT_NV12,      ///< planar YUV 4:2:0, 12bpp, 1 plane for Y and 1 plane for the UV components, which are interleaved (first
67.     yte U and the following byte V)
68.     AV_PIX_FMT_NV21,      ///< as above, but U and V bytes are swapped
69.
70.     AV_PIX_FMT_ARGB,      ///< packed ARGB 8:8:8:8, 32bpp, ARGBARGB...
71.     AV_PIX_FMT_RGBA,      ///< packed RGBA 8:8:8:8, 32bpp, RGBARGBA...
72.     AV_PIX_FMT_ABGR,      ///< packed ABGR 8:8:8:8, 32bpp, ABGRABGR...
73.     AV_PIX_FMT_BGRA,      ///< packed BGRA 8:8:8:8, 32bpp, BGRABGRA...
74.
75.     AV_PIX_FMT_GRAY16BE,  ///<      Y      , 16bpp, big-endian
76.     AV_PIX_FMT_GRAY16LE,  ///<      Y      , 16bpp, little-endian
77.     AV_PIX_FMT_YUV440P,   ///< planar YUV 4:4:0 (1 Cr & Cb sample per 1x2 Y samples)
78.     AV_PIX_FMT_YUVJ440P,  ///< planar YUV 4:4:0 full scale (JPEG), deprecated in favor of PIX_FMT_YUV440P and setting color_range
79.     AV_PIX_FMT_YUV4A20P,  ///< planar YUV 4:2:0, 20bpp, (1 Cr & Cb sample per 2x2 Y & A samples)
80.
81. #if FF_API_VDPAU
82.     AV_PIX_FMT_VDPAU_H264, ///< H.264 HW decoding with VDPAU, data[0] contains a vdpau_render_state struct which contains the bitstream
83.     of the slices as well as various fields extracted from headers
84.     AV_PIX_FMT_VDPAU_MPEG1, ///< MPEG-
85.     1 HW decoding with VDPAU, data[0] contains a vdpau_render_state struct which contains the bitstream of the slices as well as various
86.     lds extracted from headers
87.     AV_PIX_FMT_VDPAU_MPEG2, ///< MPEG-
88.     2 HW decoding with VDPAU, data[0] contains a vdpau_render_state struct which contains the bitstream of the slices as well as various
89.     lds extracted from headers
90.     AV_PIX_FMT_VDPAU_WMV3, ///< WMV3 HW decoding with VDPAU, data[0] contains a vdpau_render_state struct which contains the bitstream
91.     f the slices as well as various fields extracted from headers
92.     AV_PIX_FMT_VDPAU_VC1,  ///< VC-
93.     1 HW decoding with VDPAU, data[0] contains a vdpau_render_state struct which contains the bitstream of the slices as well as various
94.     lds extracted from headers
95. #endif
96.     AV_PIX_FMT_RGB48BE,    ///< packed RGB 16:16:16, 48bpp, 16R, 16G, 16B, the 2-
97.     byte value for each R/G/B component is stored as big-endian
98.     AV_PIX_FMT_RGB48LE,    ///< packed RGB 16:16:16, 48bpp, 16R, 16G, 16B, the 2-
99.     byte value for each R/G/B component is stored as little-endian

```

byte value for each R/G/B component is stored as little-endian

```
83.
84.     AV_PIX_FMT_RGB565BE, ///< packed RGB 5:6:5, 16bpp, (msb) 5R 6G 5B(lsb), big-endian
85.     AV_PIX_FMT_RGB565LE, ///< packed RGB 5:6:5, 16bpp, (msb) 5R 6G 5B(lsb), little-endian
86.     AV_PIX_FMT_RGB555BE, ///< packed RGB 5:5:5, 16bpp, (msb)1A 5R 5G 5B(lsb), big-endian, most significant bit to 0
87.     AV_PIX_FMT_RGB555LE, ///< packed RGB 5:5:5, 16bpp, (msb)1A 5R 5G 5B(lsb), little-endian, most significant bit to 0
88.
89.     AV_PIX_FMT_BGR565BE, ///< packed BGR 5:6:5, 16bpp, (msb) 5B 6G 5R(lsb), big-endian
90.     AV_PIX_FMT_BGR565LE, ///< packed BGR 5:6:5, 16bpp, (msb) 5B 6G 5R(lsb), little-endian
91.     AV_PIX_FMT_BGR555BE, ///< packed BGR 5:5:5, 16bpp, (msb)1A 5B 5G 5R(lsb), big-endian, most significant bit to 1
92.     AV_PIX_FMT_BGR555LE, ///< packed BGR 5:5:5, 16bpp, (msb)1A 5B 5G 5R(lsb), little-endian, most significant bit to 1
93.
94.     AV_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
95.
96.     AV_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
97.
98.     AV_PIX_FMT_VAAPI_VLD, ///< HW decoding through VA API, Picture.data[3] contains a vaapi_render_state struct which contains the b
    stream of the slices as well as various fields extracted from headers
99.
100.     AV_PIX_FMT_YUV420P16LE, ///< planar YUV 4:2:0, 24bpp, (1 Cr & Cb sample per 2x2 Y samples), little-endian
101.     AV_PIX_FMT_YUV420P16BE, ///< planar YUV 4:2:0, 24bpp, (1 Cr & Cb sample per 2x2 Y samples), big-endian
102.     AV_PIX_FMT_YUV422P16LE, ///< planar YUV 4:2:2, 32bpp, (1 Cr & Cb sample per 2x1 Y samples), little-endian
103.     AV_PIX_FMT_YUV422P16BE, ///< planar YUV 4:2:2, 32bpp, (1 Cr & Cb sample per 2x1 Y samples), big-endian
104.     AV_PIX_FMT_YUV444P16LE, ///< planar YUV 4:4:4, 48bpp, (1 Cr & Cb sample per 1x1 Y samples), little-endian
105.     AV_PIX_FMT_YUV444P16BE, ///< planar YUV 4:4:4, 48bpp, (1 Cr & Cb sample per 1x1 Y samples), big-endian
106.
107. #if FF_API_VDPAU
108.     AV_PIX_FMT_VDPAU_MPEG4, ///< MPEG4 HW decoding with VDPAU, data[0] contains a vdpau_render_state struct which contains the bitst
    am of the slices as well as various fields extracted from headers
109. #endif
110.
111.     AV_PIX_FMT_DXVA2_VLD, ///< HW decoding through DXVA2, Picture.data[3] contains a LPDIRECT3DSURFACE9 pointer
112.
113.     AV_PIX_FMT_RGB444LE, ///< packed RGB 4:4:4, 16bpp, (msb)4A 4R 4G 4B(lsb), little-endian, most significant bits to 0
114.     AV_PIX_FMT_RGB444BE, ///< packed RGB 4:4:4, 16bpp, (msb)4A 4R 4G 4B(lsb), big-endian, most significant bits to 0
115.     AV_PIX_FMT_BGR444LE, ///< packed BGR 4:4:4, 16bpp, (msb)4A 4B 4G 4R(lsb), little-endian, most significant bits to 1
116.     AV_PIX_FMT_BGR444BE, ///< packed BGR 4:4:4, 16bpp, (msb)4A 4B 4G 4R(lsb), big-endian, most significant bits to 1
117.     AV_PIX_FMT_GRAY8A, ///< 8bit gray, 8bit alpha
118.     AV_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
119.     AV_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
120.
121. /**
122.  * The following 12 formats have the disadvantage of needing 1 format for each bit depth.
123.  * Notice that each 9/10 bits sample is stored in 16 bits with extra padding.
124.  * If you want to support multiple bit depths, then using AV_PIX_FMT_YUV420P16* with the bpp stored separately is better.
125.  */
126.     AV_PIX_FMT_YUV420P9BE, ///< planar YUV 4:2:0, 13.5bpp, (1 Cr & Cb sample per 2x2 Y samples), big-endian
127.     AV_PIX_FMT_YUV420P9LE, ///< planar YUV 4:2:0, 13.5bpp, (1 Cr & Cb sample per 2x2 Y samples), little-endian
128.     AV_PIX_FMT_YUV420P10BE, ///< planar YUV 4:2:0, 15bpp, (1 Cr & Cb sample per 2x2 Y samples), big-endian
129.     AV_PIX_FMT_YUV420P10LE, ///< planar YUV 4:2:0, 15bpp, (1 Cr & Cb sample per 2x2 Y samples), little-endian
130.     AV_PIX_FMT_YUV422P10BE, ///< planar YUV 4:2:2, 20bpp, (1 Cr & Cb sample per 2x1 Y samples), big-endian
131.     AV_PIX_FMT_YUV422P10LE, ///< planar YUV 4:2:2, 20bpp, (1 Cr & Cb sample per 2x1 Y samples), little-endian
132.     AV_PIX_FMT_YUV444P9BE, ///< planar YUV 4:4:4, 27bpp, (1 Cr & Cb sample per 1x1 Y samples), big-endian
133.     AV_PIX_FMT_YUV444P9LE, ///< planar YUV 4:4:4, 27bpp, (1 Cr & Cb sample per 1x1 Y samples), little-endian
134.     AV_PIX_FMT_YUV444P10BE, ///< planar YUV 4:4:4, 30bpp, (1 Cr & Cb sample per 1x1 Y samples), big-endian
135.     AV_PIX_FMT_YUV444P10LE, ///< planar YUV 4:4:4, 30bpp, (1 Cr & Cb sample per 1x1 Y samples), little-endian
136.     AV_PIX_FMT_YUV422P9BE, ///< planar YUV 4:2:2, 18bpp, (1 Cr & Cb sample per 2x1 Y samples), big-endian
137.     AV_PIX_FMT_YUV422P9LE, ///< planar YUV 4:2:2, 18bpp, (1 Cr & Cb sample per 2x1 Y samples), little-endian
138.     AV_PIX_FMT_VDA_VLD, ///< hardware decoding through VDA
139.
140. #ifdef AV_PIX_FMT_ABI_GIT_MASTER
141.     AV_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
142.     AV_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
143.     AV_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
144.     AV_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
145. #endif
146.
147.     AV_PIX_FMT_GBRP, ///< planar GBR 4:4:4 24bpp
148.     AV_PIX_FMT_GBRP9BE, ///< planar GBR 4:4:4 27bpp, big-endian
149.     AV_PIX_FMT_GBRP9LE, ///< planar GBR 4:4:4 27bpp, little-endian
150.     AV_PIX_FMT_GBRP10BE, ///< planar GBR 4:4:4 30bpp, big-endian
151.     AV_PIX_FMT_GBRP10LE, ///< planar GBR 4:4:4 30bpp, little-endian
152.     AV_PIX_FMT_GBRP16BE, ///< planar GBR 4:4:4 48bpp, big-endian
153.     AV_PIX_FMT_GBRP16LE, ///< planar GBR 4:4:4 48bpp, little-endian
154.
155. /**
156.  * duplicated pixel formats for compatibility with libav.
157.  * FFmpeg supports these formats since May 8 2012 and Jan 28 2012 (commits f9ca1ac7 and 143a5c55)
158.  * Libav added them Oct 12 2012 with incompatible values (commit 6d5600e85)
159.  */
160.     AV_PIX_FMT_YUVA422P_LIBAV, ///< planar YUV 4:2:2 24bpp, (1 Cr & Cb sample per 2x1 Y & A samples)
161.     AV_PIX_FMT_YUVA444P_LIBAV, ///< planar YUV 4:4:4 32bpp, (1 Cr & Cb sample per 1x1 Y & A samples)
162.
163.     AV_PIX_FMT_YUVA420P9BE, ///< planar YUV 4:2:0 22.5bpp, (1 Cr & Cb sample per 2x2 Y & A samples), big-endian
164.     AV_PIX_FMT_YUVA420P9LE, ///< planar YUV 4:2:0 22.5bpp, (1 Cr & Cb sample per 2x2 Y & A samples), little-endian
165.     AV_PIX_FMT_YUVA422P9BE, ///< planar YUV 4:2:2 27bpp, (1 Cr & Cb sample per 2x1 Y & A samples), big-endian
166.     AV_PIX_FMT_YUVA422P9LE, ///< planar YUV 4:2:2 27bpp, (1 Cr & Cb sample per 2x1 Y & A samples), little-endian
167.     AV_PIX_FMT_YUVA444P9BE, ///< planar YUV 4:4:4 36bpp, (1 Cr & Cb sample per 1x1 Y & A samples), big-endian
168.     AV_PIX_FMT_YUVA444P9LE, ///< planar YUV 4:4:4 36bpp, (1 Cr & Cb sample per 1x1 Y & A samples), little-endian
```

```

163.     AV_PIX_FMT_YUV444P9LE,    ///< planar YUV 4:4:4 36bpp, (1 Cr & Cb sample per 1x1 Y & A samples), little-endian
164.     AV_PIX_FMT_YUV420P10BE,   ///< planar YUV 4:2:0 25bpp, (1 Cr & Cb sample per 2x2 Y & A samples, big-endian)
165.     AV_PIX_FMT_YUV420P10LE,   ///< planar YUV 4:2:0 25bpp, (1 Cr & Cb sample per 2x2 Y & A samples, little-endian)
166.     AV_PIX_FMT_YUV422P10BE,   ///< planar YUV 4:2:2 30bpp, (1 Cr & Cb sample per 2x1 Y & A samples, big-endian)
167.     AV_PIX_FMT_YUV422P10LE,   ///< planar YUV 4:2:2 30bpp, (1 Cr & Cb sample per 2x1 Y & A samples, little-endian)
168.     AV_PIX_FMT_YUV444P10BE,   ///< planar YUV 4:4:4 40bpp, (1 Cr & Cb sample per 1x1 Y & A samples, big-endian)
169.     AV_PIX_FMT_YUV444P10LE,   ///< planar YUV 4:4:4 40bpp, (1 Cr & Cb sample per 1x1 Y & A samples, little-endian)
170.     AV_PIX_FMT_YUV420P16BE,   ///< planar YUV 4:2:0 40bpp, (1 Cr & Cb sample per 2x2 Y & A samples, big-endian)
171.     AV_PIX_FMT_YUV420P16LE,   ///< planar YUV 4:2:0 40bpp, (1 Cr & Cb sample per 2x2 Y & A samples, little-endian)
172.     AV_PIX_FMT_YUV422P16BE,   ///< planar YUV 4:2:2 48bpp, (1 Cr & Cb sample per 2x1 Y & A samples, big-endian)
173.     AV_PIX_FMT_YUV422P16LE,   ///< planar YUV 4:2:2 48bpp, (1 Cr & Cb sample per 2x1 Y & A samples, little-endian)
174.     AV_PIX_FMT_YUV444P16BE,   ///< planar YUV 4:4:4 64bpp, (1 Cr & Cb sample per 1x1 Y & A samples, big-endian)
175.     AV_PIX_FMT_YUV444P16LE,   ///< planar YUV 4:4:4 64bpp, (1 Cr & Cb sample per 1x1 Y & A samples, little-endian)
176.
177.     AV_PIX_FMT_VDPAU,         ///< HW acceleration through VDPAU, Picture.data[3] contains a VdpVideoSurface
178.
179.     AV_PIX_FMT_XYZ12LE,       ///< packed XYZ 4:4:4, 36 bpp, (msb) 12X, 12Y, 12Z (lsb), the 2-
byte value for each X/Y/Z is stored as little-endian, the 4 lower bits are set to 0
180.     AV_PIX_FMT_XYZ12BE,       ///< packed XYZ 4:4:4, 36 bpp, (msb) 12X, 12Y, 12Z (lsb), the 2-
byte value for each X/Y/Z is stored as big-endian, the 4 lower bits are set to 0
181.     AV_PIX_FMT_NV16,           ///< interleaved chroma YUV 4:2:2, 16bpp, (1 Cr & Cb sample per 2x1 Y samples)
182.     AV_PIX_FMT_NV20LE,        ///< interleaved chroma YUV 4:2:2, 20bpp, (1 Cr & Cb sample per 2x1 Y samples), little-endian
183.     AV_PIX_FMT_NV20BE,        ///< interleaved chroma YUV 4:2:2, 20bpp, (1 Cr & Cb sample per 2x1 Y samples), big-endian
184.
185.     /**
186.      * duplicated pixel formats for compatibility with libav.
187.      * Ffmpeg supports these formats since Sat Sep 24 06:01:45 2011 +0200 (commits 9569a3c9f41387a8c7d1ce97d8693520477a66c3)
188.      * also see Fri Nov 25 01:38:21 2011 +0100 92afb431621c79155fcb7171d26f137eb1bee028
189.      * Libav added them Sun Mar 16 23:05:47 2014 +0100 with incompatible values (commit 1481d24c3a0abf81e1d7a514547bd5305232be30)
190.      */
191.     AV_PIX_FMT_RGBA64BE_LIBAV, ///< 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
192.     AV_PIX_FMT_RGBA64LE_LIBAV, ///< 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
193.     AV_PIX_FMT_BGRA64BE_LIBAV, ///< 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
194.     AV_PIX_FMT_BGRA64LE_LIBAV, ///< 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
195.
196.     AV_PIX_FMT_YVYU422,       ///< packed YUV 4:2:2, 16bpp, Y0 Cr Y1 Cb
197.
198. #ifndef AV_PIX_FMT_ABI_GIT_MASTER
199.     AV_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
200.     AV_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
201.     AV_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
202.     AV_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
203. #endif
204.     AV_PIX_FMT_0RGB=0x123+4,   ///< packed RGB 8:8:8, 32bpp, 0RGB0RGB...
205.     AV_PIX_FMT_0RGB0,          ///< packed RGB 8:8:8, 32bpp, RGB0RGB0...
206.     AV_PIX_FMT_0BGR,           ///< packed BGR 8:8:8, 32bpp, 0BGR0BGR...
207.     AV_PIX_FMT_0BGR0,          ///< packed BGR 8:8:8, 32bpp, BGR0BGR0...
208.     AV_PIX_FMT_YUV444P,        ///< planar YUV 4:4:4 32bpp, (1 Cr & Cb sample per 1x1 Y & A samples)
209.     AV_PIX_FMT_YUV422P,        ///< planar YUV 4:2:2 24bpp, (1 Cr & Cb sample per 2x1 Y & A samples)
210.
211.     AV_PIX_FMT_YUV420P12BE,    ///< planar YUV 4:2:0,18bpp, (1 Cr & Cb sample per 2x2 Y samples), big-endian
212.     AV_PIX_FMT_YUV420P12LE,    ///< planar YUV 4:2:0,18bpp, (1 Cr & Cb sample per 2x2 Y samples), little-endian
213.     AV_PIX_FMT_YUV420P14BE,    ///< planar YUV 4:2:0,21bpp, (1 Cr & Cb sample per 2x2 Y samples), big-endian
214.     AV_PIX_FMT_YUV420P14LE,    ///< planar YUV 4:2:0,21bpp, (1 Cr & Cb sample per 2x2 Y samples), little-endian
215.     AV_PIX_FMT_YUV422P12BE,    ///< planar YUV 4:2:2,24bpp, (1 Cr & Cb sample per 2x1 Y samples), big-endian
216.     AV_PIX_FMT_YUV422P12LE,    ///< planar YUV 4:2:2,24bpp, (1 Cr & Cb sample per 2x1 Y samples), little-endian
217.     AV_PIX_FMT_YUV422P14BE,    ///< planar YUV 4:2:2,28bpp, (1 Cr & Cb sample per 2x1 Y samples), big-endian
218.     AV_PIX_FMT_YUV422P14LE,    ///< planar YUV 4:2:2,28bpp, (1 Cr & Cb sample per 2x1 Y samples), little-endian
219.     AV_PIX_FMT_YUV444P12BE,    ///< planar YUV 4:4:4,36bpp, (1 Cr & Cb sample per 1x1 Y samples), big-endian
220.     AV_PIX_FMT_YUV444P12LE,    ///< planar YUV 4:4:4,36bpp, (1 Cr & Cb sample per 1x1 Y samples), little-endian
221.     AV_PIX_FMT_YUV444P14BE,    ///< planar YUV 4:4:4,42bpp, (1 Cr & Cb sample per 1x1 Y samples), big-endian
222.     AV_PIX_FMT_YUV444P14LE,    ///< planar YUV 4:4:4,42bpp, (1 Cr & Cb sample per 1x1 Y samples), little-endian
223.     AV_PIX_FMT_GBRP12BE,       ///< planar GBR 4:4:4 36bpp, big-endian
224.     AV_PIX_FMT_GBRP12LE,       ///< planar GBR 4:4:4 36bpp, little-endian
225.     AV_PIX_FMT_GBRP14BE,       ///< planar GBR 4:4:4 42bpp, big-endian
226.     AV_PIX_FMT_GBRP14LE,       ///< planar GBR 4:4:4 42bpp, little-endian
227.     AV_PIX_FMT_GBRAP,          ///< planar GBRA 4:4:4:4 32bpp
228.     AV_PIX_FMT_GBRAP16BE,      ///< planar GBRA 4:4:4:4 64bpp, big-endian
229.     AV_PIX_FMT_GBRAP16LE,      ///< planar GBRA 4:4:4:4 64bpp, little-endian
230.     AV_PIX_FMT_YUVJ411P,       ///< planar YUV 4:1:1, 12bpp, (1 Cr & Cb sample per 4x1 Y samples) full scale (JPEG), deprecated in favor
f PIX_FMT_YUV411P and setting color_range
231.
232.     AV_PIX_FMT_BAYER_BGGR8,    ///< bayer, BGBG..(odd line), GRGR..(even line), 8-bit samples */
233.     AV_PIX_FMT_BAYER_RGGB8,    ///< bayer, RGRG..(odd line), GBGB..(even line), 8-bit samples */
234.     AV_PIX_FMT_BAYER_GBRG8,    ///< bayer, GBGB..(odd line), RGRG..(even line), 8-bit samples */
235.     AV_PIX_FMT_BAYER_GRBG8,    ///< bayer, GRGR..(odd line), BGBG..(even line), 8-bit samples */
236.     AV_PIX_FMT_BAYER_BGGR16LE, ///< bayer, BGBG..(odd line), GRGR..(even line), 16-bit samples, little-endian */
237.     AV_PIX_FMT_BAYER_BGG16BE,  ///< bayer, BGBG..(odd line), GRGR..(even line), 16-bit samples, big-endian */
238.     AV_PIX_FMT_BAYER_RGGB16LE, ///< bayer, RGRG..(odd line), GBGB..(even line), 16-bit samples, little-endian */
239.     AV_PIX_FMT_BAYER_RGG16BE,  ///< bayer, RGRG..(odd line), GBGB..(even line), 16-bit samples, big-endian */
240.     AV_PIX_FMT_BAYER_GBRG16LE, ///< bayer, GBGB..(odd line), RGRG..(even line), 16-bit samples, little-endian */
241.     AV_PIX_FMT_BAYER_GBRG16BE, ///< bayer, GBGB..(odd line), RGRG..(even line), 16-bit samples, big-endian */
242.     AV_PIX_FMT_BAYER_GRBG16LE, ///< bayer, GRGR..(odd line), BGBG..(even line), 16-bit samples, little-endian */

```

```

243.     AV_PIX_FMT_BAYER_GRBG16BE, ///< bayer, GRGR..(odd line), BGBG..(even line), 16-bit samples, big-endian */
244. #if !FF_API_XVMC
245.     AV_PIX_FMT_XVMC, ///< XVideo Motion Acceleration via common packet passing
246. #endif /* !FF_API_XVMC */
247.
248.     AV_PIX_FMT_NB,          ///< number of pixel formats, DO NOT USE THIS if you want to link with shared libav* because the number of
                                rmts might differ between versions
249.
250. #if FF_API_PIX_FMT
251. #include "old_pix_fmts.h"
252. #endif
253. };

```

FFmpeg有一个专门用于描述像素格式的结构体AVPixFmtDescriptor。该结构体的定义位于libavutil/pixdesc.h，如下所示。

```

1.  /**
2.   * Descriptor that unambiguously describes how the bits of a pixel are
3.   * stored in the up to 4 data planes of an image. It also stores the
4.   * subsampling factors and number of components.
5.   *
6.   * @note This is separate of the colorspace (RGB, YCbCr, YPbPr, JPEG-style YUV
7.   *       and all the YUV variants) AVPixFmtDescriptor just stores how values
8.   *       are stored not what these values represent.
9.   */
10. typedef struct AVPixFmtDescriptor{
11.     const char *name;
12.     uint8_t nb_components;          ///< The number of components each pixel has, (1-4)
13.
14.     /**
15.      * Amount to shift the luma width right to find the chroma width.
16.      * For YV12 this is 1 for example.
17.      * chroma_width = -((-luma_width) >> log2_chroma_w)
18.      * The note above is needed to ensure rounding up.
19.      * This value only refers to the chroma components.
20.      */
21.     uint8_t log2_chroma_w;          ///< chroma_width = -((-luma_width) >> log2_chroma_w)
22.
23.     /**
24.      * Amount to shift the luma height right to find the chroma height.
25.      * For YV12 this is 1 for example.
26.      * chroma_height = -((-luma_height) >> log2_chroma_h)
27.      * The note above is needed to ensure rounding up.
28.      * This value only refers to the chroma components.
29.      */
30.     uint8_t log2_chroma_h;
31.     uint8_t flags;
32.
33.     /**
34.      * Parameters that describe how pixels are packed.
35.      * If the format has 2 or 4 components, then alpha is last.
36.      * If the format has 1 or 2 components, then luma is 0.
37.      * If the format has 3 or 4 components,
38.      * if the RGB flag is set then 0 is red, 1 is green and 2 is blue;
39.      * otherwise 0 is luma, 1 is chroma-U and 2 is chroma-V.
40.      */
41.     AVComponentDescriptor comp[4];
42. }AVPixFmtDescriptor;

```

关于AVPixFmtDescriptor这个结构体不再做过多解释。它的定义比较简单，看注释就可以理解。通过av\_pix\_fmt\_desc\_get()可以获得指定像素格式的AVPixFmtDescriptor结构体。

```

1.  /**
2.   * @return a pixel format descriptor for provided pixel format or NULL if
3.   * this pixel format is unknown.
4.   */
5.  const AVPixFmtDescriptor *av_pix_fmt_desc_get(enum AVPixelFormat pix_fmt);

```

通过AVPixFmtDescriptor结构体可以获得不同像素格式的一些信息。例如下文中用到了av\_get\_bits\_per\_pixel()，通过该函数可以获得指定像素格式每个像素占用的比特数（Bit Per Pixel）。

```

1.  /**
2.   * Return the number of bits per pixel used by the pixel format
3.   * described by pixdesc. Note that this is not the same as the number
4.   * of bits per sample.
5.   *
6.   * The returned number of bits refers to the number of bits actually
7.   * used for storing the pixel information, that is padding bits are
8.   * not counted.
9.   */
10. int av_get_bits_per_pixel(const AVPixFmtDescriptor *pixdesc);

```

其他的API在这里不做过多记录。

## 图像拉伸

FFmpeg支持多种像素拉伸的方式。这些方式的定义位于libswscale\swscale.h中，如下所示。

```
[cpp]
1. #define SWS_FAST_BILINEAR 1
2. #define SWS_BILINEAR 2
3. #define SWS_BICUBIC 4
4. #define SWS_X 8
5. #define SWS_POINT 0x10
6. #define SWS_AREA 0x20
7. #define SWS_BICUBLIN 0x40
8. #define SWS_GAUSS 0x80
9. #define SWS_SINC 0x100
10. #define SWS_LANCZOS 0x200
11. #define SWS_SPLINE 0x400
```

其中SWS\_BICUBIC性能比较好；SWS\_FAST\_BILINEAR在性能和速度之间有一个比好好的平衡，而SWS\_POINT的效果比较差。

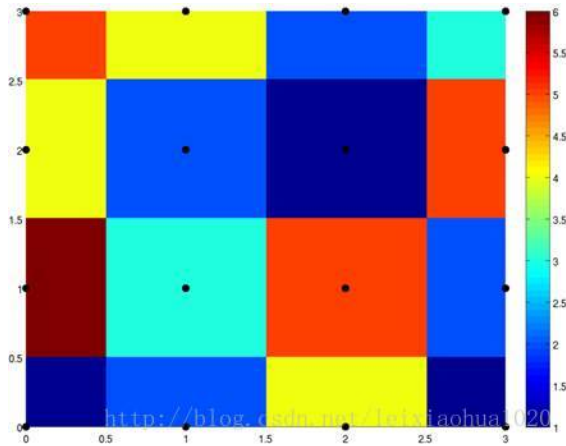
有关这些方法的评测可以参考文章：

《ffmpeg中的sws\_scale算法性能测试》

简单解释一下SWS\_BICUBIC、SWS\_BILINEAR和SWS\_POINT的原理。

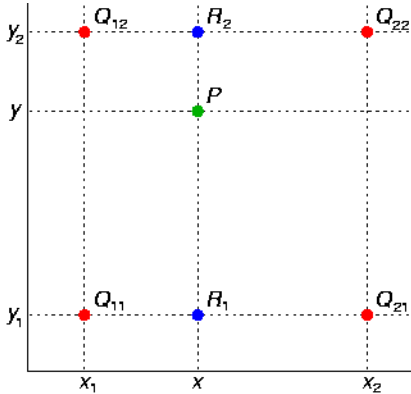
### SWS\_POINT (Nearest-neighbor interpolation, 邻域插值)

邻域插值可以简单说成“1个点确定插值的点”。例如当图像放大后，新的样点根据距离它最近的样点的值取得自己的值。换句话说就是简单拷贝附近距离它最近的样点的值。邻域插值是一种最基础的插值方法，速度最快，插值效果最不好，一般情况下不推荐使用。一般情况下使用邻域插值之后，画面会产生很多的“锯齿”。下图显示了4x4=16个彩色样点经过邻域插值后形成的图形。



### SWS\_BILINEAR (Bilinear interpolation, 双线性插值)

双线性插值可以简单说成“4个点确定插值的点”。它的计算过程可以简单用下图表示。图中绿色的P点是需要插值的点。首先通过Q11，Q21求得R1；Q12，Q22求得R2。然后根据R1，R2求得P。



其中求值的过程是一个简单的加权计算的过程。

设定Q11 = (x1, y1)，Q12 = (x1, y2)，Q21 = (x2, y1)，Q22 = (x2, y2)则各点的计算公式如下。

$$f(R_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21})$$

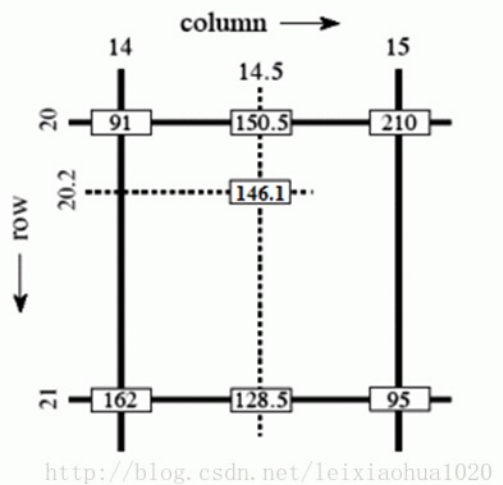


$$f(R_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22})$$

$$f(P) \approx \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2).$$

可以看出距离插值的点近一些的样点权值会大一些，远一些的样点权值要小一些。

下面看一个维基百科上的双线性插值的实例。该例子根据坐标为(20, 14)，(20, 15)，(21, 14)，(21, 15)的4个样点计算坐标为 (20.2, 14.5) 的插值点的值。



$$I_{20,14.5} = \frac{15-14.5}{15-14} \cdot 91 + \frac{14.5-14}{15-14} \cdot 210 = 150.5$$

$$I_{21,14.5} = \frac{15-14.5}{15-14} \cdot 162 + \frac{14.5-14}{15-14} \cdot 95 = 128.5$$

$$I_{20.2,14.5} = \frac{21-20.2}{21-20} \cdot 150.5 + \frac{20.2-20}{21-20} \cdot 128.5 = 146.1$$

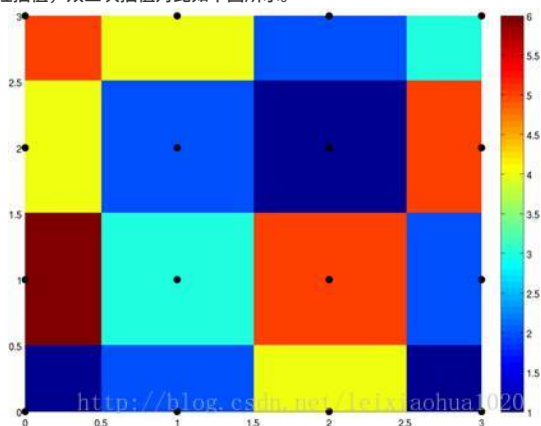
### SWS\_BICUBIC (Bicubic interpolation, 双三次插值)

双三次插值可以简单说成“16个点确定插值的点”。该插值算法比前两种算法复杂很多，插值后图像的质量也是最好的。有关它的插值方式比较复杂不再做过多记录。它的差值方法可以简单表述为下述公式。

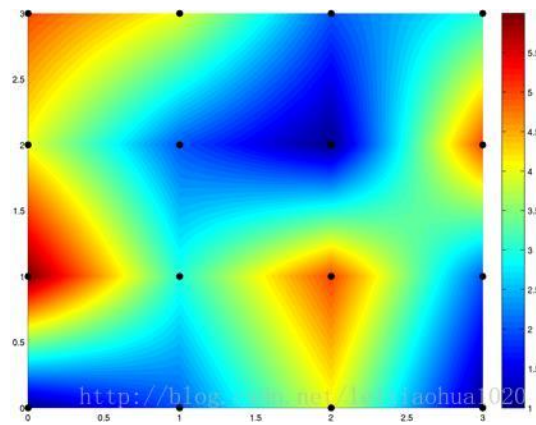
$$\sum_{i=0}^3 \sum_{j=0}^3 a_{ij} x^i y^j$$

其中 $a_{ij}$ 的过程依赖于插值数据的特性。

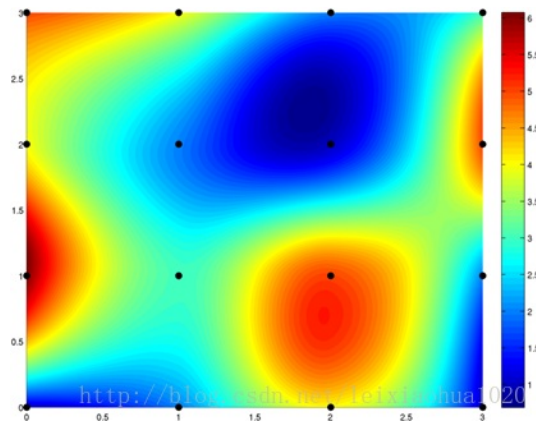
维基百科上使用同样的样点进行邻域插值，双线性插值，双三次插值对比如下图所示。



Nearest-neighbor interpolation, 邻域插值



Bilinear interpolation, 双线性插值



Bicubic interpolation, 双三次插值

## YUV像素取值范围

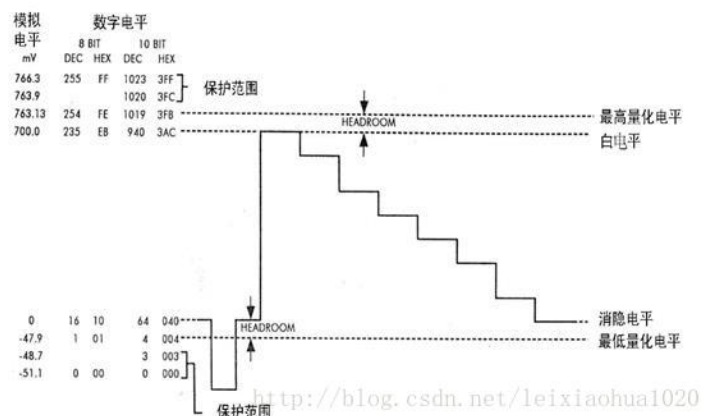
FFmpeg中可以通过使用`av_opt_set()`设置“src\_range”和“dst\_range”来设置输入和输出的YUV的取值范围。如果“dst\_range”字段设置为“1”的话，则代表输出的YUV的取值范围遵循“jpeg”标准；如果“dst\_range”字段设置为“0”的话，则代表输出的YUV的取值范围遵循“mpeg”标准。下面记录一下YUV的取值范围的概念。

与RGB每个像素点的每个分量取值范围为0-255不同（每个分量占8bit），YUV取值范围有两种：

- (1) 以Rec.601为代表（还包括BT.709 / BT.2020）的广播电视标准中，Y的取值范围是16-235，U、V的取值范围是16-240。FFmpeg中称之为“mpeg”范围。
- (2) 以JPEG为代表的标准中，Y、U、V的取值范围都是0-255。FFmpeg中称之为“jpeg”范围。

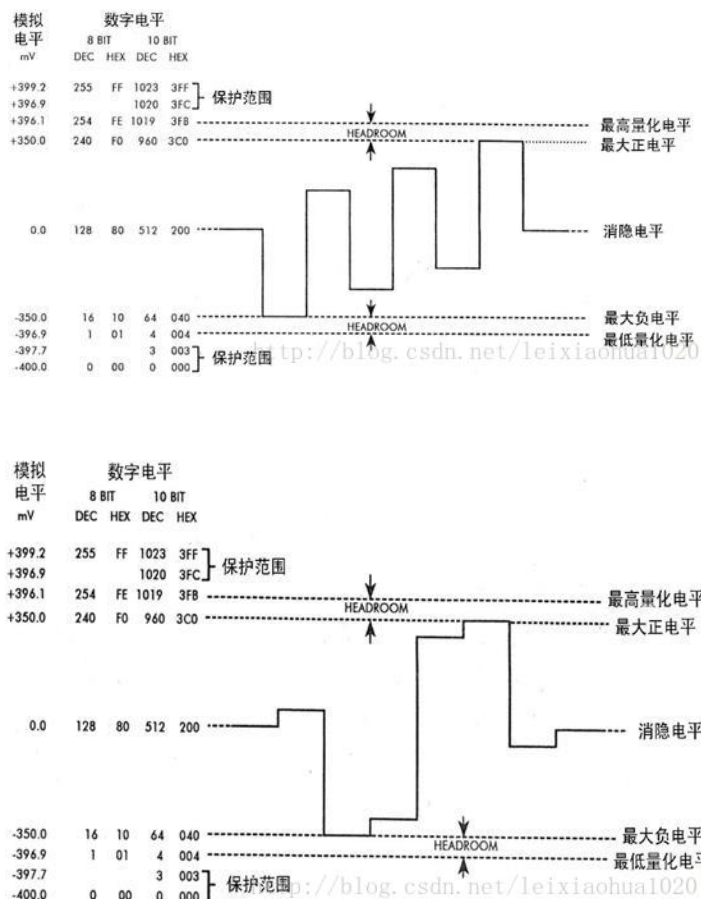
实际中最常见的是第1种取值范围的YUV（可以自己观察一下YUV的数据，会发现其中亮度分量没有取值为0、255这样的数值）。很多人在这个地方会有疑惑，为什么会去掉“两边”的取值呢？

在广播电视系统中不传输很低和很高的数值，实际上是为了防止信号变动造成过载，因而把这两边的数值作为“保护带”。下面这张图是数字电视中亮度信号量化后的电平分配图。从图中可以看出，对于8bit量化来说，信号的白电平为235，对应模拟电平为700mV；黑电平为16，对应模拟电平为0mV。信号上方的“保护带”取值范围是236至254，而信号下方的“保护带”取值范围是1-15。最边缘的0和255两个电平是保护电平，是不允许出现在数据流中的。与之类似，10bit量化的时候，白电平是 $235 \times 4 = 940$ ，黑电平是 $16 \times 4 = 64$ 。



下面两张图是数字电视中色度信号量化后的电平分配图。可以看出，色度最大正电平为240，对应模拟电平为+350mV；色度最大负电平为16，对应模拟电平为-350mV。需要注意的是，色度信号数字电平128对应的模拟电平是0mV。





## 色域

Libswscale支持色域的转换。有关色域的转换我目前还没有做太多的研究，仅记录一下目前最常见的三个标准中的色域：BT.601，BT.709，BT.2020。这三个标准中的色域逐渐增大。

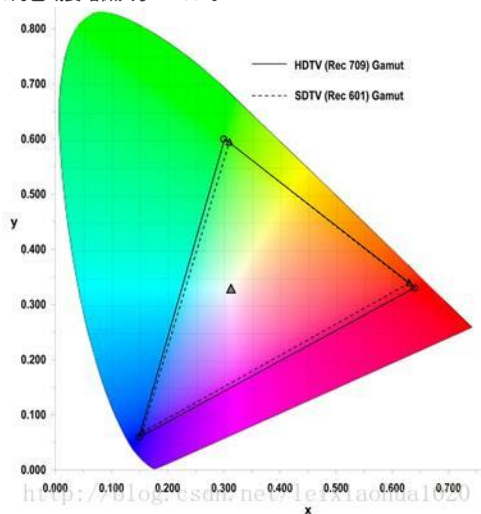
在这里先简单解释一下CIE 1931颜色空间。这个空间围绕的区域像一个“舌头”，其中包含了自然界所有的颜色。CIE 1931颜色空间中的横坐标是x，纵坐标是y，x、y、z满足如下关系：

$$x + y + z = 1$$

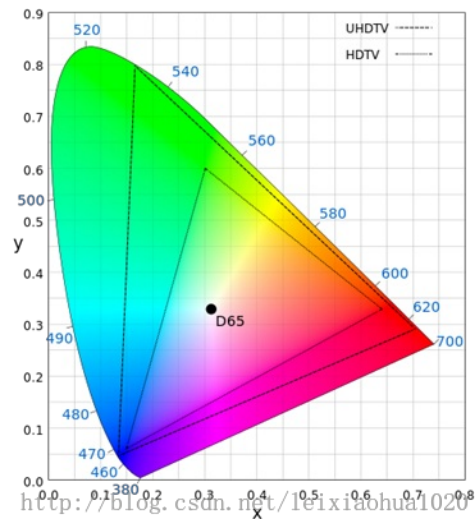
“舌头”的边缘叫做“舌形曲线”，代表着饱和度为100%的光谱色。“舌头”的中心点（1/3，1/3）对应着白色，饱和度为0。

受显示器件性能的限制，电视屏幕是无法重现所有的颜色的，尤其是位于“舌形曲线”上的100% 饱和度的光谱色一般情况下是无法显示出来的。因此电视屏幕只能根据其具体的荧光粉的配方，有选择性的显示一部分的颜色，这部分可以显示的颜色称为色域。下文分别比较标清电视、高清电视和超高清电视标准中规定的色域。可以看出随着技术的进步，色域的范围正变得越来越大。

**标清电视（SDTV）色域的规定源自于BT.601。高清电视（HDTV）色域的规定源自于BT.709。** 他们两个标准中的色域在CIE 1931颜色空间中的对比如下图所示。从图中可以看出，BT.709和BT.601色域差别不大，BT.709的色域要略微大于BT.601。



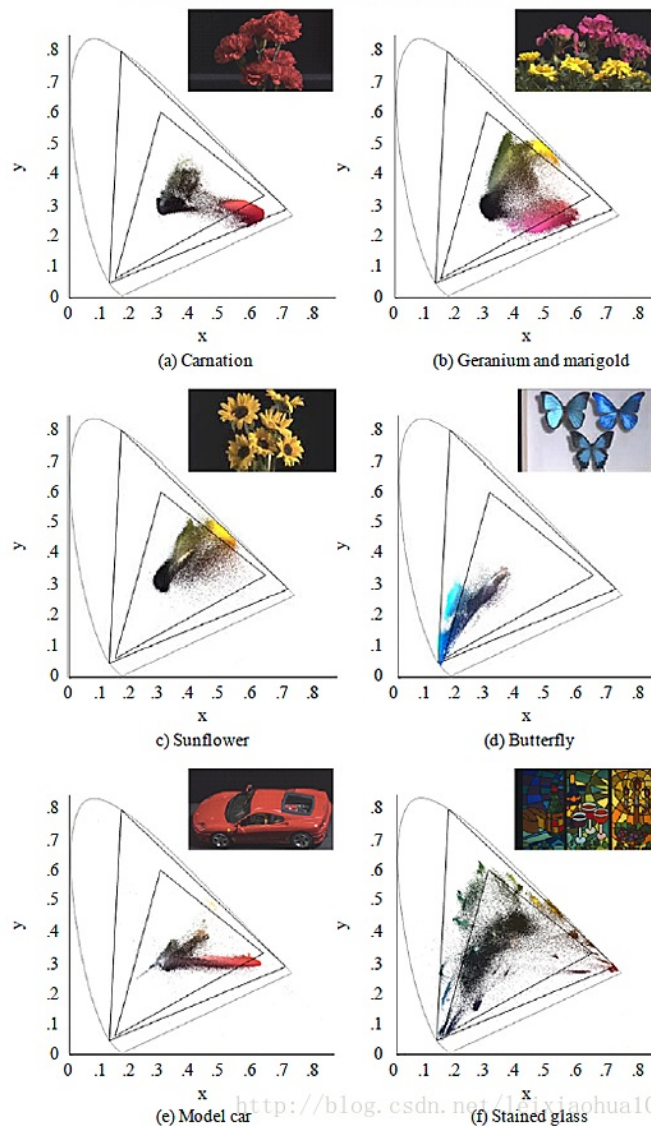
**超高清电视（UHDTV）色域的规定源自于BT.2020。** BT.2020和BT.709的色域在CIE 1931 颜色空间中的对比如下图所示。从图中可以看出，BT.2020的色域要远远大于BT.709。



从上面的对比也可以看出，对超高清电视（UHD TV）的显示器件的性能的要求更高了。这样超高清电视可以还原出一个更“真实”的世界。

下面这张图则使用实际的例子反映出色域范围大的重要性。图中的两个黑色三角形分别标识出了BT.709（小三角形）和BT.2020（大三角形）标准中的色域。从图中可以看出，如果使用色域较小的显示设备显示图片的话，将会损失掉很多颜色。

Colour distribution of objects on the x-y chromaticity coordinates  
(Inner triangle: HDTV primaries, Outer triangle: UHD TV primaries)



## 源代码

本示例程序包含一个输入和一个输出，实现了从输入图像格式（YUV420P）到输出图像格式（RGB24）之间的转换；同时将输入视频的分辨率从480x272拉伸为1280x720。

[cpp]  

```
1.  /**
2.   * 最简单的基于FFmpeg的Swscale示例
3.   *  Simplest FFmpeg Swscale
4.   *
5.   *  雷霄骅 Lei Xiaohua
6.   *  leixiaohua1020@126.com
7.   *  中国传媒大学/数字电视技术
8.   *  Communication University of China / Digital TV Technology
9.   *  http://blog.csdn.net/leixiaohua1020
10.  *
11.  *  本程序使用libswscale对像素数据进行缩放转换等处理。
12.  *  它中实现了YUV420P格式转换为RGB24格式,
13.  *  同时将分辨率从480x272拉伸为1280x720
14.  *  它是最简单的libswscale的教程。
15.  *
16.  *  This software uses libswscale to scale / convert pixels.
17.  *  It convert YUV420P format to RGB24 format,
18.  *  and changes resolution from 480x272 to 1280x720.
19.  *  It's the simplest tutorial about libswscale.
20.  */
21.
22. #include <stdio.h>
23.
24. #define __STDC_CONSTANT_MACROS
25.
26. #ifdef _WIN32
27. //Windows
28. extern "C"
29. {
30. #include "libswscale/swscale.h"
31. #include "libavutil/opt.h"
32. #include "libavutil/imgutils.h"
33. };
34. #else
35. //Linux...
36. #ifdef __cplusplus
37. extern "C"
38. {
39. #endif
40. #include <libswscale/swscale.h>
41. #include <libavutil/opt.h>
42. #include <libavutil/imgutils.h>
43. #ifdef __cplusplus
44. };
45. #endif
46. #endif
47.
48.
49. int main(int argc, char* argv[])
50. {
51.     //Parameters
52.     FILE *src_file =fopen("sintel_480x272_yuv420p.yuv", "rb");
53.     const int src_w=480,src_h=272;
54.     AVPixelFormat src_pixfmt=AV_PIX_FMT_YUV420P;
55.
56.     int src_bpp=av_get_bits_per_pixel(av_pix_fmt_desc_get(src_pixfmt));
57.
58.     FILE *dst_file = fopen("sintel_1280x720_rgb24.rgb", "wb");
59.     const int dst_w=1280,dst_h=720;
60.     AVPixelFormat dst_pixfmt=AV_PIX_FMT_RGB24;
61.     int dst_bpp=av_get_bits_per_pixel(av_pix_fmt_desc_get(dst_pixfmt));
62.
63.     //Structures
64.     uint8_t *src_data[4];
65.     int src_linesize[4];
66.
67.     uint8_t *dst_data[4];
68.     int dst_linesize[4];
69.
70.     int rescale_method=SWS_BICUBIC;
71.     struct SwsContext *img_convert_ctx;
72.     uint8_t *temp_buffer=(uint8_t *)malloc(src_w*src_h*src_bpp/8);
73.
74.     int frame_idx=0;
75.     int ret=0;
76.     ret= av_image_alloc(src_data, src_linesize,src_w, src_h, src_pixfmt, 1);
77.     if (ret< 0) {
78.         printf( "Could not allocate source image\n");
79.         return -1;
80.     }
81.     ret = av_image_alloc(dst_data, dst_linesize,dst_w, dst_h, dst_pixfmt, 1);
82.     if (ret< 0) {
83.         printf( "Could not allocate destination image\n");
84.         return -1;
85.     }
86.     //-----
87.     //Init Method 1
88.     img_convert_ctx =sws_alloc_context();
89.     //Show AVOption
90.     av_opt_show2(img_convert_ctx, stdout, AV_OPT_FLAG_VIDEO_PARAM |
```

```

90.     av_opt_show2(img_convert_ctx, stdout, AV_OPT_FLAG_VIDEO_PARAM, 0);
91.     //Set Value
92.     av_opt_set_int(img_convert_ctx, "sws_flags", SWS_BICUBIC|SWS_PRINT_INFO, 0);
93.     av_opt_set_int(img_convert_ctx, "srcw", src_w, 0);
94.     av_opt_set_int(img_convert_ctx, "srch", src_h, 0);
95.     av_opt_set_int(img_convert_ctx, "src_format", src_pixfmt, 0);
96.     // '0' for MPEG (Y:0-235); '1' for JPEG (Y:0-255)
97.     av_opt_set_int(img_convert_ctx, "src_range", 1, 0);
98.     av_opt_set_int(img_convert_ctx, "dstw", dst_w, 0);
99.     av_opt_set_int(img_convert_ctx, "dsth", dst_h, 0);
100.    av_opt_set_int(img_convert_ctx, "dst_format", dst_pixfmt, 0);
101.    av_opt_set_int(img_convert_ctx, "dst_range", 1, 0);
102.    sws_init_context(img_convert_ctx, NULL, NULL);
103.
104.    //Init Method 2
105.    //img_convert_ctx = sws_getContext(src_w, src_h, src_pixfmt, dst_w, dst_h, dst_pixfmt,
106.    // rescale_method, NULL, NULL, NULL);
107.    //-----
108.    /*
109.    //Colorspace
110.    ret=sws_setColorspaceDetails(img_convert_ctx, sws_getCoefficients(SWS_CS_ITU601), 0,
111.    sws_getCoefficients(SWS_CS_ITU709), 0,
112.    0, 1 <= 16, 1 <= 16);
113.    if (ret==-1) {
114.        printf( "Colorspace not support.\n");
115.        return -1;
116.    }
117.    */
118.    while(1)
119.    {
120.        if (fread(temp_buffer, 1, src_w*src_h*src_bpp/8, src_file) != src_w*src_h*src_bpp/8){
121.            break;
122.        }
123.
124.        switch(src_pixfmt){
125.        case AV_PIX_FMT_GRAY8:{
126.            memcpy(src_data[0], temp_buffer, src_w*src_h);
127.            break;
128.        }
129.        case AV_PIX_FMT_YUV420P:{
130.            memcpy(src_data[0], temp_buffer, src_w*src_h);           //Y
131.            memcpy(src_data[1], temp_buffer+src_w*src_h, src_w*src_h/4); //U
132.            memcpy(src_data[2], temp_buffer+src_w*src_h*5/4, src_w*src_h/4); //V
133.            break;
134.        }
135.        case AV_PIX_FMT_YUV422P:{
136.            memcpy(src_data[0], temp_buffer, src_w*src_h);           //Y
137.            memcpy(src_data[1], temp_buffer+src_w*src_h, src_w*src_h/2); //U
138.            memcpy(src_data[2], temp_buffer+src_w*src_h*3/2, src_w*src_h/2); //V
139.            break;
140.        }
141.        case AV_PIX_FMT_YUV444P:{
142.            memcpy(src_data[0], temp_buffer, src_w*src_h);           //Y
143.            memcpy(src_data[1], temp_buffer+src_w*src_h, src_w*src_h); //U
144.            memcpy(src_data[2], temp_buffer+src_w*src_h*2, src_w*src_h); //V
145.            break;
146.        }
147.        case AV_PIX_FMT_YUYV422:{
148.            memcpy(src_data[0], temp_buffer, src_w*src_h*2);           //Packed
149.            break;
150.        }
151.        case AV_PIX_FMT_RGB24:{
152.            memcpy(src_data[0], temp_buffer, src_w*src_h*3);           //Packed
153.            break;
154.        }
155.        default:{
156.            printf("Not Support Input Pixel Format.\n");
157.            break;
158.        }
159.    }
160.
161.    sws_scale(img_convert_ctx, src_data, src_linesize, 0, src_h, dst_data, dst_linesize);
162.    printf("Finish process frame %5d\n", frame_idx);
163.    frame_idx++;
164.
165.    switch(dst_pixfmt){
166.    case AV_PIX_FMT_GRAY8:{
167.        fwrite(dst_data[0], 1, dst_w*dst_h, dst_file);
168.        break;
169.    }
170.    case AV_PIX_FMT_YUV420P:{
171.        fwrite(dst_data[0], 1, dst_w*dst_h, dst_file);           //Y
172.        fwrite(dst_data[1], 1, dst_w*dst_h/4, dst_file);         //U
173.        fwrite(dst_data[2], 1, dst_w*dst_h/4, dst_file);         //V
174.        break;
175.    }
176.    case AV_PIX_FMT_YUV422P:{
177.        fwrite(dst_data[0], 1, dst_w*dst_h, dst_file);           //Y
178.        fwrite(dst_data[1], 1, dst_w*dst_h/2, dst_file);         //U
179.        fwrite(dst_data[2], 1, dst_w*dst_h/2, dst_file);         //V
180.        break;
181.    }

```

```

182.         case AV_PIX_FMT_YUV444P:{
183.             fwrite(dst_data[0],1,dst_w*dst_h,dst_file);           //Y
184.             fwrite(dst_data[1],1,dst_w*dst_h,dst_file);           //U
185.             fwrite(dst_data[2],1,dst_w*dst_h,dst_file);           //V
186.             break;
187.         }
188.         case AV_PIX_FMT_YUV422P:{
189.             fwrite(dst_data[0],1,dst_w*dst_h*2,dst_file);         //Packed
190.             break;
191.         }
192.         case AV_PIX_FMT_RGB24:{
193.             fwrite(dst_data[0],1,dst_w*dst_h*3,dst_file);         //Packed
194.             break;
195.         }
196.         default:{
197.             printf("Not Support Output Pixel Format.\n");
198.             break;
199.         }
200.     }
201. }
202.
203. sws_freeContext(img_convert_ctx);
204.
205. free(temp_buffer);
206. fclose(dst_file);
207. av_freep(&src_data[0]);
208. av_freep(&dst_data[0]);
209.
210. return 0;
211. }

```

## 运行结果

程序的输入为一个名称为"sintel\_480x272\_yuv420p.yuv"的视频。该视频像素格式是YUV420P，分辨率为480x272。



程序的输出为一个名称为"sintel\_1280x720\_rgb24.rgb"的视频。该视频像素格式是RGB24，分辨率为1280x720。



## 下载

Simplest FFmpeg Swscale

[项目主页](#)

SourceForge：<https://sourceforge.net/projects/simplestffmpegswscale/>

Github：[https://github.com/leixiaohua1020/simplest\\_ffmpeg\\_swscale](https://github.com/leixiaohua1020/simplest_ffmpeg_swscale)

开源中国：[http://git.oschina.net/leixiaohua1020/simplest\\_ffmpeg\\_swscale](http://git.oschina.net/leixiaohua1020/simplest_ffmpeg_swscale)

CSDN下载地址：<http://download.csdn.net/detail/leixiaohua1020/8292175>

本教程是最简单的基于FFmpeg的libswscale进行像素处理的教程。它包含了两个工程：

**simplest\_ffmpeg\_swscale: 最简单的libswscale的教程。**

simplest\_pic\_gen: 生成各种测试图片的工具。

## 更新-1.1 (2015.2.13)=====

这次考虑到了跨平台的要求，调整了源代码。经过这次调整之后，源代码可以在以下平台编译通过：

VC++：打开sln文件即可编译，无需配置。

cl.exe：打开compile\_cl.bat即可命令行下使用cl.exe进行编译，注意可能需要按照VC的安装路径调整脚本里面的参数。编译命令如下。

```
[plain]
1.  ::VS2010 Environment
2.  call "D:\Program Files\Microsoft Visual Studio 10.0\VC\vcvarsall.bat"
3.  ::include
4.  @set INCLUDE=include;%INCLUDE%
5.  ::lib
6.  @set LIB=lib;%LIB%
7.  ::compile and link
8.  cl simplest_ffmpeg_swscale.cpp /link swscale.lib avutil.lib /OPT:NOREF
```

MinGW：MinGW命令行下运行compile\_mingw.sh即可使用MinGW的g++进行编译。编译命令如下。

```
[plain]
1.  g++ simplest_ffmpeg_swscale.cpp -g -o simplest_ffmpeg_swscale.exe \
2.  -I /usr/local/include -L /usr/local/lib -lswscale -lavutil
```

GCC：Linux或者MacOS命令行下运行compile\_gcc.sh即可使用GCC进行编译。编译命令如下。

```
[plain]
1.  gcc simplest_ffmpeg_swscale.cpp -g -o simplest_ffmpeg_swscale.out -I /usr/local/include -L /usr/local/lib \
2.  -lswscale -lavutil
```

PS：相关的编译命令已经保存到了工程文件夹中

CSDN下载地址：<http://download.csdn.net/detail/leixiaohua1020/8445671>

SourceForge上已经更新。

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/42134965>

文章标签：[libswscale](#) [拉伸](#) [yuv](#) [ffmpeg](#) [色域](#)

个人分类：[FFMPEG](#) [我的开源项目](#)

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