

原 FFMpeg源代码简单分析：libswscale的sws_scale()

2015年03月17日 20:02:33 阅读数：25908

=====

FFmpeg的库函数源代码分析文章列表：

【架构图】

[FFmpeg 源代码结构图 - 解码](#)

[FFmpeg 源代码结构图 - 编码](#)

【通用】

[FFmpeg 源代码简单分析：av_register_all\(\)](#)

[FFmpeg 源代码简单分析：avcodec_register_all\(\)](#)

[FFmpeg 源代码简单分析：内存的分配和释放（av_malloc\(\)、av_free\(\)等）](#)

[FFmpeg 源代码简单分析：常见结构体的初始化和销毁（AVFormatContext，AVFrame等）](#)

[FFmpeg 源代码简单分析：avio_open2\(\)](#)

[FFmpeg 源代码简单分析：av_find_decoder\(\)和av_find_encoder\(\)](#)

[FFmpeg 源代码简单分析：avcodec_open2\(\)](#)

[FFmpeg 源代码简单分析：avcodec_close\(\)](#)

【解码】

[图解FFMPEG 打开媒体的函数 avformat_open_input](#)

[FFmpeg 源代码简单分析：avformat_open_input\(\)](#)

[FFmpeg 源代码简单分析：avformat_find_stream_info\(\)](#)

[FFmpeg 源代码简单分析：av_read_frame\(\)](#)

[FFmpeg 源代码简单分析：avcodec_decode_video2\(\)](#)

[FFmpeg 源代码简单分析：avformat_close_input\(\)](#)

【编码】

[FFmpeg 源代码简单分析：avformat_alloc_output_context2\(\)](#)

[FFmpeg 源代码简单分析：avformat_write_header\(\)](#)

[FFmpeg 源代码简单分析：avcodec_encode_video\(\)](#)

[FFmpeg 源代码简单分析：av_write_frame\(\)](#)

[FFmpeg 源代码简单分析：av_write_trailer\(\)](#)

【其它】

[FFmpeg 源代码简单分析：日志输出系统（av_log\(\)等）](#)

[FFmpeg 源代码简单分析：结构体成员管理系统 -AVClass](#)

[FFmpeg 源代码简单分析：结构体成员管理系统 -AVOption](#)

[FFmpeg 源代码简单分析：libswscale 的 sws_getContext\(\)](#)

[FFmpeg 源代码简单分析：libswscale 的 sws_scale\(\)](#)

[FFmpeg 源代码简单分析：libavdevice 的 avdevice_register_all\(\)](#)

[FFmpeg 源代码简单分析：libavdevice 的 gdigrab](#)

【脚本】

[FFmpeg 源代码简单分析：makefile](#)

[FFmpeg 源代码简单分析：configure](#)

【H.264】

[FFmpeg 的 H.264 解码器源代码简单分析：概述](#)

=====

本文继续上一篇文章《 [FFmpeg源代码分析:sws_getContext\(\)](#) 》的内容，简单分析FFmpeg的图像处理（缩放，YUV/RGB格式转换）类库libswscale中的sws_scale()函数。libswscale是一个主要用于处理图片像素数据的类库。可以完成图片像素格式的转换，图片的拉伸等工作。有关libswscale的使用可以参考文章：

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

该类库常用的函数数量很少，一般情况下就3个：

sws_getContext()：初始化一个SwsContext。

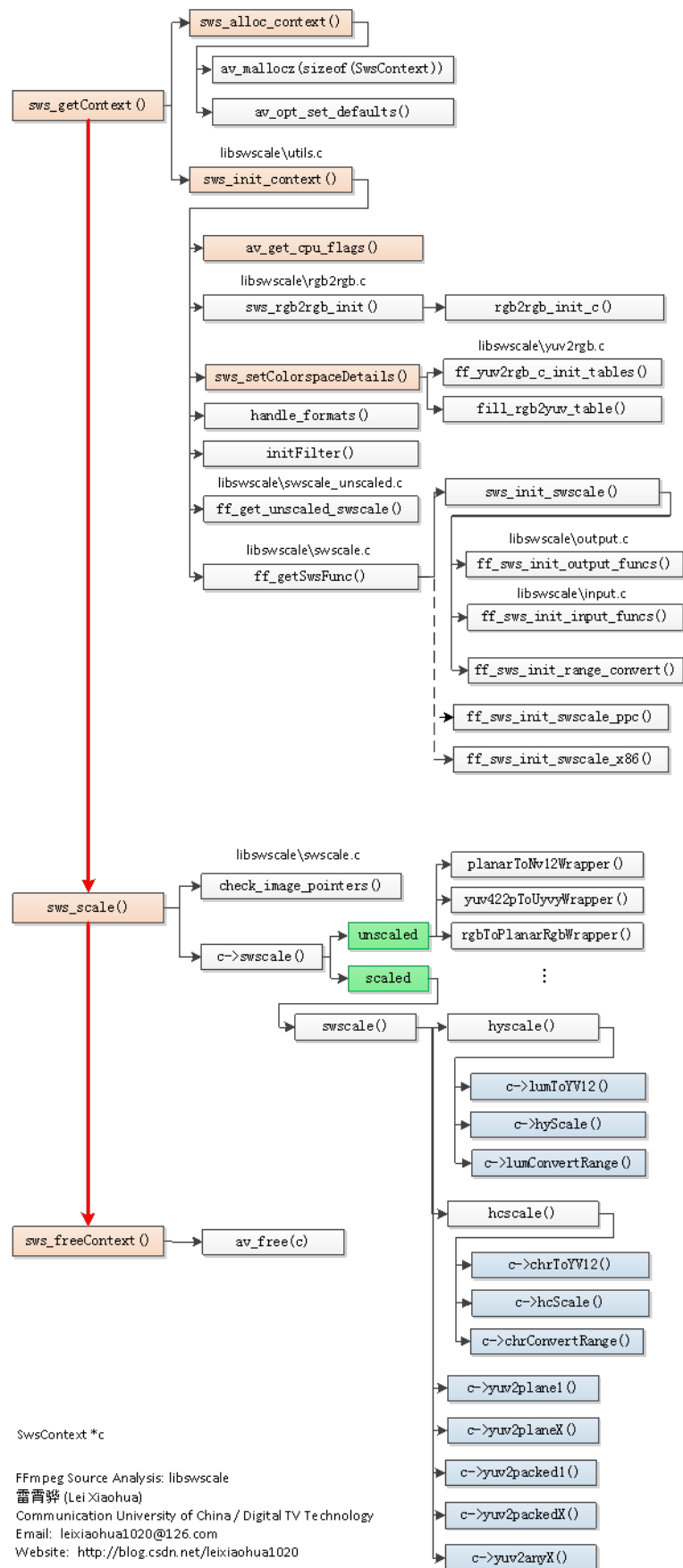
sws_scale()：处理图像数据。

sws_freeContext()：释放一个SwsContext。

在分析sws_scale()的源代码之前，先简单回顾一下上篇文章中分析得到的两张图。

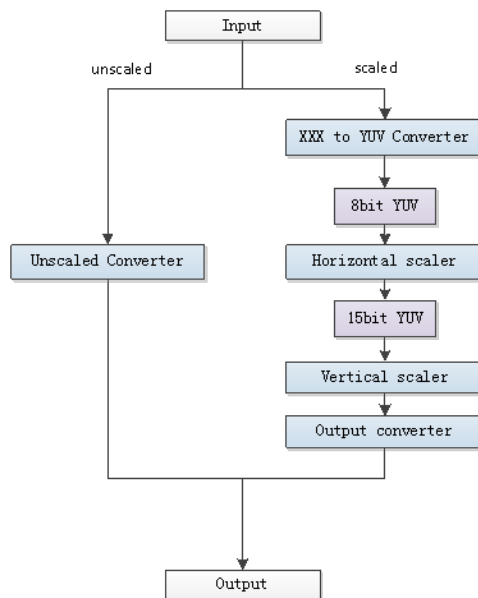
函数调用结构图

分析得到的libswscale的函数调用关系如下图所示。



Libswscale处理数据流程

Libswscale处理像素数据的流程可以概括为下图。



雷霄骅 (Lei Xiaohua)
Email: leixiaohua1020@126.com
Website: <http://blog.csdn.net/leixiaohua1020>

从图中可以看出，libswscale处理数据有两条最主要的方式：unscaled和scaled。unscaled用于处理不需要拉伸的像素数据（属于比较特殊的情况），scaled用于处理需要拉伸的像素数据。Unscaled只需要对图像像素格式进行转换；而Scaled则除了对像素格式进行转换之外，还需要对图像进行缩放。Scaled方式可以分成以下几个步骤：

XXX to YUV Converter：首相将数据像素数据转换为8bitYUV格式；

Horizontal scaler：水平拉伸图像，并且转换为15bitYUV；

Vertical scaler：垂直拉伸图像；

Output converter：转换为输出像素格式。

sws_scale()

sws_scale()是用于转换像素的函数。它的声明位于libswscale\swscale.h，如下所示。

```

1.  /**
2.   * Scale the image slice in srcSlice and put the resulting scaled
3.   * slice in the image in dst. A slice is a sequence of consecutive
4.   * rows in an image.
5.   *
6.   * Slices have to be provided in sequential order, either in
7.   * top-bottom or bottom-top order. If slices are provided in
8.   * non-sequential order the behavior of the function is undefined.
9.   *
10.  * @param c      the scaling context previously created with
11.  *                sws_getContext()
12.  * @param srcSlice the array containing the pointers to the planes of
13.  *                 the source slice
14.  * @param srcStride the array containing the strides for each plane of
15.  *                  the source image
16.  * @param srcSliceY the position in the source image of the slice to
17.  *                  process, that is the number (counted starting from
18.  *                  zero) in the image of the first row of the slice
19.  * @param srcSliceH the height of the source slice, that is the number
20.  *                  of rows in the slice
21.  * @param dst      the array containing the pointers to the planes of
22.  *                  the destination image
23.  * @param dstStride the array containing the strides for each plane of
24.  *                  the destination image
25.  * @return         the height of the output slice
26.  */
27. int sws_scale(struct SwsContext *c, const uint8_t *const srcSlice[],
28.               const int srcStride[], int srcSliceY, int srcSliceH,
29.               uint8_t *const dst[], const int dstStride[]);
  
```

sws_scale()的定义位于libswscale\swscale.c，如下所示。

```

1.  /**
2.   * swscale wrapper, so we don't need to export the SwsContext
  
```

```

2. //swscale wrapper, so we don't need to export the SwsContext.
3. * Assumes planar YUV to be in YUV order instead of YVU.
4. */
5. int sws_scale(struct SwsContext *c,
6.               const uint8_t * const srcSlice[],
7.               const int srcStride[], int srcSliceY,
8.               int srcSliceH, uint8_t *const dst[],
9.               const int dstStride[])
10. {
11.     int i, ret;
12.     const uint8_t *src2[4];
13.     uint8_t *dst2[4];
14.     uint8_t *rgb0_tmp = NULL;
15.     //检查输入参数
16.     if (!srcStride || !dstStride || !dst || !srcSlice) {
17.         av_log(c, AV_LOG_ERROR, "One of the input parameters to sws_scale() is NULL, please check the calling code\n");
18.         return 0;
19.     }
20.     if (c->cascaded_context[0] && srcSliceY == 0 && srcSliceH == c->cascaded_context[0]->srcH) {
21.         ret = sws_scale(c->cascaded_context[0],
22.                         srcSlice, srcStride, srcSliceY, srcSliceH,
23.                         c->cascaded_tmp, c->cascaded_tmpStride);
24.         if (ret < 0)
25.             return ret;
26.         ret = sws_scale(c->cascaded_context[1],
27.                         (const uint8_t * const *)c->cascaded_tmp, c->cascaded_tmpStride, 0, c->cascaded_context[0]->dstH,
28.                         dst, dstStride);
29.         return ret;
30.     }
31.
32.     memcpy(src2, srcSlice, sizeof(src2));
33.     memcpy(dst2, dst, sizeof(dst2));
34.
35.     // do not mess up sliceDir if we have a "trailing" 0-size slice
36.     if (srcSliceH == 0)
37.         return 0;
38.     //检查
39.     if (!check_image_pointers(srcSlice, c->srcFormat, srcStride)) {
40.         av_log(c, AV_LOG_ERROR, "bad src image pointers\n");
41.         return 0;
42.     }
43.     if (!check_image_pointers((const uint8_t* const*)dst, c->dstFormat, dstStride)) {
44.         av_log(c, AV_LOG_ERROR, "bad dst image pointers\n");
45.         return 0;
46.     }
47.
48.     if (c->sliceDir == 0 && srcSliceY != 0 && srcSliceY + srcSliceH != c->srcH) {
49.         av_log(c, AV_LOG_ERROR, "Slices start in the middle!\n");
50.         return 0;
51.     }
52.     if (c->sliceDir == 0) {
53.         if (srcSliceY == 0) c->sliceDir = 1; else c->sliceDir = -1;
54.     }
55.     //使用调色板palette的特殊处理? 应该不常见
56.     if (usePal(c->srcFormat)) {
57.         for (i = 0; i < 256; i++) {
58.             int r, g, b, y, u, v, a = 0xff;
59.             if (c->srcFormat == AV_PIX_FMT_PAL8) {
60.                 uint32_t p = ((const uint32_t *) (srcSlice[1]))[i];
61.                 a = (p >> 24) & 0xFF;
62.                 r = (p >> 16) & 0xFF;
63.                 g = (p >> 8) & 0xFF;
64.                 b = p & 0xFF;
65.             } else if (c->srcFormat == AV_PIX_FMT_RGB8) {
66.                 r = (i >> 5) * 36;
67.                 g = ((i >> 2) & 7) * 36;
68.                 b = (i & 3) * 85;
69.             } else if (c->srcFormat == AV_PIX_FMT_BGR8) {
70.                 b = (i >> 6) * 85;
71.                 g = ((i >> 3) & 7) * 36;
72.                 r = (i & 7) * 36;
73.             } else if (c->srcFormat == AV_PIX_FMT_RGB4_BYTE) {
74.                 r = (i >> 3) * 255;
75.                 g = ((i >> 1) & 3) * 85;
76.                 b = (i & 1) * 255;
77.             } else if (c->srcFormat == AV_PIX_FMT_GRAY8 || c->srcFormat == AV_PIX_FMT_GRAY8A) {
78.                 r = g = b = i;
79.             } else {
80.                 av_assert1(c->srcFormat == AV_PIX_FMT_BGR4_BYTE);
81.                 b = (i >> 3) * 255;
82.                 g = ((i >> 1) & 3) * 85;
83.                 r = (i & 1) * 255;
84.             }
85.         }
86.         #define RGB2YUV_SHIFT 15
87.         #define BY (int) (0.114 * 219 / 255 * (1 << RGB2YUV_SHIFT) + 0.5))
88.         #define BV (int) (0.081 * 224 / 255 * (1 << RGB2YUV_SHIFT) + 0.5))
89.         #define BU (int) (0.500 * 224 / 255 * (1 << RGB2YUV_SHIFT) + 0.5))
90.         #define GY (int) (0.587 * 219 / 255 * (1 << RGB2YUV_SHIFT) + 0.5))
91.         #define GV (int) (0.419 * 224 / 255 * (1 << RGB2YUV_SHIFT) + 0.5))
92.         #define GU (int) (0.331 * 224 / 255 * (1 << RGB2YUV_SHIFT) + 0.5))
93.         #define RY (int) (0.299 * 219 / 255 * (1 << RGB2YUV_SHIFT) + 0.5))
94.         #define RV (int) (0.500 * 224 / 255 * (1 << RGB2YUV_SHIFT) + 0.5))

```

```

94. #define RU (- (int) (0.169 * 224 / 255 * (1 << RGB2YUV_SHIFT) + 0.5))
95.
96.     y = av_clip_uint8((RY * r + GY * g + BY * b + ( 33 << (RGB2YUV_SHIFT - 1))) >> RGB2YUV_SHIFT);
97.     u = av_clip_uint8((RU * r + GU * g + BU * b + (257 << (RGB2YUV_SHIFT - 1))) >> RGB2YUV_SHIFT);
98.     v = av_clip_uint8((RV * r + GV * g + BV * b + (257 << (RGB2YUV_SHIFT - 1))) >> RGB2YUV_SHIFT);
99.     c->pal_yuv[i] = y + (u<<8) + (v<<16) + ((unsigned)a<<24);
100.
101.     switch (c->dstFormat) {
102.     case AV_PIX_FMT_BGR32:
103.     #if !HAVE_BIGENDIAN
104.         case AV_PIX_FMT_RGB24:
105.     #endif
106.         c->pal_rgb[i] = r + (g<<8) + (b<<16) + ((unsigned)a<<24);
107.         break;
108.     case AV_PIX_FMT_BGR32_1:
109.     #if HAVE_BIGENDIAN
110.         case AV_PIX_FMT_BGR24:
111.     #endif
112.         c->pal_rgb[i] = a + (r<<8) + (g<<16) + ((unsigned)b<<24);
113.         break;
114.     case AV_PIX_FMT_RGB32_1:
115.     #if HAVE_BIGENDIAN
116.         case AV_PIX_FMT_RGB24:
117.     #endif
118.         c->pal_rgb[i] = a + (b<<8) + (g<<16) + ((unsigned)r<<24);
119.         break;
120.     case AV_PIX_FMT_RGB32:
121.     #if !HAVE_BIGENDIAN
122.         case AV_PIX_FMT_BGR24:
123.     #endif
124.     default:
125.         c->pal_rgb[i] = b + (g<<8) + (r<<16) + ((unsigned)a<<24);
126.     }
127. }
128. }
129. //Alpha的特殊处理?
130. if (c->src0Alpha && !c->dst0Alpha && isALPHA(c->dstFormat)) {
131.     uint8_t *base;
132.     int x,y;
133.     rgb0_tmp = av_malloc(FFABS(srcStride[0]) * srcSliceH + 32);
134.     if (!rgb0_tmp)
135.         return AERROR(ENOMEM);
136.
137.     base = srcStride[0] < 0 ? rgb0_tmp - srcStride[0] * (srcSliceH-1) : rgb0_tmp;
138.     for (y=0; y<srcSliceH; y++){
139.         memcpy(base + srcStride[0]*y, src2[0] + srcStride[0]*y, 4*c->srcW);
140.         for (x=c->src0Alpha-1; x<4*c->srcW; x+=4) {
141.             base[ srcStride[0]*y + x] = 0xFF;
142.         }
143.     }
144.     src2[0] = base;
145. }
146. //XYZ的特殊处理?
147. if (c->srcXYZ && !(c->dstXYZ && c->srcW==c->dstW && c->srcH==c->dstH)) {
148.     uint8_t *base;
149.     rgb0_tmp = av_malloc(FFABS(srcStride[0]) * srcSliceH + 32);
150.     if (!rgb0_tmp)
151.         return AERROR(ENOMEM);
152.
153.     base = srcStride[0] < 0 ? rgb0_tmp - srcStride[0] * (srcSliceH-1) : rgb0_tmp;
154.
155.     xyz12Torgb48(c, (uint16_t*)base, (const uint16_t*)src2[0], srcStride[0]/2, srcSliceH);
156.     src2[0] = base;
157. }
158.
159. if (!srcSliceY && (c->flags & SWS_BITEXACT) && c->dither == SWS_DITHER_ED && c->dither_error[0])
160.     for (i = 0; i < 4; i++)
161.         memset(c->dither_error[i], 0, sizeof(c->dither_error[0][0]) * (c->dstW+2));
162.
163.
164. // copy strides, so they can safely be modified
165. // sliceDir: 1 = top-to-bottom; -1 = bottom-to-top;
166. if (c->sliceDir == 1) {
167.     // slices go from top to bottom
168.     int srcStride2[4] = { srcStride[0], srcStride[1], srcStride[2],
169.                          srcStride[3] };
170.     int dstStride2[4] = { dstStride[0], dstStride[1], dstStride[2],
171.                          dstStride[3] };
172.
173.     reset_ptr(src2, c->srcFormat);
174.     reset_ptr((void*)dst2, c->dstFormat);
175.
176.     /* reset slice direction at end of frame */
177.     if (srcSliceY + srcSliceH == c->srcH)
178.         c->sliceDir = 0;
179.     //关键:调用
180.     ret = c->swscale(c, src2, srcStride2, srcSliceY, srcSliceH, dst2,
181.                     dstStride2);
182. } else {
183.     // slices go from bottom to top => we flip the image internally
184.     int srcStride2[4] = { -srcStride[0], -srcStride[1], -srcStride[2],

```

```

185.         -srcStride[3] };
186.     int dstStride2[4] = { -dstStride[0], -dstStride[1], -dstStride[2],
187.         -dstStride[3] };
188.
189.     src2[0] += (srcSliceH - 1) * srcStride[0];
190.     if (!usePal(c->srcFormat))
191.         src2[1] += ((srcSliceH >> c->chrSrcVSubSample) - 1) * srcStride[1];
192.     src2[2] += ((srcSliceH >> c->chrSrcVSubSample) - 1) * srcStride[2];
193.     src2[3] += (srcSliceH - 1) * srcStride[3];
194.     dst2[0] += ( c->dstH - 1) * dstStride[0];
195.     dst2[1] += ((c->dstH >> c->chrDstVSubSample) - 1) * dstStride[1];
196.     dst2[2] += ((c->dstH >> c->chrDstVSubSample) - 1) * dstStride[2];
197.     dst2[3] += ( c->dstH - 1) * dstStride[3];
198.
199.     reset_ptr(src2, c->srcFormat);
200.     reset_ptr((void*)dst2, c->dstFormat);
201.
202.     /* reset slice direction at end of frame */
203.     if (!srcSliceY)
204.         c->sliceDir = 0;
205.     //关键：调用
206.     ret = c->swscale(c, src2, srcStride2, c->srcH-srcSliceY-srcSliceH,
207.         srcSliceH, dst2, dstStride2);
208. }
209.
210.
211.     if (c->dstXYZ && !(c->srcXYZ && c->srcW==c->dstW && c->srcH==c->dstH)) {
212.         /* replace on the same data */
213.         rgb48Toxyz12(c, (uint16_t*)dst2[0], (const uint16_t*)dst2[0], dstStride[0]/2, ret);
214.     }
215.
216.     av_free(rgb0_tmp);
217.     return ret;
218. }

```

从sws_scale()的定义可以看出，它封装了SwsContext中的swscale()（注意这个函数中间没有“_”）。函数最重要的一句代码就是“c->swscale()”。除此之外，函数还做了一些增加“兼容性”的一些处理。函数的主要步骤如下所示。

1.检查输入的图像参数的合理性。

这一步骤首先检查输入输出的参数是否为空，然后通过调用check_image_pointers()检查输入输出图像的内存是否正确分配。check_image_pointers()的定义如下所示。

```

1. static int check_image_pointers(const uint8_t * const data[4], enum AVPixelFormat pix_fmt,
2.                                const int linesizes[4])
3. {
4.     const AVPixFmtDescriptor *desc = av_pix_fmt_desc_get(pix_fmt);
5.     int i;
6.
7.     for (i = 0; i < 4; i++) {
8.         int plane = desc->comp[i].plane;
9.         if (!data[plane] || !linesizes[plane])
10.            return 0;
11.     }
12.
13.     return 1;
14. }

```

从check_image_pointers()的定义可以看出，在特定像素格式前提下，如果该像素格式应该包含像素的分量为空，就返回0，否则返回1。

2.如果输入像素数据中使用了“调色板”（palette），则进行一些相应的处理。这一步通过函数usePal()来判定。usePal()的定义如下。

```

1. static av_always_inline int usePal(enum AVPixelFormat pix_fmt)
2. {
3.     const AVPixFmtDescriptor *desc = av_pix_fmt_desc_get(pix_fmt);
4.     av_assert0(desc);
5.     return (desc->flags & AV_PIX_FMT_FLAG_PAL) || (desc->flags & AV_PIX_FMT_FLAG_PSEUDOPAL);
6. }

```

从定义可以看出该函数通过判定AVPixFmtDescriptor中的flag是否包含AV_PIX_FMT_FLAG_PAL来断定像素格式是否使用了“调色板”。

3.其它一些特殊格式的处理，比如说Alpha, XYZ等的处理（这方面没有研究过）。

4.如果输入的图像的扫描方式是从底部到顶部的（一般情况下是从顶部到底部），则将图像进行反转。

5.调用SwsContext中的swscale()。

SwsContext中的swscale()

swscale这个变量的类型是SwsFunc，实际上就是一个函数指针。它是整个类库的 **核心**。当我们从外部调用swscale()函数的时候，实际上就是调用了SwsContext中的这个名称为swscale的变量（注意外部函数接口和这个内部函数指针的名字是一样的，但不是一回事）。

可以看一下SwsFunc这个类型的定义：

```
[cpp]
1. typedef int (*SwsFunc)(struct SwsContext *context, const uint8_t *src[],
2.                       int srcStride[], int srcSliceY, int srcSliceH,
3.                       uint8_t *dst[], int dstStride[]);
```

可以看出SwsFunc的定义的参数类型和libswscale类库外部接口函数swscale()的参数类型一模一样。

在libswscale中，该指针的指向可以分成2种情况：

- 1.图像没有伸缩的时候，指向专有的像素转换函数
- 2.图像有伸缩的时候，指向swscale()函数。

在调用sws_getContext()初始化SwsContext的时候，会在其子函数sws_init_context()中对swscale指针进行赋值。如果图像没有进行拉伸，则会调用ff_get_unscaled_swscale()对其进行赋值；如果图像进行了拉伸，则会调用ff_getSwsFunc()对其进行赋值。下面分别看一下这2种情况。

没有拉伸--专有的像素转换函数

如果图像没有进行拉伸，则会调用ff_get_unscaled_swscale()对SwsContext的swscale进行赋值。上篇文章中记录了这个函数，在这里回顾一下。

ff_get_unscaled_swscale()

ff_get_unscaled_swscale()的定义如下。

```
[cpp]
1. void ff_get_unscaled_swscale(SwsContext *c)
2. {
3.     const enum AVPixelFormat srcFormat = c->srcFormat;
4.     const enum AVPixelFormat dstFormat = c->dstFormat;
5.     const int flags = c->flags;
6.     const int dstH = c->dstH;
7.     int needsDither;
8.
9.     needsDither = isAnyRGB(dstFormat) &&
10.                  c->dstFormatBpp < 24 &&
11.                  (c->dstFormatBpp < c->srcFormatBpp || (!isAnyRGB(srcFormat)));
12.
13.     /* yv12_to_nv12 */
14.     if ((srcFormat == AV_PIX_FMT_YUV420P || srcFormat == AV_PIX_FMT_YUVA420P) &&
15.         (dstFormat == AV_PIX_FMT_NV12 || dstFormat == AV_PIX_FMT_NV21)) {
16.         c->swscale = planarToNv12Wrapper;
17.     }
18.     /* nv12_to_yv12 */
19.     if (dstFormat == AV_PIX_FMT_YUV420P &&
20.         (srcFormat == AV_PIX_FMT_NV12 || srcFormat == AV_PIX_FMT_NV21)) {
21.         c->swscale = nv12ToPlanarWrapper;
22.     }
23.     /* yuv2bgr */
24.     if ((srcFormat == AV_PIX_FMT_YUV420P || srcFormat == AV_PIX_FMT_YUVA422P ||
25.         srcFormat == AV_PIX_FMT_YUVA420P) && isAnyRGB(dstFormat) &&
26.         !(flags & SWS_ACCURATE_RND) && (c->dither == SWS_DITHER_BAYER || c->dither == SWS_DITHER_AUTO) && !(dstH & 1)) {
27.         c->swscale = ff_yuv2rgb_get_func_ptr(c);
28.     }
29.
30.     if (srcFormat == AV_PIX_FMT_YUV410P && !(dstH & 3) &&
31.         (dstFormat == AV_PIX_FMT_YUV420P || dstFormat == AV_PIX_FMT_YUVA420P) &&
32.         !(flags & SWS_BITEXACT)) {
33.         c->swscale = yvu9ToYv12Wrapper;
34.     }
35.
36.     /* bgr24toYV12 */
37.     if (srcFormat == AV_PIX_FMT_BGR24 &&
38.         (dstFormat == AV_PIX_FMT_YUV420P || dstFormat == AV_PIX_FMT_YUVA420P) &&
39.         !(flags & SWS_ACCURATE_RND))
40.         c->swscale = bgr24ToYv12Wrapper;
41.
42.     /* RGB/BGR -> RGB/BGR (no dither needed forms) */
43.     if (isAnyRGB(srcFormat) && isAnyRGB(dstFormat) && findRgbConvFn(c)
44.         && (!needsDither || (c->flags & (SWS_FAST_BILINEAR | SWS_POINT))))
45.         c->swscale = rgbToRgbWrapper;
46.
47.     if ((srcFormat == AV_PIX_FMT_GBRP && dstFormat == AV_PIX_FMT_GBRAP) ||
48.         (srcFormat == AV_PIX_FMT_GBRAP && dstFormat == AV_PIX_FMT_GBRP))
49.         c->swscale = planarRgbToplanarRgbWrapper;
50.
51.     #define isByteRGB(f) ( \
52.         f == AV_PIX_FMT_RGB32 || \
53.         f == AV_PIX_FMT_RGB32_1 || \
54.         f == AV_PIX_FMT_RGB24 || \
55.         f == AV_PIX_FMT_BGR32 || \
56.         f == AV_PIX_FMT_BGR32_1 || \
57.         f == AV_PIX_FMT_BGR24)
58.
```



```

59.     if (srcFormat == AV_PIX_FMT_GBRP && isPlanar(srcFormat) && isByteRGB(dstFormat))
60.         c->swscale = planarRgbToRgbWrapper;
61.
62.     if ((srcFormat == AV_PIX_FMT_RGB48LE || srcFormat == AV_PIX_FMT_RGB48BE ||
63.          srcFormat == AV_PIX_FMT_BGR48LE || srcFormat == AV_PIX_FMT_BGR48BE ||
64.          srcFormat == AV_PIX_FMT_RGBA64LE || srcFormat == AV_PIX_FMT_RGBA64BE ||
65.          srcFormat == AV_PIX_FMT_BGRA64LE || srcFormat == AV_PIX_FMT_BGRA64BE) &&
66.         (dstFormat == AV_PIX_FMT_GBRP9LE || dstFormat == AV_PIX_FMT_GBRP9BE ||
67.          dstFormat == AV_PIX_FMT_GBRP10LE || dstFormat == AV_PIX_FMT_GBRP10BE ||
68.          dstFormat == AV_PIX_FMT_GBRP12LE || dstFormat == AV_PIX_FMT_GBRP12BE ||
69.          dstFormat == AV_PIX_FMT_GBRP14LE || dstFormat == AV_PIX_FMT_GBRP14BE ||
70.          dstFormat == AV_PIX_FMT_GBRP16LE || dstFormat == AV_PIX_FMT_GBRP16BE ||
71.          dstFormat == AV_PIX_FMT_GBRAP16LE || dstFormat == AV_PIX_FMT_GBRAP16BE ))
72.         c->swscale = Rgb16ToPlanarRgb16Wrapper;
73.
74.     if ((srcFormat == AV_PIX_FMT_GBRP9LE || srcFormat == AV_PIX_FMT_GBRP9BE ||
75.          srcFormat == AV_PIX_FMT_GBRP16LE || srcFormat == AV_PIX_FMT_GBRP16BE ||
76.          srcFormat == AV_PIX_FMT_GBRP10LE || srcFormat == AV_PIX_FMT_GBRP10BE ||
77.          srcFormat == AV_PIX_FMT_GBRP12LE || srcFormat == AV_PIX_FMT_GBRP12BE ||
78.          srcFormat == AV_PIX_FMT_GBRP14LE || srcFormat == AV_PIX_FMT_GBRP14BE ||
79.          srcFormat == AV_PIX_FMT_GBRAP16LE || srcFormat == AV_PIX_FMT_GBRAP16BE) &&
80.         (dstFormat == AV_PIX_FMT_RGB48LE || dstFormat == AV_PIX_FMT_RGB48BE ||
81.          dstFormat == AV_PIX_FMT_BGR48LE || dstFormat == AV_PIX_FMT_BGR48BE ||
82.          dstFormat == AV_PIX_FMT_RGBA64LE || dstFormat == AV_PIX_FMT_RGBA64BE ||
83.          dstFormat == AV_PIX_FMT_BGRA64LE || dstFormat == AV_PIX_FMT_BGRA64BE))
84.         c->swscale = planarRgb16ToRgb16Wrapper;
85.
86.     if (av_pix_fmt_desc_get(srcFormat)->comp[0].depth_minus1 == 7 &&
87.         isPackedRGB(srcFormat) && dstFormat == AV_PIX_FMT_GBRP)
88.         c->swscale = rgbToPlanarRgbWrapper;
89.
90.     if (isBayer(srcFormat)) {
91.         if (dstFormat == AV_PIX_FMT_RGB24)
92.             c->swscale = bayer_to_rgb24_wrapper;
93.         else if (dstFormat == AV_PIX_FMT_YUV420P)
94.             c->swscale = bayer_to_yv12_wrapper;
95.         else if (!isBayer(dstFormat)) {
96.             av_log(c, AV_LOG_ERROR, "unsupported bayer conversion\n");
97.             av_assert0(0);
98.         }
99.     }
100.
101.     /* bswap 16 bits per pixel/component packed formats */
102.     if (IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_BAYER_BGGR16) ||
103.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_BAYER_RGGB16) ||
104.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_BAYER_GBRG16) ||
105.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_BAYER_GRBG16) ||
106.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_BGR444) ||
107.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_BGR48) ||
108.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_BGRA64) ||
109.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_BGR555) ||
110.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_BGR565) ||
111.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_BGRA64) ||
112.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_GRAY16) ||
113.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YA16) ||
114.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_GBRP9) ||
115.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_GBRP10) ||
116.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_GBRP12) ||
117.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_GBRP14) ||
118.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_GBRP16) ||
119.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_GBRAP16) ||
120.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_RGB444) ||
121.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_RGB48) ||
122.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_RGBA64) ||
123.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_RGB555) ||
124.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_RGB565) ||
125.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_RGBA64) ||
126.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_XYZ12) ||
127.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV420P9) ||
128.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV420P10) ||
129.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV420P12) ||
130.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV420P14) ||
131.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV420P16) ||
132.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV422P9) ||
133.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV422P10) ||
134.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV422P12) ||
135.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV422P14) ||
136.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV422P16) ||
137.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV444P9) ||
138.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV444P10) ||
139.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV444P12) ||
140.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV444P14) ||
141.         IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV444P16))
142.         c->swscale = packed_16bpc_bswap;
143.
144.     if (usePal(srcFormat) && isByteRGB(dstFormat))
145.         c->swscale = palToRgbWrapper;
146.
147.     if (srcFormat == AV_PIX_FMT_YUV422P) {
148.         if (dstFormat == AV_PIX_FMT_YUV422)
149.             c->swscale = yuv422nToYuv2Wrapper;

```

```

150.         else if (dstFormat == AV_PIX_FMT_UYVY422)
151.             c->swscale = yuv422pToUyvyWrapper;
152.     }
153.
154.     /* LQ converters if -sws 0 or -sws 4*/
155.     if (c->flags&(SWS_FAST_BILINEAR|SWS_POINT)) {
156.         /* yv12 to uyv2 */
157.         if (srcFormat == AV_PIX_FMT_YUV420P || srcFormat == AV_PIX_FMT_YUVA420P) {
158.             if (dstFormat == AV_PIX_FMT_UYVY422)
159.                 c->swscale = planarToUyvy2Wrapper;
160.             else if (dstFormat == AV_PIX_FMT_UYVY422)
161.                 c->swscale = planarToUyvyWrapper;
162.         }
163.     }
164.     if (srcFormat == AV_PIX_FMT_UYVY422 &&
165.         (dstFormat == AV_PIX_FMT_YUV420P || dstFormat == AV_PIX_FMT_YUVA420P))
166.         c->swscale = uyvyToYuv420Wrapper;
167.     if (srcFormat == AV_PIX_FMT_UYVY422 &&
168.         (dstFormat == AV_PIX_FMT_YUV420P || dstFormat == AV_PIX_FMT_YUVA420P))
169.         c->swscale = uyvyToYuv420Wrapper;
170.     if (srcFormat == AV_PIX_FMT_UYVY422 && dstFormat == AV_PIX_FMT_YUV422P)
171.         c->swscale = uyvyToYuv422Wrapper;
172.     if (srcFormat == AV_PIX_FMT_UYVY422 && dstFormat == AV_PIX_FMT_YUV422P)
173.         c->swscale = uyvyToYuv422Wrapper;
174.
175. #define isPlanarGray(x) (isGray(x) && (x) != AV_PIX_FMT_YA8 && (x) != AV_PIX_FMT_YA16LE && (x) != AV_PIX_FMT_YA16BE)
176.     /* simple copy */
177.     if ( srcFormat == dstFormat ||
178.         (srcFormat == AV_PIX_FMT_YUVA420P && dstFormat == AV_PIX_FMT_YUV420P) ||
179.         (srcFormat == AV_PIX_FMT_YUV420P && dstFormat == AV_PIX_FMT_YUVA420P) ||
180.         (isPlanarYUV(srcFormat) && isPlanarGray(dstFormat)) ||
181.         (isPlanarYUV(dstFormat) && isPlanarGray(srcFormat)) ||
182.         (isPlanarGray(dstFormat) && isPlanarGray(srcFormat)) ||
183.         (isPlanarYUV(srcFormat) && isPlanarYUV(dstFormat) &&
184.          c->chrDsthSubSample == c->chrSrcHSubSample &&
185.          c->chrDsthSubSample == c->chrSrcVSubSample &&
186.          dstFormat != AV_PIX_FMT_NV12 && dstFormat != AV_PIX_FMT_NV21 &&
187.          srcFormat != AV_PIX_FMT_NV12 && srcFormat != AV_PIX_FMT_NV21))
188.     {
189.         if (isPacked(c->srcFormat))
190.             c->swscale = packedCopyWrapper;
191.         else /* Planar YUV or gray */
192.             c->swscale = planarCopyWrapper;
193.     }
194.
195.     if (ARCH_PPC)
196.         ff_get_unscaled_swscale_ppc(c);
197.     // if (ARCH_ARM)
198.     //     ff_get_unscaled_swscale_arm(c);
199. }

```

从代码中可以看出，它根据输入输出像素格式的不同，选择了不同的转换函数。例如YUV420P转换NV12的时候，就会将planarToNv12Wrapper()赋值给SwsContext的swscale指针。

有拉伸--swscale()

如果图像进行了拉伸，则会调用ff_getSwsFunc()对SwsContext的swscale进行赋值。上篇文章中记录了这个函数，在这里回顾一下。

```

1. SwsFunc ff_getSwsFunc(SwsContext *c)
2. {
3.     sws_init_swscale(c);
4.
5.     if (ARCH_PPC)
6.         ff_sws_init_swscale_ppc(c);
7.     if (ARCH_X86)
8.         ff_sws_init_swscale_x86(c);
9.
10.    return swscale;
11. }

```

注意，sws_init_context()对SwsContext的swscale进行赋值的语句是：

```

1. c->swscale = ff_getSwsFunc(c);

```

即把ff_getSwsFunc()的返回值赋值给SwsContext的swscale指针；而ff_getSwsFunc()的返回值是一个静态函数，名称就叫做“swscale”。下面我们看一下这个swscale()静态函数的定义。

```

1. static int swscale(SwsContext *c, const uint8_t *src[],
2.                   int srcStride[], int srcSliceY,
3.                   int srcSliceH, uint8_t *dst[], int dstStride[])
4. {

```

```

4.  {
5.      /* load a few things into local vars to make the code more readable?
6.       * and faster */
7.      //注意一下这些参数
8.      //以亮度为准
9.      const int srcW          = c->srcW;
10.     const int dstW          = c->dstW;
11.     const int dstH          = c->dstH;
12.     //以色度为准
13.     const int chrDstW       = c->chrDstW;
14.     const int chrSrcW       = c->chrSrcW;
15.     const int lumXInc       = c->lumXInc;
16.     const int chrXInc       = c->chrXInc;
17.     const enum AVPixelFormat dstFormat = c->dstFormat;
18.     const int flags         = c->flags;
19.     int32_t *vLumFilterPos   = c->vLumFilterPos;
20.     int32_t *vChrFilterPos   = c->vChrFilterPos;
21.     int32_t *hLumFilterPos   = c->hLumFilterPos;
22.     int32_t *hChrFilterPos   = c->hChrFilterPos;
23.     int16_t *hLumFilter      = c->hLumFilter;
24.     int16_t *hChrFilter      = c->hChrFilter;
25.     int32_t *lumMmxFilter    = c->lumMmxFilter;
26.     int32_t *chrMmxFilter    = c->chrMmxFilter;
27.     const int vLumFilterSize = c->vLumFilterSize;
28.     const int vChrFilterSize = c->vChrFilterSize;
29.     const int hLumFilterSize = c->hLumFilterSize;
30.     const int hChrFilterSize = c->hChrFilterSize;
31.     int16_t **lumPixBuf      = c->lumPixBuf;
32.     int16_t **chrUPixBuf     = c->chrUPixBuf;
33.     int16_t **chrVPixBuf     = c->chrVPixBuf;
34.     int16_t **alpPixBuf      = c->alpPixBuf;
35.     const int vLumBufSize    = c->vLumBufSize;
36.     const int vChrBufSize    = c->vChrBufSize;
37.     uint8_t *formatConvBuffer = c->formatConvBuffer;
38.     uint32_t *pal            = c->pal_yuv;
39.     yuv2planar1_fn yuv2plane1 = c->yuv2plane1;
40.     yuv2planarX_fn yuv2planeX = c->yuv2planeX;
41.     yuv2interleavedX_fn yuv2nv12cX = c->yuv2nv12cX;
42.     yuv2packed1_fn yuv2packed1 = c->yuv2packed1;
43.     yuv2packed2_fn yuv2packed2 = c->yuv2packed2;
44.     yuv2packedX_fn yuv2packedX = c->yuv2packedX;
45.     yuv2anyX_fn yuv2anyX = c->yuv2anyX;
46.     const int chrSrcSliceY     = srcSliceY >> c->chrSrcVSubSample;
47.     const int chrSrcSliceH     = FF_CEIL_RSHIFT(srcSliceH, c->chrSrcVSubSample);
48.     int should_dither          = is9_OR_10BPS(c->srcFormat) ||
49.                                 is16BPS(c->srcFormat);
50.
51.     int lastDstY;
52.
53.     /* vars which will change and which we need to store back in the context */
54.     int dstY          = c->dstY;
55.     int lumBufIndex   = c->lumBufIndex;
56.     int chrBufIndex   = c->chrBufIndex;
57.     int lastInLumBuf  = c->lastInLumBuf;
58.     int lastInChrBuf  = c->lastInChrBuf;
59.
60.     if (!usePal(c->srcFormat)) {
61.         pal = c->input_rgb2yuv_table;
62.     }
63.
64.     if (isPacked(c->srcFormat)) {
65.         src[0] =
66.         src[1] =
67.         src[2] =
68.         src[3] = src[0];
69.         srcStride[0] =
70.         srcStride[1] =
71.         srcStride[2] =
72.         srcStride[3] = srcStride[0];
73.     }
74.     srcStride[1] <= c->vChrDrop;
75.     srcStride[2] <= c->vChrDrop;
76.
77.     DEBUG_BUFFERS("swscale() %p[%d] %p[%d] %p[%d] %p[%d] -> %p[%d] %p[%d] %p[%d] %p[%d]\n",
78.         src[0], srcStride[0], src[1], srcStride[1],
79.         src[2], srcStride[2], src[3], srcStride[3],
80.         dst[0], dstStride[0], dst[1], dstStride[1],
81.         dst[2], dstStride[2], dst[3], dstStride[3]);
82.     DEBUG_BUFFERS("srcSliceY: %d srcSliceH: %d dstY: %d dstH: %d\n",
83.         srcSliceY, srcSliceH, dstY, dstH);
84.     DEBUG_BUFFERS("vLumFilterSize: %d vLumBufSize: %d vChrFilterSize: %d vChrBufSize: %d\n",
85.         vLumFilterSize, vLumBufSize, vChrFilterSize, vChrBufSize);
86.
87.     if (dstStride[0]&15 || dstStride[1]&15 ||
88.         dstStride[2]&15 || dstStride[3]&15) {
89.         static int warnedAlready = 0; // FIXME maybe move this into the context
90.         if (flags & SWS_PRINT_INFO && !warnedAlready) {
91.             av_log(c, AV_LOG_WARNING,
92.                 "Warning: dstStride is not aligned!\n"
93.                 "                ->cannot do aligned memory accesses anymore\n");
94.             warnedAlready = 1;
95.         }
96.     }
97. }
98. }

```

```

96.     }
97.     if ( (uintptr_t)dst[0]&15 || (uintptr_t)dst[1]&15 || (uintptr_t)dst[2]&15
98.         || (uintptr_t)src[0]&15 || (uintptr_t)src[1]&15 || (uintptr_t)src[2]&15
99.         || dstStride[0]&15 || dstStride[1]&15 || dstStride[2]&15 || dstStride[3]&15
100.        || srcStride[0]&15 || srcStride[1]&15 || srcStride[2]&15 || srcStride[3]&15
101.    ) {
102.        static int warnedAlready=0;
103.        int cpu_flags = av_get_cpu_flags();
104.        if (HAVE_MMXEXT && (cpu_flags & AV_CPU_FLAG_SSE2) && !warnedAlready){
105.            av_log(c, AV_LOG_WARNING, "Warning: data is not aligned! This can lead to a speedloss\n");
106.            warnedAlready=1;
107.        }
108.    }
109.
110.    /* Note the user might start scaling the picture in the middle so this
111.     * will not get executed. This is not really intended but works
112.     * currently, so people might do it. */
113.    if (srcSliceY == 0) {
114.        lumBufIndex = -1;
115.        chrBufIndex = -1;
116.        dstY = 0;
117.        lastInLumBuf = -1;
118.        lastInChrBuf = -1;
119.    }
120.
121.    if (!should_dither) {
122.        c->chrDither8 = c->lumDither8 = sws_pb_64;
123.    }
124.    lastDstY = dstY;
125.    //逐行循环，一次循环代表处理一行
126.    //注意dstY和dstH两个变量
127.    for (; dstY < dstH; dstY++) {
128.        //色度的和亮度之间的关系
129.        const int chrDstY = dstY >> c->chrDstVSubSample;
130.        uint8_t *dest[4] = {
131.            dst[0] + dstStride[0] * dstY,
132.            dst[1] + dstStride[1] * chrDstY,
133.            dst[2] + dstStride[2] * chrDstY,
134.            (CONFIG_SWSCALE_ALPHA && alpPixBuf) ? dst[3] + dstStride[3] * dstY : NULL,
135.        };
136.        int use_mmx_vfilter= c->use_mmx_vfilter;
137.
138.        // First line needed as input
139.        const int firstLumSrcY = FFMX(1 - vLumFilterSize, vLumFilterPos[dstY]);
140.        const int firstLumSrcY2 = FFMX(1 - vLumFilterSize, vLumFilterPos[FFMIN(dstY | ((1 << c-
141.    >chrDstVSubSample) - 1), dstH - 1)]);
142.        // First line needed as input
143.        const int firstChrSrcY = FFMX(1 - vChrFilterSize, vChrFilterPos[chrDstY]);
144.
145.        // Last line needed as input
146.        int lastLumSrcY = FFMIN(c->srcH, firstLumSrcY + vLumFilterSize) - 1;
147.        int lastLumSrcY2 = FFMIN(c->srcH, firstLumSrcY2 + vLumFilterSize) - 1;
148.        int lastChrSrcY = FFMIN(c->chrSrcH, firstChrSrcY + vChrFilterSize) - 1;
149.        int enough_lines;
150.
151.        // handle holes (FAST_BILINEAR & weird filters)
152.        if (firstLumSrcY > lastInLumBuf)
153.            lastInLumBuf = firstLumSrcY - 1;
154.        if (firstChrSrcY > lastInChrBuf)
155.            lastInChrBuf = firstChrSrcY - 1;
156.        av_assert0(firstLumSrcY >= lastInLumBuf - vLumBufSize + 1);
157.        av_assert0(firstChrSrcY >= lastInChrBuf - vChrBufSize + 1);
158.
159.        DEBUG_BUFFERS("dstY: %d\n", dstY);
160.        DEBUG_BUFFERS("\tfirstLumSrcY: %d lastLumSrcY: %d lastInLumBuf: %d\n",
161.            firstLumSrcY, lastLumSrcY, lastInLumBuf);
162.        DEBUG_BUFFERS("\tfirstChrSrcY: %d lastChrSrcY: %d lastInChrBuf: %d\n",
163.            firstChrSrcY, lastChrSrcY, lastInChrBuf);
164.
165.        // Do we have enough lines in this slice to output the dstY line
166.        enough_lines = lastLumSrcY2 < srcSliceY + srcSliceH &&
167.            lastChrSrcY < FF_CEIL_RSHIFT(srcSliceY + srcSliceH, c->chrSrcVSubSample);
168.
169.        if (!enough_lines) {
170.            lastLumSrcY = srcSliceY + srcSliceH - 1;
171.            lastChrSrcY = chrSrcSliceY + chrSrcSliceH - 1;
172.            DEBUG_BUFFERS("buffering slice: lastLumSrcY %d lastChrSrcY %d\n",
173.                lastLumSrcY, lastChrSrcY);
174.        }
175.
176.        // Do horizontal scaling
177.        //水平拉伸
178.        //亮度
179.        while (lastInLumBuf < lastLumSrcY) {
180.            const uint8_t *src1[4] = {
181.                src[0] + (lastInLumBuf + 1 - srcSliceY) * srcStride[0],
182.                src[1] + (lastInLumBuf + 1 - srcSliceY) * srcStride[1],
183.                src[2] + (lastInLumBuf + 1 - srcSliceY) * srcStride[2],
184.                src[3] + (lastInLumBuf + 1 - srcSliceY) * srcStride[3],
185.            };
186.            lumBufIndex++;

```

```

186.     av_assert0(lumBufIndex < 2 * vLumBufSize);
187.     av_assert0(lastInLumBuf + 1 - srcSliceY < srcSliceH);
188.     av_assert0(lastInLumBuf + 1 - srcSliceY >= 0);
189.     //关键: 拉伸
190.     hyscale(c, lumPixBuf[lumBufIndex], dstW, src1, srcW, lumXInc,
191.             hLumFilter, hLumFilterPos, hLumFilterSize,
192.             formatConvBuffer, pal, 0);
193.     if (CONFIG_SWSCALE_ALPHA && alpPixBuf)
194.         hyscale(c, alpPixBuf[lumBufIndex], dstW, src1, srcW,
195.                 lumXInc, hLumFilter, hLumFilterPos, hLumFilterSize,
196.                 formatConvBuffer, pal, 1);
197.     lastInLumBuf++;
198.     DEBUG_BUFFERS("\t\tlumBufIndex %d: lastInLumBuf: %d\n",
199.                   lumBufIndex, lastInLumBuf);
200. }
201. //水平拉伸
202. //色度
203. while (lastInChrBuf < lastChrSrcY) {
204.     const uint8_t *src1[4] = {
205.         src[0] + (lastInChrBuf + 1 - chrSrcSliceY) * srcStride[0],
206.         src[1] + (lastInChrBuf + 1 - chrSrcSliceY) * srcStride[1],
207.         src[2] + (lastInChrBuf + 1 - chrSrcSliceY) * srcStride[2],
208.         src[3] + (lastInChrBuf + 1 - chrSrcSliceY) * srcStride[3],
209.     };
210.     chrBufIndex++;
211.     av_assert0(chrBufIndex < 2 * vChrBufSize);
212.     av_assert0(lastInChrBuf + 1 - chrSrcSliceY < (chrSrcSliceH));
213.     av_assert0(lastInChrBuf + 1 - chrSrcSliceY >= 0);
214.     // FIXME replace parameters through context struct (some at least)
215.     //关键: 拉伸
216.     if (c->needs_hcscale)
217.         hcscale(c, chrUPixBuf[chrBufIndex], chrVPixBuf[chrBufIndex],
218.                 chrDstW, src1, chrSrcW, chrXInc,
219.                 hChrFilter, hChrFilterPos, hChrFilterSize,
220.                 formatConvBuffer, pal);
221.     lastInChrBuf++;
222.     DEBUG_BUFFERS("\t\ttchrBufIndex %d: lastInChrBuf: %d\n",
223.                   chrBufIndex, lastInChrBuf);
224. }
225. // wrap buf index around to stay inside the ring buffer
226. if (lumBufIndex >= vLumBufSize)
227.     lumBufIndex -= vLumBufSize;
228. if (chrBufIndex >= vChrBufSize)
229.     chrBufIndex -= vChrBufSize;
230. if (!enough_lines)
231.     break; // we can't output a dstY line so let's try with the next slice
232.
233. #if HAVE_MMX_INLINE
234.     updateMMXDitherTables(c, dstY, lumBufIndex, chrBufIndex,
235.                           lastInLumBuf, lastInChrBuf);
236. #endif
237.     if (should_dither) {
238.         c->chrDither8 = ff_dither_8x8_128[chrDstY & 7];
239.         c->lumDither8 = ff_dither_8x8_128[dstY & 7];
240.     }
241.     if (dstY >= dstH - 2) {
242.         /* hmm looks like we can't use MMX here without overwriting
243.          * this array's tail */
244.         ff_sws_init_output_funcs(c, &yuv2plane1, &yuv2planeX, &yuv2nv12cX,
245.                                   &yuv2packed1, &yuv2packed2, &yuv2packedX, &yuv2anyX);
246.         use_mmx_vfilter= 0;
247.     }
248.
249.     {
250.         const int16_t **lumSrcPtr = (const int16_t **)
251. (void*) lumPixBuf + lumBufIndex + firstLumSrcY - lastInLumBuf + vLumBufSize;
252.         const int16_t **chrUSrcPtr = (const int16_t **)
253. (void*) chrUPixBuf + chrBufIndex + firstChrSrcY - lastInChrBuf + vChrBufSize;
254.         const int16_t **chrVSrcPtr = (const int16_t **)
255. (void*) chrVPixBuf + chrBufIndex + firstChrSrcY - lastInChrBuf + vChrBufSize;
256.         const int16_t **alpSrcPtr = (CONFIG_SWSCALE_ALPHA && alpPixBuf) ?
257. (const int16_t **)
258. (void*) alpPixBuf + lumBufIndex + firstLumSrcY - lastInLumBuf + vLumBufSize : NULL;
259.         int16_t *vLumFilter = c->vLumFilter;
260.         int16_t *vChrFilter = c->vChrFilter;
261.
262.         if (isPlanarYUV(dstFormat) ||
263.             (isGray(dstFormat) && !isALPHA(dstFormat))) { // YV12 like
264.             const int chrSkipMask = (1 << c->chrDstVSubSample) - 1;
265.
266.             vLumFilter += dstY * vLumFilterSize;
267.             vChrFilter += chrDstY * vChrFilterSize;
268.
269.             // av_assert0(use_mmx_vfilter != (
270.             //             yuv2planeX == yuv2planeX_10BE_c
271.             //             || yuv2planeX == yuv2planeX_10LE_c
272.             //             || yuv2planeX == yuv2planeX_9BE_c
273.             //             || yuv2planeX == yuv2planeX_9LE_c
274.             //             || yuv2planeX == yuv2planeX_16BE_c
275.             //             || yuv2planeX == yuv2planeX_16LE_c
276.             //             || yuv2planeX == yuv2planeX_8 c) || !ARCH_X86);

```

```

273.
274.     if(use_mmx_vfilter){
275.         vLumFilter= (int16_t *)c->lumMmxFilter;
276.         vChrFilter= (int16_t *)c->chrMmxFilter;
277.     }
278.     //输出一行水平拉伸过的像素
279.     //亮度
280.     //是否垂直拉伸?
281.     if (vLumFilterSize == 1) {
282.         //亮度-不垂直拉伸-分量模式 (planar) -输出一行水平拉伸的像素
283.         yuv2plane1(lumSrcPtr[0], dest[0], dstW, c->lumDither8, 0);
284.     } else {
285.         //亮度-垂直拉伸-分量模式 (planar) -输出一行水平拉伸的像素
286.         yuv2planeX(vLumFilter, vLumFilterSize,
287.                     lumSrcPtr, dest[0],
288.                     dstW, c->lumDither8, 0);
289.     }
290.     //色度
291.     //是否垂直拉伸?
292.     if (!(dstY & chrSkipMask) || isGray(dstFormat)) {
293.         if (yuv2nv12cX(c, vChrFilter,
294.                         vChrFilterSize, chrUSrcPtr, chrVSrcPtr,
295.                         dest[1], chrDstW);
296.         } else if (vChrFilterSize == 1) {
297.             //色度-不垂直拉伸-分量模式 (planar) -输出一行水平拉伸的像素
298.             //注意是2个分量
299.             yuv2plane1(chrUSrcPtr[0], dest[1], chrDstW, c->chrDither8, 0);
300.             yuv2plane1(chrVSrcPtr[0], dest[2], chrDstW, c->chrDither8, 3);
301.         } else {
302.             //色度-垂直拉伸-分量模式 (planar) -输出一行水平拉伸的像素
303.             //注意是2个分量
304.             yuv2planeX(vChrFilter,
305.                         vChrFilterSize, chrUSrcPtr, dest[1],
306.                         chrDstW, c->chrDither8, 0);
307.             yuv2planeX(vChrFilter,
308.                         vChrFilterSize, chrVSrcPtr, dest[2],
309.                         chrDstW, c->chrDither8, use_mmx_vfilter ? (c->uv_offx2 >> 1) : 3);
310.         }
311.     }
312. }
313.
314. if (CONFIG_SWSCALE_ALPHA && alpPixBuf) {
315.     if(use_mmx_vfilter){
316.         vLumFilter= (int16_t *)c->alpMmxFilter;
317.     }
318.     if (vLumFilterSize == 1) {
319.         yuv2plane1(alpSrcPtr[0], dest[3], dstW,
320.                     c->lumDither8, 0);
321.     } else {
322.         yuv2planeX(vLumFilter,
323.                     vLumFilterSize, alpSrcPtr, dest[3],
324.                     dstW, c->lumDither8, 0);
325.     }
326. }
327. } else if (yuv2packedX) {
328.     av_assert1(lumSrcPtr + vLumFilterSize - 1 < (const int16_t *)lumPixBuf + vLumBufSize * 2);
329.     av_assert1(chrUSrcPtr + vChrFilterSize - 1 < (const int16_t *)chrUPixBuf + vChrBufSize * 2);
330.     if (c->yuv2packed1 && vLumFilterSize == 1 &&
331.         vChrFilterSize <= 2) { // unscaled RGB
332.         int chrAlpha = vChrFilterSize == 1 ? 0 : vChrFilter[2 * dstY + 1];
333.         //不垂直拉伸-打包模式 (packed) -输出一行水平拉伸的像素
334.         yuv2packed1(c, *lumSrcPtr, chrUSrcPtr, chrVSrcPtr,
335.                     alpPixBuf ? *alpSrcPtr : NULL,
336.                     dest[0], dstW, chrAlpha, dstY);
337.     } else if (c->yuv2packed2 && vLumFilterSize == 2 &&
338.         vChrFilterSize == 2) { // bilinear upscale RGB
339.         int lumAlpha = vLumFilter[2 * dstY + 1];
340.         int chrAlpha = vChrFilter[2 * dstY + 1];
341.         lumMmxFilter[2] =
342.         lumMmxFilter[3] = vLumFilter[2 * dstY] * 0x10001;
343.         chrMmxFilter[2] =
344.         chrMmxFilter[3] = vChrFilter[2 * chrDstY] * 0x10001;
345.         yuv2packed2(c, lumSrcPtr, chrUSrcPtr, chrVSrcPtr,
346.                     alpPixBuf ? alpSrcPtr : NULL,
347.                     dest[0], dstW, lumAlpha, chrAlpha, dstY);
348.     } else { // general RGB
349.         //垂直拉伸-打包模式 (packed) -输出一行水平拉伸的像素
350.         yuv2packedX(c, vLumFilter + dstY * vLumFilterSize,
351.                     lumSrcPtr, vLumFilterSize,
352.                     vChrFilter + dstY * vChrFilterSize,
353.                     chrUSrcPtr, chrVSrcPtr, vChrFilterSize,
354.                     alpSrcPtr, dest[0], dstW, dstY);
355.     }
356. } else {
357.     av_assert1(!yuv2packed1 && !yuv2packed2);
358.     yuv2anyX(c, vLumFilter + dstY * vLumFilterSize,
359.              lumSrcPtr, vLumFilterSize,
360.              vChrFilter + dstY * vChrFilterSize,
361.              chrUSrcPtr, chrVSrcPtr, vChrFilterSize,
362.              alpSrcPtr, dest, dstW, dstY);
363. }

```

```

364.     }
365. }
366. if (isPlanar(dstFormat) && isALPHA(dstFormat) && !alpPixBuf) {
367.     int length = dstW;
368.     int height = dstY - lastDstY;
369.
370.     if (is16BPS(dstFormat) || isNBPS(dstFormat)) {
371.         const AVPixFmtDescriptor *desc = av_pix_fmt_desc_get(dstFormat);
372.         fillPlane16(dst[3], dstStride[3], length, height, lastDstY,
373.                     1, desc->comp[3].depth_minus1,
374.                     isBE(dstFormat));
375.     } else
376.         fillPlane(dst[3], dstStride[3], length, height, lastDstY, 255);
377. }
378.
379. #if HAVE_MMEXT_INLINE
380.     if (av_get_cpu_flags() & AV_CPU_FLAG_MMEXT)
381.         __asm__ volatile ("sfence" ::: "memory");
382. #endif
383.     emms_c();
384.
385.     /* store changed local vars back in the context */
386.     c->dstY = dstY;
387.     c->lumBufIndex = lumBufIndex;
388.     c->chrBufIndex = chrBufIndex;
389.     c->lastInLumBuf = lastInLumBuf;
390.     c->lastInChrBuf = lastInChrBuf;
391.
392.     return dstY - lastDstY;
393. }

```

可以看出swscale()是一行一行的进行图像缩放工作的。其中每行数据的处理按照“先水平拉伸，然后垂直拉伸”的方式进行处理。具体的实现函数如下所示：

1.
 - 水平拉伸
 - a)
 - 亮度水平拉伸：hyscale()
 - b)
 - 色度水平拉伸：hcscale()
2.
 - 垂直拉伸
 - a)
 - Planar
 - i.
 - 亮度垂直拉伸-不拉伸：yuv2plane1()
 - ii.
 - 亮度垂直拉伸-拉伸：yuv2planeX()
 - iii.
 - 色度垂直拉伸-不拉伸：yuv2plane1()
 - iv.
 - 色度垂直拉伸-拉伸：yuv2planeX()
 - b)
 - Packed
 - i.
 - 垂直拉伸-不拉伸：yuv2packed1()
 - ii.
 - 垂直拉伸-拉伸：yuv2packedX()

下面具体看看这几个函数的定义。

hyscale()

水平亮度拉伸函数hyscale()的定义位于libswscale\swscale.c，如下所示。

```

1. // *** horizontal scale Y line to temp buffer
2. static av_always_inline void hyscale(SwsContext *c, int16_t *dst, int dstWidth,
3.                                     const uint8_t *src_in[4],
4.                                     int srcW, int xInc,
5.                                     const int16_t *hLumFilter,
6.                                     const int32_t *hLumFilterPos,
7.                                     int hLumFilterSize,
8.                                     uint8_t *formatConvBuffer,
9.                                     uint32_t *pal, int isAlpha)
10. {
11.     void (*toYV12)(uint8_t *, const uint8_t *, const uint8_t *, const uint8_t *, int, uint32_t *) =
12.         isAlpha ? c->alpToYV12 : c->lumToYV12;
13.     void (*convertRange)(int16_t *, int) = isAlpha ? NULL : c->lumConvertRange;
14.     const uint8_t *src = src_in[isAlpha ? 3 : 0];
15.
16.     if (toYV12) {
17.         toYV12(formatConvBuffer, src, src_in[1], src_in[2], srcW, pal);
18.         src = formatConvBuffer;
19.     } else if (c->readLumPlanar && !isAlpha) {
20.         //读取
21.         c->readLumPlanar(formatConvBuffer, src_in, srcW, c->input_rgb2yuv_table);
22.         //赋值
23.         src = formatConvBuffer;
24.     } else if (c->readAlpPlanar && isAlpha) {
25.         c->readAlpPlanar(formatConvBuffer, src_in, srcW, NULL);
26.         src = formatConvBuffer;
27.     }
28.
29.     if (!c->hyscale_fast) {
30.         //亮度-水平拉伸
31.         c->hyScale(c, dst, dstWidth, src, hLumFilter,
32.                  hLumFilterPos, hLumFilterSize);
33.     } else { // fast bilinear upscale / crap downscale
34.         c->hyscale_fast(c, dst, dstWidth, src, srcW, xInc);
35.     }
36.     //如果需要取值范围的转换 (0-255和16-235之间)
37.     if (convertRange)
38.         convertRange(dst, dstWidth);
39. }

```

从hyscale()的源代码可以看出，它的流程如下所示。

1.转换成Y（亮度）

如果SwsContext的toYV12()函数存在，调用用该函数将数据转换为Y。如果该函数不存在，则调用SwsContext的readLumPlanar()读取Y。

2.拉伸

拉伸通过SwsContext的hyScale ()函数完成。如果存在hyscale_fast()方法的话，系统会优先调用hyscale_fast()。

3.转换范围（如果需要的话）

如果需要转换亮度的取值范围（例如需要进行16-235的MPEG标准与0-255的JPEG标准之间的转换），则会调用SwsContext的lumConvertRange ()函数。

上述几个步骤的涉及到的函数在上一篇文章中几经介绍过了，在这里重复一下。

toYV12() [SwsContext ->lumToYV12()]

toYV12()的实现函数是在ff_sws_init_input_funcs()中初始化的。在这里举几种具体的输入像素格式。

输入格式为YUYV422/ YVYU422

ff_sws_init_input_funcs()中，输入像素格式为YUYV422/ YVYU422的时候，toYV12()指向yuy2ToY_c()函数。源代码如下所示。

```

1. case AV_PIX_FMT_YUYV422:
2. case AV_PIX_FMT_YVYU422:
3. case AV_PIX_FMT_YA8:
4.     c->lumToYV12 = yuy2ToY_c;
5.     break;

```

yuy2ToY_c()的定义如下所示。

```

1. static void yuy2ToY_c(uint8_t *dst, const uint8_t *src, const uint8_t *unused1, const uint8_t *unused2, int width,
2.                      uint32_t *unused)
3. {
4.     int i;
5.     for (i = 0; i < width; i++)
6.         dst[i] = src[2 * i];
7. }

```

从yuy2ToY_c()的定义可以看出，该函数取出了所有的Y值（Y值在src[]数组中的下标为偶数）。

输入格式为RGB24

ff_sws_init_input_funcs()中, 输入像素格式为RGB24的时候, toYV12()指向yuy2ToY_c()函数。源代码如下所示。

```
[cpp]
1. case AV_PIX_FMT_RGB24:
2.     c->lumToYV12 = rgb24ToY_c;
3.     break;
```

rgb24ToY_c()的定义如下所示。

```
[cpp]
1. static void rgb24ToY_c(uint8_t *dst, const uint8_t *src, const uint8_t *unused1, const uint8_t *unused2, int width,
2.                      uint32_t *rgb2yuv)
3. {
4.     int16_t *dst = (int16_t *)dst;
5.     int32_t ry = rgb2yuv[RY_IDX], gy = rgb2yuv[G_Y_IDX], by = rgb2yuv[BY_IDX];
6.     int i;
7.     for (i = 0; i < width; i++) {
8.         int r = src[i * 3 + 0];
9.         int g = src[i * 3 + 1];
10.        int b = src[i * 3 + 2];
11.
12.        dst[i] = ((ry*r + gy*g + by*b + (32<<(RGB2YUV_SHIFT-1)) + (1<<(RGB2YUV_SHIFT-7)))>>(RGB2YUV_SHIFT-6));
13.    }
14. }
```

从rgb24ToY_c()的定义可以看出, 该函数通过R、G、B三个元素计算Y的值。其中R、G、B的系数取自于数组rgb2yuv[] (这个地方还没有研究) ; RGB2YUV_SHIFT似乎代表了转换后YUV的位数, 取值为15 (这个地方也还没有深入看)。

SwsContext -> hyScale ()

SwsContext -> hyScale ()的实现函数是在sws_init_swscale ()中初始化的。可以回顾一下sws_init_swscale ()的定义, 如下所示。

```
[cpp]
1. static av_cold void sws_init_swscale(SwsContext *c)
2. {
3.     enum AVPixelFormat srcFormat = c->srcFormat;
4.
5.     ff_sws_init_output_funcs(c, &c->yuv2plane1, &c->yuv2planeX,
6.                             &c->yuv2nv12cX, &c->yuv2packed1,
7.                             &c->yuv2packed2, &c->yuv2packedX, &c->yuv2anyX);
8.
9.     ff_sws_init_input_funcs(c);
10.
11.
12.     if (c->srcBpc == 8) {
13.         if (c->dstBpc <= 14) {
14.             c->hyScale = c->hcScale = hScale8To15_c;
15.             if (c->flags & SWS_FAST_BILINEAR) {
16.                 c->hyscale_fast = ff_hyscale_fast_c;
17.                 c->hcscale_fast = ff_hcscale_fast_c;
18.             }
19.         } else {
20.             c->hyScale = c->hcScale = hScale8To19_c;
21.         }
22.     } else {
23.         c->hyScale = c->hcScale = c->dstBpc > 14 ? hScale16To19_c
24.                                                  : hScale16To15_c;
25.     }
26.
27.     ff_sws_init_range_convert(c);
28.
29.     if (!(isGray(srcFormat) || isGray(c->dstFormat) ||
30.          srcFormat == AV_PIX_FMT_MONOBLACK || srcFormat == AV_PIX_FMT_MONOWHITE))
31.         c->needs_hcscale = 1;
32. }
```

从sws_init_swscale ()的定义可以看出, ff_sws_init_input_funcs()和ff_sws_init_range_convert()之间的代码完成了hyScale()的初始化。根据srcBpc和dstBpc取值的不同, 有几种不同的拉伸函数。根据我的理解, srcBpc代表了输入的第个像素单个分量的位数, dstBpc代表了输出的第个像素单个分量的位数。最常见的像素单个分量的位数是8位。从代码中可以看出, 在输入像素单个分量的位数为8位, 而且输出像素单个分量的位数也为8位的时候, SwsContext 的 hyScale ()会指向hScale8To15_c()函数。

hScale8To15_c()

hScale8To15_c()的定义如下所示。有关这个方面的代码还没有详细研究, 日后再作补充。

```
[cpp]
1. // bilinear / bicubic scaling
2. static void hScale8To15_c(SwsContext *c, int16_t *dst, int dstW,
3.                          const uint8_t *src, const int16_t *filter,
4.                          const int32_t *filterPos, int filterSize)
5. {
6.     int i;
7.     for (i = 0; i < dstW; i++) {
8.         int j;
9.         int srcPos = filterPos[i];
10.        int val = 0;
11.        for (j = 0; j < filterSize; j++) {
12.            val += ((int)src[srcPos + j]) * filter[filterSize * i + j];
13.        }
14.        dst[i] = FFMIN(val >> 7, (1 << 15) - 1); // the cubic equation does overflow ...
15.    }
16. }
```

lumConvertRange () [SwsContext -> lumConvertRange()]

SwsContext -> hyScale ()的实现函数是在ff_sws_init_range_convert()中初始化的。可以回顾一下ff_sws_init_range_convert ()的定义,如下所示。

```
[cpp]
1. av_cold void ff_sws_init_range_convert(SwsContext *c)
2. {
3.     c->lumConvertRange = NULL;
4.     c->chrConvertRange = NULL;
5.     if (c->srcRange != c->dstRange && !isAnyRGB(c->dstFormat)) {
6.         if (c->dstBpc <= 14) {
7.             if (c->srcRange) {
8.                 c->lumConvertRange = lumRangeFromJpeg_c;
9.                 c->chrConvertRange = chrRangeFromJpeg_c;
10.            } else {
11.                c->lumConvertRange = lumRangeToJpeg_c;
12.                c->chrConvertRange = chrRangeToJpeg_c;
13.            }
14.        } else {
15.            if (c->srcRange) {
16.                c->lumConvertRange = lumRangeFromJpeg16_c;
17.                c->chrConvertRange = chrRangeFromJpeg16_c;
18.            } else {
19.                c->lumConvertRange = lumRangeToJpeg16_c;
20.                c->chrConvertRange = chrRangeToJpeg16_c;
21.            }
22.        }
23.    }
24. }
```

SwsContext 的lumConvertRange()函数主要用于JPEG标准像素取值范围（0-255）和MPEG标准像素取值范围（16-235）之间的转换。有关这方面的分析在上一篇文章中一详细叙述过，在这里不再重复。简单看一下其中的一个函数。

lumRangeFromJpeg_c()

把亮度从JPEG标准转换为MPEG标准（0-255转换为16-235）的函数lumRangeFromJpeg_c()的定义如下所示。

```
[cpp]
1. static void lumRangeFromJpeg_c(int16_t *dst, int width)
2. {
3.     int i;
4.     for (i = 0; i < width; i++)
5.         dst[i] = (dst[i] * 14071 + 33561947) >> 14;
6. }
```

其实这个函数就是做了一个（0-255）到（16-235）的映射。它将亮度值“0”映射成“16”，“255”映射成“235”，因此我们可以代入一个“255”看看转换后的数值是否为“235”。在这里需要注意，dst中存储的像素数值是15bit的亮度值。因此我们需要将8bit的数值“255”左移7位后带入。经过计算，255左移7位后取值为32640，计算后得到的数值为30080，右移7位后得到的8bit亮度值即为235。

hcscale()

水平色度拉伸函数hcscale()的定义位于libswscale\swscale.c，如下所示。

```

1. static av_always_inline void hcscale(SwsContext *c, int16_t *dst1,
2.                                     int16_t *dst2, int dstWidth,
3.                                     const uint8_t *src_in[4],
4.                                     int srcW, int xInc,
5.                                     const int16_t *hChrFilter,
6.                                     const int32_t *hChrFilterPos,
7.                                     int hChrFilterSize,
8.                                     uint8_t *formatConvBuffer, uint32_t *pal)
9. {
10.     const uint8_t *src1 = src_in[1], *src2 = src_in[2];
11.     if (c->chrToYV12) {
12.         uint8_t *buf2 = formatConvBuffer +
13.             FFALIGN(srcW*2+78, 16);
14.         //转换
15.         c->chrToYV12(formatConvBuffer, buf2, src_in[0], src1, src2, srcW, pal);
16.         src1= formatConvBuffer;
17.         src2= buf2;
18.     } else if (c->readChrPlanar) {
19.         uint8_t *buf2 = formatConvBuffer +
20.             FFALIGN(srcW*2+78, 16);
21.         //读取
22.         c->readChrPlanar(formatConvBuffer, buf2, src_in, srcW, c->input_rgb2yuv_table);
23.         //赋值
24.         src1 = formatConvBuffer;
25.         src2 = buf2;
26.     }
27.
28.     if (!c->hcscale_fast) {
29.         //色度-水平拉伸
30.         c->hcScale(c, dst1, dstWidth, src1, hChrFilter, hChrFilterPos, hChrFilterSize);
31.         c->hcScale(c, dst2, dstWidth, src2, hChrFilter, hChrFilterPos, hChrFilterSize);
32.     } else { // fast bilinear upscale / crap downscale
33.         c->hcscale_fast(c, dst1, dst2, dstWidth, src1, src2, srcW, xInc);
34.     }
35.     //如果需要取值范围的转换 (0-255和16-235之间)
36.     if (c->chrConvertRange)
37.         c->chrConvertRange(dst1, dst2, dstWidth);
38. }

```

从hcscale()的源代码可以看出，它的流程如下所示。

1.转换成UV

该功能通过SwsContext的chrToYV12 ()函数完成。如果该函数不存在，则调用SwsContext的readChrPlanar ()读取UV。

2.拉伸

拉伸通过SwsContext的hcScale ()函数完成。如果存在hcscale_fast()方法的话，系统会优先调用hcscale_fast ()。

3.转换范围（如果需要的话）

如果需要转换色度的取值范围（例如色度取值范围从0-255转换为16-240），则会调用SwsContext的chrConvertRange ()函数。

hcscale()的原理和hyScale ()的原理基本上是一样的，在这里既不再详细研究了。

还有几个函数没有分析，但是时间有限，以后有机会再进行补充。

雷霄骅

leixiaohua1020@126.com

<http://blog.csdn.net/leixiaohua1020>

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/44346687>

文章标签： FFmpeg swscale YUV RGB 源代码

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