

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

2015年03月17日 12:16:43 阅读数：23209

=====

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

=====

打算写两篇文章记录FFmpeg中的图像处理（缩放，YUV/RGB格式转换）类库libswscale的源代码。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） 》

libswscale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。

其中sws_getContext()也可以用sws_getCachedContext()取代。

尽管libswscale从表面上看常用函数的个数不多，它的内部却有一个大大的“世界”。做为一个几乎“万能”的图片像素数据处理类库，它的内部包含了大量的代码。因此计划写两篇文章分析它的源代码。本文首先分析它的初始化函数sws_getContext()，而下一篇文章则分析它的数据处理函数sws_scale()。

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。

□

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。

SwsContext

SwsContext是使用libswscale时候一个贯穿始终的结构体。但是我们在使用FFmpeg的类库进行开发的时候，是无法看到它的内部结构的。在libswscale\swscale.h中只能看到一行定义：

[cpp] [icon] [icon]

```
1. struct SwsContext;
```

一般人看到这个只有一行定义的结构体，会猜测它的内部一定十分简单。但是假使我们看一下FFmpeg的源代码，会发现这个猜测是完全错误的——SwsContext的定义是十分复杂的。它的定义位于libswscale\swscale_internal.h中，如下所示。

[cpp] [icon] [icon]

```
1. /* This struct should be aligned on at least a 32-byte boundary. */
2. typedef struct SwsContext {
3.     /**
4.      * info on struct for av_log
5.      */
6.     const AVClass *av_class;
```

```

8.  /**
9.  * Note that src, dst, srcStride, dstStride will be copied in the
10. * sws_scale() wrapper so they can be freely modified here.
11. */
12. SwsFunc swscale;
13. int srcW;          ///< Width of source luma/alpha planes.
14. int srcH;          ///< Height of source luma/alpha planes.
15. int dstH;          ///< Height of destination luma/alpha planes.
16. int chrSrcW;       ///< Width of source chroma planes.
17. int chrSrcH;       ///< Height of source chroma planes.
18. int chrDstW;       ///< Width of destination chroma planes.
19. int chrDstH;       ///< Height of destination chroma planes.
20. int lumXInc, chrXInc;
21. int lumYInc, chrYInc;
22. enum AVPixelFormat dstFormat; ///< Destination pixel format.
23. enum AVPixelFormat srcFormat; ///< Source pixel format.
24. int dstFormatBpp;   ///< Number of bits per pixel of the destination pixel format.
25. int srcFormatBpp;   ///< Number of bits per pixel of the source pixel format.
26. int dstBpc, srcBpc;
27. int chrSrcHSubSample; ///< Binary logarithm of horizontal subsampling factor between luma/alpha and chroma planes in source
    image.
28. int chrSrcVSubSample; ///< Binary logarithm of vertical subsampling factor between luma/alpha and chroma planes in source
    image.
29. int chrDstHSubSample; ///< Binary logarithm of horizontal subsampling factor between luma/alpha and chroma planes in destination
    image.
30. int chrDstVSubSample; ///< Binary logarithm of vertical subsampling factor between luma/alpha and chroma planes in destination
    image.
31. int vChrDrop;        ///< Binary logarithm of extra vertical subsampling factor in source image chroma planes specified
    user.
32. int sliceDir;        ///< Direction that slices are fed to the scaler (1 = top-to-bottom, -1 = bottom-to-top).
33. double param[2];     ///< Input parameters for scaling algorithms that need them.
34.
35. /* The cascaded_* fields allow splitting a scaler task into multiple
36. * sequential steps, this is for example used to limit the maximum
37. * downscaling factor that needs to be supported in one scaler.
38. */
39. struct SwsContext *cascaded_context[2];
40. int cascaded_tmpStride[4];
41. uint8_t *cascaded_tmp[4];
42.
43. uint32_t pal_yuv[256];
44. uint32_t pal_rgb[256];
45.
46. /**
47. * @name Scaled horizontal lines ring buffer.
48. * The horizontal scaler keeps just enough scaled lines in a ring buffer
49. * so they may be passed to the vertical scaler. The pointers to the
50. * allocated buffers for each line are duplicated in sequence in the ring
51. * buffer to simplify indexing and avoid wrapping around between lines
52. * inside the vertical scaler code. The wrapping is done before the
53. * vertical scaler is called.
54. */
55. /**@{
56. int16_t **lumPixBuf;    ///< Ring buffer for scaled horizontal luma plane lines to be fed to the vertical scaler.
57. int16_t **chrUPixBuf;   ///< Ring buffer for scaled horizontal chroma plane lines to be fed to the vertical scaler.
58. int16_t **chrVPixBuf;   ///< Ring buffer for scaled horizontal chroma plane lines to be fed to the vertical scaler.
59. int16_t **alpPixBuf;    ///< Ring buffer for scaled horizontal alpha plane lines to be fed to the vertical scaler.
60. int vLumBufSize;        ///< Number of vertical luma/alpha lines allocated in the ring buffer.
61. int vChrBufSize;        ///< Number of vertical chroma lines allocated in the ring buffer.
62. int lastInLumBuf;       ///< Last scaled horizontal luma/alpha line from source in the ring buffer.
63. int lastInChrBuf;       ///< Last scaled horizontal chroma line from source in the ring buffer.
64. int lumBufIndex;        ///< Index in ring buffer of the last scaled horizontal luma/alpha line from source.
65. int chrBufIndex;        ///< Index in ring buffer of the last scaled horizontal chroma line from source.
66. /**@}
67.
68. uint8_t *formatConvBuffer;
69.
70. /**
71. * @name Horizontal and vertical filters.
72. * To better understand the following fields, here is a pseudo-code of
73. * their usage in filtering a horizontal line:
74. * @code
75. * for (i = 0; i < width; i++) {
76. *     dst[i] = 0;
77. *     for (j = 0; j < filterSize; j++)
78. *         dst[i] += src[ filterPos[i] + j ] * filter[ filterSize * i + j ];
79. *     dst[i] >>= FRAC_BITS; // The actual implementation is fixed-point.
80. * }
81. * @endcode
82. */
83. /**@{
84. int16_t *hLumFilter;    ///< Array of horizontal filter coefficients for luma/alpha planes.
85. int16_t *hChrFilter;    ///< Array of horizontal filter coefficients for chroma planes.
86. int16_t *vLumFilter;    ///< Array of vertical filter coefficients for luma/alpha planes.
87. int16_t *vChrFilter;    ///< Array of vertical filter coefficients for chroma planes.
88. int32_t *hLumFilterPos; ///< Array of horizontal filter starting positions for each dst[i] for luma/alpha planes.
89. int32_t *hChrFilterPos; ///< Array of horizontal filter starting positions for each dst[i] for chroma planes.
90. int32_t *vLumFilterPos; ///< Array of vertical filter starting positions for each dst[i] for luma/alpha planes.
91. int32_t *vChrFilterPos; ///< Array of vertical filter starting positions for each dst[i] for chroma planes.
92. int hLumFilterSize;     ///< Horizontal filter size for luma/alpha pixels.
93. int hChrFilterSize;     ///< Horizontal filter size for chroma pixels.

```

```

94.     int vLumFilterSize;          ///< Vertical filter size for luma/alpha pixels.
95.     int vChrFilterSize;          ///< Vertical filter size for chroma pixels.
96. }
97.
98.     int lumMmxextFilterCodeSize; ///< Runtime-generated MMXEXT horizontal fast bilinear scaler code size for luma/alpha planes.
99.     int chrMmxextFilterCodeSize; ///< Runtime-generated MMXEXT horizontal fast bilinear scaler code size for chroma planes.
100.     uint8_t *lumMmxextFilterCode; ///< Runtime-generated MMXEXT horizontal fast bilinear scaler code for luma/alpha planes.
101.     uint8_t *chrMmxextFilterCode; ///< Runtime-generated MMXEXT horizontal fast bilinear scaler code for chroma planes.
102.
103.     int canMMXEXTBeUsed;
104.
105.     int dstY;                    ///< Last destination vertical line output from last slice.
106.     int flags;                   ///< Flags passed by the user to select scaler algorithm, optimizations, subsampling, etc...
107.     void *yuvTable;              // pointer to the yuv->rgb table start so it can be freed()
108.     // alignment ensures the offset can be added in a single
109.     // instruction on e.g. ARM
110.     DECLARE_ALIGNED(16, int, table_gV)[256 + 2*YUVRGB_TABLE_HEADROOM];
111.     uint8_t *table_rV[256 + 2*YUVRGB_TABLE_HEADROOM];
112.     uint8_t *table_gU[256 + 2*YUVRGB_TABLE_HEADROOM];
113.     uint8_t *table_bU[256 + 2*YUVRGB_TABLE_HEADROOM];
114.     DECLARE_ALIGNED(16, int32_t, input_rgb2yuv_table)
[16*40*4]; // This table can contain both C and SIMD formatted values, the C vales are always at the XY_IDX points
115. #define RY_IDX 0
116. #define GY_IDX 1
117. #define BY_IDX 2
118. #define RU_IDX 3
119. #define GU_IDX 4
120. #define BU_IDX 5
121. #define RV_IDX 6
122. #define GV_IDX 7
123. #define BV_IDX 8
124. #define RGB2YUV_SHIFT 15
125.
126.     int *dither_error[4];
127.
128.     //Colorspace stuff
129.     int contrast, brightness, saturation; // for sws_getColorspaceDetails
130.     int srcColorspaceTable[4];
131.     int dstColorspaceTable[4];
132.     int srcRange;                ///< 0 = MPG YUV range, 1 = JPG YUV range (source image).
133.     int dstRange;                ///< 0 = MPG YUV range, 1 = JPG YUV range (destination image).
134.     int src0Alpha;
135.     int dst0Alpha;
136.     int srcXYZ;
137.     int dstXYZ;
138.     int src_h_chr_pos;
139.     int dst_h_chr_pos;
140.     int src_v_chr_pos;
141.     int dst_v_chr_pos;
142.     int yuv2rgb_y_offset;
143.     int yuv2rgb_y_coeff;
144.     int yuv2rgb_v2r_coeff;
145.     int yuv2rgb_v2g_coeff;
146.     int yuv2rgb_u2g_coeff;
147.     int yuv2rgb_u2b_coeff;
148.
149. #define RED_DITHER "0*8"
150. #define GREEN_DITHER "1*8"
151. #define BLUE_DITHER "2*8"
152. #define Y_COEFF "3*8"
153. #define VR_COEFF "4*8"
154. #define UB_COEFF "5*8"
155. #define VG_COEFF "6*8"
156. #define UG_COEFF "7*8"
157. #define Y_OFFSET "8*8"
158. #define U_OFFSET "9*8"
159. #define V_OFFSET "10*8"
160. #define LUM_MMX_FILTER_OFFSET "11*8"
161. #define CHR_MMX_FILTER_OFFSET "11*8+4*4*AV_STRINGIFY(MAX_FILTER_SIZE)"
162. #define DSTW_OFFSET "11*8+4*4*AV_STRINGIFY(MAX_FILTER_SIZE)*2"
163. #define ESP_OFFSET "11*8+4*4*AV_STRINGIFY(MAX_FILTER_SIZE)*2+8"
164. #define VROUNDER_OFFSET "11*8+4*4*AV_STRINGIFY(MAX_FILTER_SIZE)*2+16"
165. #define U_TEMP "11*8+4*4*AV_STRINGIFY(MAX_FILTER_SIZE)*2+24"
166. #define V_TEMP "11*8+4*4*AV_STRINGIFY(MAX_FILTER_SIZE)*2+32"
167. #define Y_TEMP "11*8+4*4*AV_STRINGIFY(MAX_FILTER_SIZE)*2+40"
168. #define ALP_MMX_FILTER_OFFSET "11*8+4*4*AV_STRINGIFY(MAX_FILTER_SIZE)*2+48"
169. #define UV_OFF_PX "11*8+4*4*AV_STRINGIFY(MAX_FILTER_SIZE)*3+48"
170. #define UV_OFF_BYTE "11*8+4*4*AV_STRINGIFY(MAX_FILTER_SIZE)*3+56"
171. #define DITHER16 "11*8+4*4*AV_STRINGIFY(MAX_FILTER_SIZE)*3+64"
172. #define DITHER32 "11*8+4*4*AV_STRINGIFY(MAX_FILTER_SIZE)*3+80"
173. #define DITHER32_INT (11*8+4*4*MAX_FILTER_SIZE*3+80) // value equal to above, used for checking that the struct hasn't been
nged by mistake
174.
175.     DECLARE_ALIGNED(8, uint64_t, redDither);
176.     DECLARE_ALIGNED(8, uint64_t, greenDither);
177.     DECLARE_ALIGNED(8, uint64_t, blueDither);
178.
179.     DECLARE_ALIGNED(8, uint64_t, yCoeff);
180.     DECLARE_ALIGNED(8, uint64_t, vrCoeff);
181.     DECLARE_ALIGNED(8, uint64_t, ubCoeff);
182.     DECLARE_ALIGNED(8, uint64_t, vgCoeff);

```

```

183. DECLARE_ALIGNED(8, uint64_t, ugCoeff);
184. DECLARE_ALIGNED(8, uint64_t, yOffset);
185. DECLARE_ALIGNED(8, uint64_t, uOffset);
186. DECLARE_ALIGNED(8, uint64_t, vOffset);
187. int32_t lumMmxFilter[4 * MAX_FILTER_SIZE];
188. int32_t chrMmxFilter[4 * MAX_FILTER_SIZE];
189. int dstW; //< Width of destination luma/alpha planes.
190. DECLARE_ALIGNED(8, uint64_t, esp);
191. DECLARE_ALIGNED(8, uint64_t, vRounder);
192. DECLARE_ALIGNED(8, uint64_t, u_temp);
193. DECLARE_ALIGNED(8, uint64_t, v_temp);
194. DECLARE_ALIGNED(8, uint64_t, y_temp);
195. int32_t alpMmxFilter[4 * MAX_FILTER_SIZE];
196. // alignment of these values is not necessary, but merely here
197. // to maintain the same offset across x8632 and x86-64. Once we
198. // use proper offset macros in the asm, they can be removed.
199. DECLARE_ALIGNED(8, ptrdiff_t, uv_off); //< offset (in pixels) between u and v planes
200. DECLARE_ALIGNED(8, ptrdiff_t, uv_offx2); //< offset (in bytes) between u and v planes
201. DECLARE_ALIGNED(8, uint16_t, dither16)[8];
202. DECLARE_ALIGNED(8, uint32_t, dither32)[8];
203.
204. const uint8_t *chrDither8, *lumDither8;
205.
206. #if HAVE_ALTIVEC
207. vector signed short CY;
208. vector signed short CRV;
209. vector signed short CBU;
210. vector signed short CGU;
211. vector signed short CGV;
212. vector signed short OY;
213. vector unsigned short CSHIFT;
214. vector signed short *vCCoeffsBank, *vCCoeffsBank;
215. #endif
216.
217. int use_mmx_vfilter;
218.
219. /* pre defined color-spaces gamma */
220. #define XYZ_GAMMA (2.6f)
221. #define RGB_GAMMA (2.2f)
222. int16_t *xyzgamma;
223. int16_t *rbggamma;
224. int16_t *xyzgammainv;
225. int16_t *rbggammainv;
226. int16_t xyz2rgb_matrix[3][4];
227. int16_t rgb2xyz_matrix[3][4];
228.
229. /* function pointers for swscale() */
230. yuv2planar1_fn yuv2plane1;
231. yuv2planarX_fn yuv2planeX;
232. yuv2interleavedX_fn yuv2nv12cX;
233. yuv2packed1_fn yuv2packed1;
234. yuv2packed2_fn yuv2packed2;
235. yuv2packedX_fn yuv2packedX;
236. yuv2anyX_fn yuv2anyX;
237.
238. /// Unscaled conversion of luma plane to YV12 for horizontal scaler.
239. void (*lumToYV12)(uint8_t *dst, const uint8_t *src, const uint8_t *src2, const uint8_t *src3,
240. int width, uint32_t *pal);
241. /// Unscaled conversion of alpha plane to YV12 for horizontal scaler.
242. void (*alpToYV12)(uint8_t *dst, const uint8_t *src, const uint8_t *src2, const uint8_t *src3,
243. int width, uint32_t *pal);
244. /// Unscaled conversion of chroma planes to YV12 for horizontal scaler.
245. void (*chrToYV12)(uint8_t *dstU, uint8_t *dstV,
246. const uint8_t *src1, const uint8_t *src2, const uint8_t *src3,
247. int width, uint32_t *pal);
248.
249. /**
250. * Functions to read planar input, such as planar RGB, and convert
251. * internally to Y/UV/A.
252. */
253. /** @{ */
254. void (*readLumPlanar)(uint8_t *dst, const uint8_t *src[4], int width, int32_t *rgb2yuv);
255. void (*readChrPlanar)(uint8_t *dstU, uint8_t *dstV, const uint8_t *src[4],
256. int width, int32_t *rgb2yuv);
257. void (*readAlpPlanar)(uint8_t *dst, const uint8_t *src[4], int width, int32_t *rgb2yuv);
258. /** @} */
259.
260. /**
261. * Scale one horizontal line of input data using a bilinear filter
262. * to produce one line of output data. Compared to SwsContext->hScale(),
263. * please take note of the following caveats when using these:
264. * - Scaling is done using only 7bit instead of 14bit coefficients.
265. * - You can use no more than 5 input pixels to produce 4 output
266. * pixels. Therefore, this filter should not be used for downscaling
267. * by more than ~20% in width (because that equals more than 5/4th
268. * downscaling and thus more than 5 pixels input per 4 pixels output).
269. * - In general, bilinear filters create artifacts during downscaling
270. * (even when <20%), because one output pixel will span more than one
271. * input pixel, and thus some pixels will need edges of both neighbor
272. * pixels to interpolate the output pixel. Since you can use at most
273. * two input pixels per output pixel in bilinear scaling, this is

```

```

274.     * impossible and thus downscaling by any size will create artifacts.
275.     * To enable this type of scaling, set SWS_FLAG_FAST_BILINEAR
276.     * in SwsContext->flags.
277.     */
278. /** @{ */
279. void (*hyscale_fast)(struct SwsContext *c,
280.                     int16_t *dst, int dstWidth,
281.                     const uint8_t *src, int srcW, int xInc);
282. void (*hcscale_fast)(struct SwsContext *c,
283.                     int16_t *dst1, int16_t *dst2, int dstWidth,
284.                     const uint8_t *src1, const uint8_t *src2,
285.                     int srcW, int xInc);
286. /** @} */
287.
288. /**
289.  * Scale one horizontal line of input data using a filter over the input
290.  * lines, to produce one (differently sized) line of output data.
291.  *
292.  * @param dst      pointer to destination buffer for horizontally scaled
293.  *                  data. If the number of bits per component of one
294.  *                  destination pixel (SwsContext->dstBpc) is <= 10, data
295.  *                  will be 15bpc in 16bits (int16_t) width. Else (i.e.
296.  *                  SwsContext->dstBpc == 16), data will be 19bpc in
297.  *                  32bits (int32_t) width.
298.  * @param dstW     width of destination image
299.  * @param src      pointer to source data to be scaled. If the number of
300.  *                  bits per component of a source pixel (SwsContext->srcBpc)
301.  *                  is 8, this is 8bpc in 8bits (uint8_t) width. Else
302.  *                  (i.e. SwsContext->dstBpc > 8), this is native depth
303.  *                  in 16bits (uint16_t) width. In other words, for 9-bit
304.  *                  YUV input, this is 9bpc, for 10-bit YUV input, this is
305.  *                  10bpc, and for 16-bit RGB or YUV, this is 16bpc.
306.  * @param filter    filter coefficients to be used per output pixel for
307.  *                  scaling. This contains 14bpc filtering coefficients.
308.  *                  Guaranteed to contain dstW * filterSize entries.
309.  * @param filterPos position of the first input pixel to be used for
310.  *                  each output pixel during scaling. Guaranteed to
311.  *                  contain dstW entries.
312.  * @param filterSize the number of input coefficients to be used (and
313.  *                  thus the number of input pixels to be used) for
314.  *                  creating a single output pixel. Is aligned to 4
315.  *                  (and input coefficients thus padded with zeroes)
316.  *                  to simplify creating SIMD code.
317.  */
318. /** @{ */
319. void (*hyScale)(struct SwsContext *c, int16_t *dst, int dstW,
320.                const uint8_t *src, const int16_t *filter,
321.                const int32_t *filterPos, int filterSize);
322. void (*hcScale)(struct SwsContext *c, int16_t *dst, int dstW,
323.                const uint8_t *src, const int16_t *filter,
324.                const int32_t *filterPos, int filterSize);
325. /** @} */
326.
327. /// Color range conversion function for luma plane if needed.
328. void (*lumConvertRange)(int16_t *dst, int width);
329. /// Color range conversion function for chroma planes if needed.
330. void (*chrConvertRange)(int16_t *dst1, int16_t *dst2, int width);
331.
332. int needs_hcscale; ///< Set if there are chroma planes to be converted.
333.
334. SwsDither dither;
335. } SwsContext;

```

这个结构体的定义确实比较复杂，里面包含了libswscale所需要的所有变量。——分析这些变量是不太现实的，在后文中会简单分析其中的几个变量。

sws_getContext()

sws_getContext()是初始化SwsContext的函数。sws_getContext()的声明位于libswscale\swscale.h，如下所示。

```

1.  /**
2.   * Allocate and return an SwsContext. You need it to perform
3.   * scaling/conversion operations using sws_scale().
4.   *
5.   * @param srcW the width of the source image
6.   * @param srcH the height of the source image
7.   * @param srcFormat the source image format
8.   * @param dstW the width of the destination image
9.   * @param dstH the height of the destination image
10.  * @param dstFormat the destination image format
11.  * @param flags specify which algorithm and options to use for rescaling
12.  * @return a pointer to an allocated context, or NULL in case of error
13.  * @note this function is to be removed after a saner alternative is
14.  *       written
15.  */
16.  struct SwsContext *sws_getContext(int srcW, int srcH, enum AVPixelFormat srcFormat,
17.                                   int dstW, int dstH, enum AVPixelFormat dstFormat,
18.                                   int flags, SwsFilter *srcFilter,
19.                                   SwsFilter *dstFilter, const double *param);

```

该函数包含以下参数：

srcW：源图像的宽

srcH：源图像的高

srcFormat：源图像的像素格式

dstW：目标图像的宽

dstH：目标图像的高

dstFormat：目标图像的像素格式

flags：设定图像拉伸使用的算法

成功执行的话返回生成的SwsContext，否则返回NULL。

sws_getContext()的定义位于libswscaleutils.c，如下所示。

```

1.  SwsContext *sws_getContext(int srcW, int srcH, enum AVPixelFormat srcFormat,
2.                             int dstW, int dstH, enum AVPixelFormat dstFormat,
3.                             int flags, SwsFilter *srcFilter,
4.                             SwsFilter *dstFilter, const double *param)
5.  {
6.      SwsContext *c;
7.
8.      if (!(c = sws_alloc_context()))
9.          return NULL;
10.
11.      c->flags      = flags;
12.      c->srcW       = srcW;
13.      c->srcH       = srcH;
14.      c->dstW       = dstW;
15.      c->dstH       = dstH;
16.      c->srcFormat  = srcFormat;
17.      c->dstFormat  = dstFormat;
18.
19.      if (param) {
20.          c->param[0] = param[0];
21.          c->param[1] = param[1];
22.      }
23.
24.      if (sws_init_context(c, srcFilter, dstFilter) < 0) {
25.          sws_freeContext(c);
26.          return NULL;
27.      }
28.
29.      return c;
30.  }

```

从sws_getContext()的定义中可以看出，它首先调用了函数sws_alloc_context()用于给SwsContext分配内存。然后将传入的源图像，目标图像的宽高，像素格式，以及标志位分别赋值给该SwsContext相应的字段。最后调用一个函数sws_init_context()完成初始化工作。下面我们分别看一下sws_alloc_context()和sws_init_context()这两个函数。

sws_alloc_context()

sws_alloc_context()是FFmpeg的一个API，用于给SwsContext分配内存，它的声明如下所示。

```

1.  /**
2.   * Allocate an empty SwsContext. This must be filled and passed to
3.   * sws_init_context(). For filling see AVOptions, options.c and
4.   * sws_setColorspaceDetails().
5.   */
6.  struct SwsContext *sws_alloc_context(void);

```

sws_alloc_context()的定义位于libswscale\utils.c, 如下所示。

```

1.  SwsContext *sws_alloc_context(void)
2.  {
3.      SwsContext *c = av_mallocz(sizeof(SwsContext));
4.
5.      av_assert0(offsetof(SwsContext, redDither) + DITHER32_INT == offsetof(SwsContext, dither32));
6.
7.      if (c) {
8.          c->av_class = &sws_context_class;
9.          av_opt_set_defaults(c);
10.     }
11.
12.     return c;
13. }

```

从代码中可以看出, sws_alloc_context()首先调用av_mallocz()为SwsContext(结构体分配了一块内存; 然后设置了该结构体的AVClass, 并且给该结构体的字段设置了默认值。

sws_init_context()

sws_init_context()的是FFmpeg的一个API, 用于初始化SwsContext。

```

1.  /**
2.   * Initialize the swscaler context sws_context.
3.   *
4.   * @return zero or positive value on success, a negative value on
5.   * error
6.   */
7.  int sws_init_context(struct SwsContext *sws_context, SwsFilter *srcFilter, SwsFilter *dstFilter);

```

sws_init_context()的函数定义非常的长, 位于libswscale\utils.c, 如下所示。

```

1.  av_cold int sws_init_context(SwsContext *c, SwsFilter *srcFilter,
2.                              SwsFilter *dstFilter)
3.  {
4.      int i, j;
5.      int usesVFilter, usesHFilter;
6.      int unscaled;
7.      SwsFilter dummyFilter = { NULL, NULL, NULL, NULL };
8.      int srcW      = c->srcW;
9.      int srcH      = c->srcH;
10.     int dstW      = c->dstW;
11.     int dstH      = c->dstH;
12.     int dst_stride = FFALIGN(dstW * sizeof(int16_t) + 66, 16);
13.     int flags, cpu_flags;
14.     enum AVPixelFormat srcFormat = c->srcFormat;
15.     enum AVPixelFormat dstFormat = c->dstFormat;
16.     const AVPixFmtDescriptor *desc_src;
17.     const AVPixFmtDescriptor *desc_dst;
18.     int ret = 0;
19.     //获取
20.     cpu_flags = av_get_cpu_flags();
21.     flags     = c->flags;
22.     emms_c();
23.     if (!rgb15to16)
24.         sws_rgb2rgb_init();
25.     //如果输入的宽高和输出的宽高一样, 则做特殊处理
26.     unscaled = (srcW == dstW && srcH == dstH);
27.     //如果是JPEG标准 (Y取值0-255), 则需要设置这两项
28.     c->srcRange |= handle_jpeg(&c->srcFormat);
29.     c->dstRange |= handle_jpeg(&c->dstFormat);
30.
31.     if (srcFormat != c->srcFormat || dstFormat != c->dstFormat)
32.         av_log(c, AV_LOG_WARNING, "deprecated pixel format used, make sure you did set range correctly\n");
33.     //设置Colorspace
34.     if (!c->contrast && !c->saturation && !c->dstFormatBpp)
35.         sws_setColorspaceDetails(c, ff_yuv2rgb_coeffs[SWS_CS_DEFAULT], c->srcRange,
36.                                 ff_yuv2rgb_coeffs[SWS_CS_DEFAULT],
37.                                 c->dstRange, 0, 1 << 16, 1 << 16);
38. }

```



```

39.     handle_formats(c);
40.     srcFormat = c->srcFormat;
41.     dstFormat = c->dstFormat;
42.     desc_src = av_pix_fmt_desc_get(srcFormat);
43.     desc_dst = av_pix_fmt_desc_get(dstFormat);
44.     //转换大小端?
45.     if (!(unscaled && sws_isSupportedEndiannessConversion(srcFormat) &&
46.         av_pix_fmt_swap_endianness(srcFormat) == dstFormat)) {
47.         //检查输入格式是否支持
48.         if (!sws_isSupportedInput(srcFormat)) {
49.             av_log(c, AV_LOG_ERROR, "%s is not supported as input pixel format\n",
50.                 av_get_pix_fmt_name(srcFormat));
51.             return AERROR(EINVAL);
52.         }
53.         //检查输出格式是否支持
54.         if (!sws_isSupportedOutput(dstFormat)) {
55.             av_log(c, AV_LOG_ERROR, "%s is not supported as output pixel format\n",
56.                 av_get_pix_fmt_name(dstFormat));
57.             return AERROR(EINVAL);
58.         }
59.     }
60.     //检查拉伸的方法
61.     i = flags & (SWS_POINT |
62.        SWS_AREA |
63.        SWS_BILINEAR |
64.        SWS_FAST_BILINEAR |
65.        SWS_BICUBIC |
66.        SWS_X |
67.        SWS_GAUSS |
68.        SWS_LANCZOS |
69.        SWS_SINC |
70.        SWS_SPLINE |
71.        SWS_BICUBLIN);
72.
73.     /* provide a default scaler if not set by caller */
74.     //如果没有指定,就使用默认的
75.     if (!i) {
76.         if (dstW < srcW && dstH < srcH)
77.             flags |= SWS_BICUBIC;
78.         else if (dstW > srcW && dstH > srcH)
79.             flags |= SWS_BICUBIC;
80.         else
81.             flags |= SWS_BICUBIC;
82.         c->flags = flags;
83.     } else if (i & (i - 1)) {
84.         av_log(c, AV_LOG_ERROR,
85.             "Exactly one scaler algorithm must be chosen, got %X\n", i);
86.         return AERROR(EINVAL);
87.     }
88.     /* sanity check */
89.     //检查宽高参数
90.     if (srcW < 1 || srcH < 1 || dstW < 1 || dstH < 1) {
91.         /* FIXME check if these are enough and try to lower them after
92.          * fixing the relevant parts of the code */
93.         av_log(c, AV_LOG_ERROR, "%dx%d -> %dx%d is invalid scaling dimension\n",
94.             srcW, srcH, dstW, dstH);
95.         return AERROR(EINVAL);
96.     }
97.
98.     if (!dstFilter)
99.         dstFilter = &dummyFilter;
100.    if (!srcFilter)
101.        srcFilter = &dummyFilter;
102.
103.    c->lumXInc = (((int64_t)srcW << 16) + (dstW >> 1)) / dstW;
104.    c->lumYInc = (((int64_t)srcH << 16) + (dstH >> 1)) / dstH;
105.    c->dstFormatBpp = av_get_bits_per_pixel(desc_dst);
106.    c->srcFormatBpp = av_get_bits_per_pixel(desc_src);
107.    c->vRounder = 4 * 0x0001000100010001ULL;
108.
109.    usesVFilter = (srcFilter->lumV && srcFilter->lumV->length > 1) ||
110.        (srcFilter->chrV && srcFilter->chrV->length > 1) ||
111.        (dstFilter->lumV && dstFilter->lumV->length > 1) ||
112.        (dstFilter->chrV && dstFilter->chrV->length > 1);
113.    usesHFilter = (srcFilter->lumH && srcFilter->lumH->length > 1) ||
114.        (srcFilter->chrH && srcFilter->chrH->length > 1) ||
115.        (dstFilter->lumH && dstFilter->lumH->length > 1) ||
116.        (dstFilter->chrH && dstFilter->chrH->length > 1);
117.
118.    av_pix_fmt_get_chroma_sub_sample(srcFormat, &c->chrSrcSubSample, &c->chrSrcVSubSample);
119.    av_pix_fmt_get_chroma_sub_sample(dstFormat, &c->chrDstSubSample, &c->chrDstVSubSample);
120.
121.    if (isAnyRGB(dstFormat) && !(flags&SWS_FULL_CHR_H_INT)) {
122.        if (dstW&1) {
123.            av_log(c, AV_LOG_DEBUG, "Forcing full internal H chroma due to odd output size\n");
124.            flags |= SWS_FULL_CHR_H_INT;
125.            c->flags = flags;
126.        }
127.
128.        if (c->chrSrcSubSample == 0
129.            && c->chrSrcVSubSample == 0
130.            && c->dstFormat != SWS_ETHER_DAYER) { //SWS_FULL_CHR_H_INT is currently not supported with SWS_ETHER_DAYER

```

```

130.         && c->dither != SWS_DITHER_BAYER //SWS_FULL_CHR_H_INT is currently not supported with SWS_DITHER_BAYER
131.         && !(c->flags & SWS_FAST_BILINEAR)
132.     ) {
133.         av_log(c, AV_LOG_DEBUG, "Forcing full internal H chroma due to input having non subsampled chroma\n");
134.         flags |= SWS_FULL_CHR_H_INT;
135.         c->flags = flags;
136.     }
137. }
138.
139. if (c->dither == SWS_DITHER_AUTO) {
140.     if (flags & SWS_ERROR_DIFFUSION)
141.         c->dither = SWS_DITHER_ED;
142. }
143.
144. if(dstFormat == AV_PIX_FMT_BGR4_BYTE ||
145.    dstFormat == AV_PIX_FMT_RGBA_BYTE ||
146.    dstFormat == AV_PIX_FMT_BGR8 ||
147.    dstFormat == AV_PIX_FMT_RGB8) {
148.     if (c->dither == SWS_DITHER_AUTO)
149.         c->dither = (flags & SWS_FULL_CHR_H_INT) ? SWS_DITHER_ED : SWS_DITHER_BAYER;
150.     if (!(flags & SWS_FULL_CHR_H_INT)) {
151.         if (c->dither == SWS_DITHER_ED || c->dither == SWS_DITHER_A_DITHER || c->dither == SWS_DITHER_X_DITHER) {
152.             av_log(c, AV_LOG_DEBUG,
153.                  "Desired dithering only supported in full chroma interpolation for destination format '%s'\n",
154.                  av_get_pix_fmt_name(dstFormat));
155.             flags |= SWS_FULL_CHR_H_INT;
156.             c->flags = flags;
157.         }
158.     }
159.     if (flags & SWS_FULL_CHR_H_INT) {
160.         if (c->dither == SWS_DITHER_BAYER) {
161.             av_log(c, AV_LOG_DEBUG,
162.                  "Ordered dither is not supported in full chroma interpolation for destination format '%s'\n",
163.                  av_get_pix_fmt_name(dstFormat));
164.             c->dither = SWS_DITHER_ED;
165.         }
166.     }
167. }
168. if (isPlanarRGB(dstFormat)) {
169.     if (!(flags & SWS_FULL_CHR_H_INT)) {
170.         av_log(c, AV_LOG_DEBUG,
171.              "%s output is not supported with half chroma resolution, switching to full\n",
172.              av_get_pix_fmt_name(dstFormat));
173.         flags |= SWS_FULL_CHR_H_INT;
174.         c->flags = flags;
175.     }
176. }
177.
178. /* reuse chroma for 2 pixels RGB/BGR unless user wants full
179.  * chroma interpolation */
180. if (flags & SWS_FULL_CHR_H_INT &&
181.     isAnyRGB(dstFormat) &&
182.     !isPlanarRGB(dstFormat) &&
183.     dstFormat != AV_PIX_FMT_RGBA &&
184.     dstFormat != AV_PIX_FMT_ARGB &&
185.     dstFormat != AV_PIX_FMT_BGRA &&
186.     dstFormat != AV_PIX_FMT_ABGR &&
187.     dstFormat != AV_PIX_FMT_RGB24 &&
188.     dstFormat != AV_PIX_FMT_BGR24 &&
189.     dstFormat != AV_PIX_FMT_BGR4_BYTE &&
190.     dstFormat != AV_PIX_FMT_RGBA_BYTE &&
191.     dstFormat != AV_PIX_FMT_BGR8 &&
192.     dstFormat != AV_PIX_FMT_RGB8)
193. ) {
194.     av_log(c, AV_LOG_WARNING,
195.          "full chroma interpolation for destination format '%s' not yet implemented\n",
196.          av_get_pix_fmt_name(dstFormat));
197.     flags &= ~SWS_FULL_CHR_H_INT;
198.     c->flags = flags;
199. }
200. if (isAnyRGB(dstFormat) && !(flags & SWS_FULL_CHR_H_INT))
201.     c->chrDstHSubSample = 1;
202.
203. // drop some chroma lines if the user wants it
204. c->vChrDrop = (flags & SWS_SRC_V_CHR_DROP_MASK) >>
205.              SWS_SRC_V_CHR_DROP_SHIFT;
206. c->chrSrcVSubSample += c->vChrDrop;
207.
208. /* drop every other pixel for chroma calculation unless user
209.  * wants full chroma */
210. if (isAnyRGB(srcFormat) && !(flags & SWS_FULL_CHR_H_INP) &&
211.     srcFormat != AV_PIX_FMT_RGB8 && srcFormat != AV_PIX_FMT_BGR8 &&
212.     srcFormat != AV_PIX_FMT_RGBA && srcFormat != AV_PIX_FMT_BGRA &&
213.     srcFormat != AV_PIX_FMT_RGBA_BYTE && srcFormat != AV_PIX_FMT_BGRA_BYTE &&
214.     srcFormat != AV_PIX_FMT_GBRP9BE && srcFormat != AV_PIX_FMT_GBRP9LE &&
215.     srcFormat != AV_PIX_FMT_GBRP10BE && srcFormat != AV_PIX_FMT_GBRP10LE &&
216.     srcFormat != AV_PIX_FMT_GBRP12BE && srcFormat != AV_PIX_FMT_GBRP12LE &&
217.     srcFormat != AV_PIX_FMT_GBRP14BE && srcFormat != AV_PIX_FMT_GBRP14LE &&
218.     srcFormat != AV_PIX_FMT_GBRP16BE && srcFormat != AV_PIX_FMT_GBRP16LE &&
219.     ((dstW >> c->chrDstHSubSample) <= (srcW >> 1) ||
220.      (flags & SWS_FAST_BILINEAR)))
221.     c->chrSrcHSubSample = 1;

```

```

221.     c->chrSrcSubSample = 1;
222.
223.     // Note the FF_CEIL_RSHIFT is so that we always round toward +inf.
224.     c->chrSrcW = FF_CEIL_RSHIFT(srcW, c->chrSrcHSubSample);
225.     c->chrSrcH = FF_CEIL_RSHIFT(srcH, c->chrSrcVSubSample);
226.     c->chrDstW = FF_CEIL_RSHIFT(dstW, c->chrDstHSubSample);
227.     c->chrDstH = FF_CEIL_RSHIFT(dstH, c->chrDstVSubSample);
228.
229.     FF_ALLOC_OR_GOTO(c, c->formatConvBuffer, FFALIGN(srcW*2+78, 16) * 2, fail);
230.
231.     c->srcBpc = 1 + desc_src->comp[0].depth_minus1;
232.     if (c->srcBpc < 8)
233.         c->srcBpc = 8;
234.     c->dstBpc = 1 + desc_dst->comp[0].depth_minus1;
235.     if (c->dstBpc < 8)
236.         c->dstBpc = 8;
237.     if (isAnyRGB(srcFormat) || srcFormat == AV_PIX_FMT_PAL8)
238.         c->srcBpc = 16;
239.     if (c->dstBpc == 16)
240.         dst_stride <= 1;
241.
242.     if (INLINE_MMEXT(cpu_flags) && c->srcBpc == 8 && c->dstBpc <= 14) {
243.         c->canMMEXTBeUsed = dstW >= srcW && (dstW & 31) == 0 &&
244.             c->chrDstW >= c->chrSrcW &&
245.             (srcW & 15) == 0;
246.         if (!c->canMMEXTBeUsed && dstW >= srcW && c->chrDstW >= c->chrSrcW && (srcW & 15) == 0
247.
248.             && (flags & SWS_FAST_BILINEAR)) {
249.             if (flags & SWS_PRINT_INFO)
250.                 av_log(c, AV_LOG_INFO,
251.                     "output width is not a multiple of 32 -> no MMEXT scaler\n");
252.         }
253.         if (usesHFilter || isNBPS(c->srcFormat) || is16BPS(c->srcFormat) || isAnyRGB(c->srcFormat))
254.             c->canMMEXTBeUsed = 0;
255.     } else
256.         c->canMMEXTBeUsed = 0;
257.
258.     c->chrXInc = (((int64_t)c->chrSrcW << 16) + (c->chrDstW >> 1)) / c->chrDstW;
259.     c->chrYInc = (((int64_t)c->chrSrcH << 16) + (c->chrDstH >> 1)) / c->chrDstH;
260.
261.     /* Match pixel 0 of the src to pixel 0 of dst and match pixel n-2 of src
262.     * to pixel n-2 of dst, but only for the FAST_BILINEAR mode otherwise do
263.     * correct scaling.
264.     * n-2 is the last chrominance sample available.
265.     * This is not perfect, but no one should notice the difference, the more
266.     * correct variant would be like the vertical one, but that would require
267.     * some special code for the first and last pixel */
268.     if (flags & SWS_FAST_BILINEAR) {
269.         if (c->canMMEXTBeUsed) {
270.             c->lumXInc += 20;
271.             c->chrXInc += 20;
272.         }
273.         // we don't use the x86 asm scaler if MMX is available
274.         else if (INLINE_MMX(cpu_flags) && c->dstBpc <= 14) {
275.             c->lumXInc = ((int64_t)(srcW - 2) << 16) / (dstW - 2) - 20;
276.             c->chrXInc = ((int64_t)(c->chrSrcW - 2) << 16) / (c->chrDstW - 2) - 20;
277.         }
278.     }
279.
280.     if (isBayer(srcFormat)) {
281.         if (!unscaled ||
282.             (dstFormat != AV_PIX_FMT_RGB24 && dstFormat != AV_PIX_FMT_YUV420P)) {
283.             enum AVPixelFormat tmpFormat = AV_PIX_FMT_RGB24;
284.
285.             ret = av_image_alloc(c->cascaded_tmp, c->cascaded_tmpStride,
286.                                 srcW, srcH, tmpFormat, 64);
287.             if (ret < 0)
288.                 return ret;
289.
290.             c->cascaded_context[0] = sws_getContext(srcW, srcH, srcFormat,
291.                                                    srcW, srcH, tmpFormat,
292.                                                    flags, srcFilter, NULL, c->param);
293.             if (!c->cascaded_context[0])
294.                 return -1;
295.
296.             c->cascaded_context[1] = sws_getContext(srcW, srcH, tmpFormat,
297.                                                    dstW, dstH, dstFormat,
298.                                                    flags, NULL, dstFilter, c->param);
299.             if (!c->cascaded_context[1])
300.                 return -1;
301.             return 0;
302.         }
303.     }
304.
305. #define USE_MMAP (HAVE_MMAP && HAVE_MPROTECT && defined MAP_ANONYMOUS)
306.
307.     /* precalculate horizontal scaler filter coefficients */
308.     {
309. #if HAVE_MMEXT_INLINE
310.         // can't downscale !!!
311.         if (c->canMMEXTBeUsed && (flags & SWS_FAST_BILINEAR)) {
312.             c->lumMmxextFilterCodeSize = ff_init_hscaler_mmxext(dstW, c->lumXInc, NULL,

```

```

313.         NULL, NULL, 8);
314.     c->chrMmxextFilterCodeSize = ff_init_hscaler_mmxext(c->chrDstW, c->chrXInc,
315.         NULL, NULL, NULL, 4);
316.
317. #if USE_MMAP
318.     c->lumMmxextFilterCode = mmap(NULL, c->lumMmxextFilterCodeSize,
319.         PROT_READ | PROT_WRITE,
320.         MAP_PRIVATE | MAP_ANONYMOUS,
321.         -1, 0);
322.     c->chrMmxextFilterCode = mmap(NULL, c->chrMmxextFilterCodeSize,
323.         PROT_READ | PROT_WRITE,
324.         MAP_PRIVATE | MAP_ANONYMOUS,
325.         -1, 0);
326. #elif HAVE_VIRTUALALLOC
327.     c->lumMmxextFilterCode = VirtualAlloc(NULL,
328.         c->lumMmxextFilterCodeSize,
329.         MEM_COMMIT,
330.         PAGE_EXECUTE_READWRITE);
331.     c->chrMmxextFilterCode = VirtualAlloc(NULL,
332.         c->chrMmxextFilterCodeSize,
333.         MEM_COMMIT,
334.         PAGE_EXECUTE_READWRITE);
335. #else
336.     c->lumMmxextFilterCode = av_malloc(c->lumMmxextFilterCodeSize);
337.     c->chrMmxextFilterCode = av_malloc(c->chrMmxextFilterCodeSize);
338. #endif
339.
340. #ifndef MAP_ANONYMOUS
341.     if (c->lumMmxextFilterCode == MAP_FAILED || c->chrMmxextFilterCode == MAP_FAILED)
342. #else
343.     if (!c->lumMmxextFilterCode || !c->chrMmxextFilterCode)
344. #endif
345.     {
346.         av_log(c, AV_LOG_ERROR, "Failed to allocate MMX2FilterCode\n");
347.         return AVERROR(ENOMEM);
348.     }
349.
350.     FF_ALLOCZ_OR_GOTO(c, c->hLumFilter, (dstW / 8 + 8) * sizeof(int16_t), fail);
351.     FF_ALLOCZ_OR_GOTO(c, c->hChrFilter, (c->chrDstW / 4 + 8) * sizeof(int16_t), fail);
352.     FF_ALLOCZ_OR_GOTO(c, c->hLumFilterPos, (dstW / 2 / 8 + 8) * sizeof(int32_t), fail);
353.     FF_ALLOCZ_OR_GOTO(c, c->hChrFilterPos, (c->chrDstW / 2 / 4 + 8) * sizeof(int32_t), fail);
354.
355.     ff_init_hscaler_mmxext(dstW, c->lumXInc, c->lumMmxextFilterCode,
356.         c->hLumFilter, (uint32_t*)c->hLumFilterPos, 8);
357.     ff_init_hscaler_mmxext(c->chrDstW, c->chrXInc, c->chrMmxextFilterCode,
358.         c->hChrFilter, (uint32_t*)c->hChrFilterPos, 4);
359.
360. #if USE_MMAP
361.     if ( mprotect(c->lumMmxextFilterCode, c->lumMmxextFilterCodeSize, PROT_EXEC | PROT_READ) == -1
362.         || mprotect(c->chrMmxextFilterCode, c->chrMmxextFilterCodeSize, PROT_EXEC | PROT_READ) == -1) {
363.         av_log(c, AV_LOG_ERROR, "mprotect failed, cannot use fast bilinear scaler\n");
364.         goto fail;
365.     }
366. #endif
367. } else
368. #endif /* HAVE_MMEXT_INLINE */
369. {
370.     const int filterAlign = X86_MMX(cpu_flags) ? 4 :
371.         PPC_ALTIVEC(cpu_flags) ? 8 : 1;
372.
373.     if ((ret = initFilter(&c->hLumFilter, &c->hLumFilterPos,
374.         &c->hLumFilterSize, c->lumXInc,
375.         srcW, dstW, filterAlign, 1 << 14,
376.         (flags & SWS_BICUBLIN) ? (flags | SWS_BICUBIC) : flags,
377.         cpu_flags, srcFilter->lumH, dstFilter->lumH,
378.         c->param,
379.         get_local_pos(c, 0, 0, 0),
380.         get_local_pos(c, 0, 0, 0))) < 0)
381.         goto fail;
382.     if ((ret = initFilter(&c->hChrFilter, &c->hChrFilterPos,
383.         &c->hChrFilterSize, c->chrXInc,
384.         c->chrSrcW, c->chrDstW, filterAlign, 1 << 14,
385.         (flags & SWS_BICUBLIN) ? (flags | SWS_BILINEAR) : flags,
386.         cpu_flags, srcFilter->chrH, dstFilter->chrH,
387.         c->param,
388.         get_local_pos(c, c->chrSrcSubSample, c->src_h_chr_pos, 0),
389.         get_local_pos(c, c->chrDstSubSample, c->dst_h_chr_pos, 0))) < 0)
390.         goto fail;
391. }
392. } // initialize horizontal stuff
393.
394. /* precalculate vertical scaler filter coefficients */
395. {
396.     const int filterAlign = X86_MMX(cpu_flags) ? 2 :
397.         PPC_ALTIVEC(cpu_flags) ? 8 : 1;
398.
399.     if ((ret = initFilter(&c->vLumFilter, &c->vLumFilterPos, &c->vLumFilterSize,
400.         c->lumYInc, srcH, dstH, filterAlign, (1 << 12),
401.         (flags & SWS_BICUBLIN) ? (flags | SWS_BICUBIC) : flags,
402.         cpu_flags, srcFilter->lumV, dstFilter->lumV,
403.         c->param,

```

```

404.         get_local_pos(c, 0, 0, 1),
405.         get_local_pos(c, 0, 0, 1))) < 0)
406.     goto fail;
407.     if ((ret = initFilter(&c->vChrFilter, &c->vChrFilterPos, &c->vChrFilterSize,
408.         c->chrYInC, c->chrSrcH, c->chrDstH,
409.         filterAlign, (1 < 12),
410.         (flags & SWS_BICUBLIN) ? (flags | SWS_BILINEAR) : flags,
411.         cpu_flags, srcFilter->chrV, dstFilter->chrV,
412.         c->param,
413.         get_local_pos(c, c->chrSrcVSubSample, c->src_v_chr_pos, 1),
414.         get_local_pos(c, c->chrDstVSubSample, c->dst_v_chr_pos, 1))) < 0)
415.
416.         goto fail;
417.
418. #if HAVE_ALTIVEC
419.     FF_ALLOC_OR_GOTO(c, c->vYCoeffsBank, sizeof(vector signed short) * c->vLumFilterSize * c->dstH, fail);
420.     FF_ALLOC_OR_GOTO(c, c->vCCoeffsBank, sizeof(vector signed short) * c->vChrFilterSize * c->chrDstH, fail);
421.
422.     for (i = 0; i < c->vLumFilterSize * c->dstH; i++) {
423.         int j;
424.         short *p = (short *) &c->vYCoeffsBank[i];
425.         for (j = 0; j < 8; j++)
426.             p[j] = c->vLumFilter[i];
427.     }
428.
429.     for (i = 0; i < c->vChrFilterSize * c->chrDstH; i++) {
430.         int j;
431.         short *p = (short *) &c->vCCoeffsBank[i];
432.         for (j = 0; j < 8; j++)
433.             p[j] = c->vChrFilter[i];
434.     }
435. #endif
436. }
437.
438. // calculate buffer sizes so that they won't run out while handling these damn slices
439. c->vLumBufSize = c->vLumFilterSize;
440. c->vChrBufSize = c->vChrFilterSize;
441. for (i = 0; i < dstH; i++) {
442.     int chrI = (int64_t)i * c->chrDstH / dstH;
443.     int nextSlice = FFMAX(c->vLumFilterPos[i] + c->vLumFilterSize - 1,
444.         ((c->vChrFilterPos[chrI] + c->vChrFilterSize - 1)
445.         << c->chrSrcVSubSample));
446.
447.     nextSlice >= c->chrSrcVSubSample;
448.     nextSlice <= c->chrSrcVSubSample;
449.     if (c->vLumFilterPos[i] + c->vLumBufSize < nextSlice)
450.         c->vLumBufSize = nextSlice - c->vLumFilterPos[i];
451.     if (c->vChrFilterPos[chrI] + c->vChrBufSize <
452.         (nextSlice >> c->chrSrcVSubSample))
453.         c->vChrBufSize = (nextSlice >> c->chrSrcVSubSample) -
454.             c->vChrFilterPos[chrI];
455. }
456.
457. for (i = 0; i < 4; i++)
458.     FF_ALLOCZ_OR_GOTO(c, c->dither_error[i], (c->dstW+2) * sizeof(int), fail);
459.
460. /* Allocate pixbufs (we use dynamic allocation because otherwise we would
461.  * need to allocate several megabytes to handle all possible cases) */
462. FF_ALLOC_OR_GOTO(c, c->lumPixBuf, c->vLumBufSize * 3 * sizeof(int16_t *), fail);
463. FF_ALLOC_OR_GOTO(c, c->chrUPixBuf, c->vChrBufSize * 3 * sizeof(int16_t *), fail);
464. FF_ALLOC_OR_GOTO(c, c->chrVPixBuf, c->vChrBufSize * 3 * sizeof(int16_t *), fail);
465. if (CONFIG_SWSCALE_ALPHA && isALPHA(c->srcFormat) && isALPHA(c->dstFormat))
466.     FF_ALLOCZ_OR_GOTO(c, c->alpPixBuf, c->vLumBufSize * 3 * sizeof(int16_t *), fail);
467. /* Note we need at least one pixel more at the end because of the MMX code
468.  * (just in case someone wants to replace the 4000/8000). */
469. /* align at 16 bytes for AltiVec */
470. for (i = 0; i < c->vLumBufSize; i++) {
471.     FF_ALLOCZ_OR_GOTO(c, c->lumPixBuf[i + c->vLumBufSize],
472.         dst_stride + 16, fail);
473.     c->lumPixBuf[i] = c->lumPixBuf[i + c->vLumBufSize];
474. }
475. // 64 / c->scalingBpp is the same as 16 / sizeof(scaling_intermediate)
476. c->uv_off = (dst_stride >> 1) + 64 / (c->dstBpc &~ 7);
477. c->uv_offx2 = dst_stride + 16;
478. for (i = 0; i < c->vChrBufSize; i++) {
479.     FF_ALLOC_OR_GOTO(c, c->chrUPixBuf[i + c->vChrBufSize],
480.         dst_stride * 2 + 32, fail);
481.     c->chrUPixBuf[i] = c->chrUPixBuf[i + c->vChrBufSize];
482.     c->chrVPixBuf[i] = c->chrVPixBuf[i + c->vChrBufSize]
483.         = c->chrUPixBuf[i] + (dst_stride >> 1) + 8;
484. }
485. if (CONFIG_SWSCALE_ALPHA && c->alpPixBuf)
486.     for (i = 0; i < c->vLumBufSize; i++) {
487.         FF_ALLOCZ_OR_GOTO(c, c->alpPixBuf[i + c->vLumBufSize],
488.             dst_stride + 16, fail);
489.         c->alpPixBuf[i] = c->alpPixBuf[i + c->vLumBufSize];
490.     }
491.
492. // try to avoid drawing green stuff between the right end and the stride end
493. for (i = 0; i < c->vChrBufSize; i++)
494.     if(desc_dst->comp[0].depth_minus1 == 15){

```

```

495.         av_assert0(c->dstBpc > 14);
496.         for(j=0; j<dst_stride/2+1; j++)
497.             ((int32_t*)(c->chrUPixBuf[i]))[j] = 1<<18;
498.     } else
499.         for(j=0; j<dst_stride+1; j++)
500.             ((int16_t*)(c->chrUPixBuf[i]))[j] = 1<<14;
501.
502.     av_assert0(c->chrDsth <= dstH);
503.     //是否要输出
504.     if (flags & SWS_PRINT_INFO) {
505.         const char *scaler = NULL, *cpucaps;
506.
507.         for (i = 0; i < FF_ARRAY_ELEMS(scale_algorithms); i++) {
508.             if (flags & scale_algorithms[i].flag) {
509.                 scaler = scale_algorithms[i].description;
510.                 break;
511.             }
512.         }
513.         if (!scaler)
514.             scaler = "ehh flags invalid?!";
515.         av_log(c, AV_LOG_INFO, "%s scaler, from %s to %s ",
516.             scaler,
517.             av_get_pix_fmt_name(srcFormat),
518. #ifdef DITHER1XBPB
519.             dstFormat == AV_PIX_FMT_BGR555 || dstFormat == AV_PIX_FMT_BGR565 ||
520.             dstFormat == AV_PIX_FMT_RGB444BE || dstFormat == AV_PIX_FMT_RGB444LE ||
521.             dstFormat == AV_PIX_FMT_BGR444BE || dstFormat == AV_PIX_FMT_BGR444LE ?
522.                 "dithered " : "",
523. #else
524.             "",
525. #endif
526.             av_get_pix_fmt_name(dstFormat));
527.
528.         if (INLINE_MMEXT(cpu_flags))
529.             cpucaps = "MMEXT";
530.         else if (INLINE_AMD3DNOW(cpu_flags))
531.             cpucaps = "3DNOW";
532.         else if (INLINE_MMX(cpu_flags))
533.             cpucaps = "MMX";
534.         else if (PPC_ALTIVEC(cpu_flags))
535.             cpucaps = "AltiVec";
536.         else
537.             cpucaps = "C";
538.
539.         av_log(c, AV_LOG_INFO, "using %s\n", cpucaps);
540.
541.         av_log(c, AV_LOG_VERBOSE, "%dx%d -> %dx%d\n", srcW, srcH, dstW, dstH);
542.         av_log(c, AV_LOG_DEBUG,
543.             "lum srcW=%d srcH=%d dstW=%d dstH=%d xInc=%d yInc=%d\n",
544.             c->srcW, c->srcH, c->dstW, c->dstH, c->lumXInc, c->lumYInc);
545.         av_log(c, AV_LOG_DEBUG,
546.             "chr srcW=%d srcH=%d dstW=%d dstH=%d xInc=%d yInc=%d\n",
547.             c->chrSrcW, c->chrSrcH, c->chrDstW, c->chrDstH,
548.             c->chrXInc, c->chrYInc);
549.     }
550.
551.     /* unscaled special cases */
552.     //不拉伸的情况
553.     if (unscaled && !usesHFilter && !usesVFilter &&
554.         (c->srcRange == c->dstRange || isAnyRGB(dstFormat))) {
555.         //不许拉伸的情况下, 初始化相应的函数
556.         ff_get_unscaled_swscale(c);
557.
558.         if (c->swscale) {
559.             if (flags & SWS_PRINT_INFO)
560.                 av_log(c, AV_LOG_INFO,
561.                     "using unscaled %s -> %s special converter\n",
562.                     av_get_pix_fmt_name(srcFormat), av_get_pix_fmt_name(dstFormat));
563.             return 0;
564.         }
565.     }
566.     //关键: 设置SwsContext中的swscale()指针
567.     c->swscale = ff_getSwsFunc(c);
568.     return 0;
569. fail: // FIXME replace things by appropriate error codes
570.     if (ret == RETCODE_USE_CASCADE) {
571.         int tmpW = sqrt(srcW * (int64_t)dstW);
572.         int tmpH = sqrt(srcH * (int64_t)dstH);
573.         enum AVPixelFormat tmpFormat = AV_PIX_FMT_YUV420P;
574.
575.         if (srcW*(int64_t)srcH <= 4LL*dstW*dstH)
576.             return AVERROR(EINVAL);
577.
578.         ret = av_image_alloc(c->cascaded_tmp, c->cascaded_tmpStride,
579.             tmpW, tmpH, tmpFormat, 64);
580.         if (ret < 0)
581.             return ret;
582.
583.         c->cascaded_context[0] = sws_getContext(srcW, srcH, srcFormat,
584.             tmpW, tmpH, tmpFormat,
585.             flags, srcFilter, NULL, c->param);

```

```

586.         if (!c->cascaded_context[0])
587.             return -1;
588.
589.         c->cascaded_context[1] = sws_getContext(tmpW, tmpH, tmpFormat,
590.                                                 dstW, dstH, dstFormat,
591.                                                 flags, NULL, dstFilter, c->param);
592.         if (!c->cascaded_context[1])
593.             return -1;
594.         return 0;
595.     }
596.     return -1;
597. }

```

sws_init_context()除了对SwsContext中的各种变量进行赋值之外，主要按照顺序完成了以下一些工作：

1. 通过sws_rgb2rgb_init()初始化RGB转RGB（或者YUV转YUV）的函数（注意不包含RGB与YUV相互转换的函数）。
2. 通过判断输入输出图像的宽高来判断图像是否需要拉伸。如果图像需要拉伸，那么unscaled变量会被标记为1。
3. 通过sws_setColorspaceDetails()初始化颜色空间。
4. 一些输入参数的检测。例如：如果没有设置图像拉伸方法的话，默认设置为SWS_BICUBIC；如果输入和输出图像的宽高小于等于0的话，也会返回错误信息。
5. 初始化Filter。这一步根据拉伸方法的不同，初始化不同的Filter。
6. 如果flags中设置了“打印信息”选项SWS_PRINT_INFO，则输出信息。
7. 如果不需要拉伸的话，调用ff_get_unscaled_swscale()将特定的像素转换函数的指针赋值给SwsContext中的swscale指针。
8. 如果需要拉伸的话，调用ff_getSwsFunc()将通用的swscale()赋值给SwsContext中的swscale指针（这个地方有点绕，但是确实是这样的）。

下面分别记录一下上述步骤的实现。

1.初始化RGB转RGB（或者YUV转YUV）的函数。注意这部分函数不包含RGB与YUV相互转换的函数。

sws_rgb2rgb_init()

sws_rgb2rgb_init()的定义位于libswscale\rgb2rgb.c，如下所示。

```

1. av_cold void sws_rgb2rgb_init(void){
2.     rgb2rgb_init_c();
3.     if (ARCH_X86)
4.         rgb2rgb_init_x86();
5. }

```

从sws_rgb2rgb_init()代码中可以看出，有两个初始化函数：rgb2rgb_init_c()是初始化C语言版本的RGB互转（或者YUV互转）的函数，rgb2rgb_init_x86()则是初始化x86汇编版本的RGB互转的函数。

PS：在libswscale中有一点需要注意：很多的函数名称中包含类似“_c”这样的字符串，代表了该函数是C语言写的。与之对应的还有其它标记，比如“_mmx”，“sse2”等。

rgb2rgb_init_c()

首先来看一下C语言版本的RGB互转函数的初始化函数rgb2rgb_init_c()，定义位于libswscale\rgb2rgb_template.c，如下所示。

```
[cpp]
1. static av_cold void rgb2rgb_init_c(void)
2. {
3.     rgb15to16      = rgb15to16_c;
4.     rgb15to16gr24  = rgb15to16gr24_c;
5.     rgb15to32      = rgb15to32_c;
6.     rgb16to16gr24  = rgb16to16gr24_c;
7.     rgb16to32      = rgb16to32_c;
8.     rgb16to15      = rgb16to15_c;
9.     rgb24to16gr16  = rgb24to16gr16_c;
10.    rgb24to16gr15   = rgb24to16gr15_c;
11.    rgb24to16gr32   = rgb24to16gr32_c;
12.    rgb32to16       = rgb32to16_c;
13.    rgb32to15       = rgb32to15_c;
14.    rgb32to16gr24   = rgb32to16gr24_c;
15.    rgb24to15       = rgb24to15_c;
16.    rgb24to16       = rgb24to16_c;
17.    rgb24to16gr24   = rgb24to16gr24_c;
18.    shuffle_bytes_2103 = shuffle_bytes_2103_c;
19.    rgb32to16gr16   = rgb32to16gr16_c;
20.    rgb32to16gr15   = rgb32to16gr15_c;
21.    yv12toyuy2      = yv12toyuy2_c;
22.    yv12toyuyvy     = yv12toyuyvy_c;
23.    yuv422ptoyuy2   = yuv422ptoyuy2_c;
24.    yuv422ptoyuyvy  = yuv422ptoyuyvy_c;
25.    yuy2toyv12      = yuy2toyv12_c;
26.    planar2x        = planar2x_c;
27.    ff_rgb24toyv12  = ff_rgb24toyv12_c;
28.    interleavedBytes = interleavedBytes_c;
29.    deinterleaveBytes = deinterleaveBytes_c;
30.    vu9_to_vu12     = vu9_to_vu12_c;
31.    yvu9_to_yuy2    = yvu9_to_yuy2_c;
32.
33.    uyvytoyuv420     = uyvytoyuv420_c;
34.    uyvytoyuv422     = uyvytoyuv422_c;
35.    yuyvtoyuv420     = yuyvtoyuv420_c;
36.    yuyvtoyuv422     = yuyvtoyuv422_c;
37. }
```

可以看出rgb2rgb_init_c()执行后，会把C语言版本的图像格式转换函数赋值给系统的函数指针。

下面我们选择几个函数看一下这些转换函数的定义。

rgb24to16gr24_c()

rgb24to16gr24_c()完成了RGB24向BGR24格式的转换。函数的定义如下所示。从代码中可以看出，该函数实现了“R”与“B”之间位置的对调，从而完成了这两种格式之间的转换。

```
[cpp]
1. static inline void rgb24to16gr24_c(const uint8_t *src, uint8_t *dst, int src_size)
2. {
3.     unsigned i;
4.
5.     for (i = 0; i < src_size; i += 3) {
6.         register uint8_t x = src[i + 2];
7.         dst[i + 1]         = src[i + 1];
8.         dst[i + 2]         = src[i + 0];
9.         dst[i + 0]         = x;
10.    }
11. }
```



rgb24to16_c()

rgb24to16_c()完成了RGB24向RGB16像素格式的转换。函数的定义如下所示。



```
[cpp]
1. static inline void rgb24to16_c(const uint8_t *src, uint8_t *dst, int src_size)
2. {
3.     uint16_t *d      = (uint16_t *)dst;
4.     const uint8_t *s  = src;
5.     const uint8_t *end = s + src_size;
6.
7.     while (s < end) {
8.         const int r = *s++;
9.         const int g = *s++;
10.        const int b = *s++;
11.        *d++         = (b >> 3) | ((g & 0xFC) << 3) | ((r & 0xF8) << 8);
12.    }
13. }
```


yuyvtoyuv422_c()



yuyvtoyuv422_c()完成了YUYV向YUV422像素格式的转换。函数的定义如下所示。

```
[cpp]    
1. static void yuyvtoyuv422_c(uint8_t *ydst, uint8_t *udst, uint8_t *vdst,  
2.                          const uint8_t *src, int width, int height,  
3.                          int lumStride, int chromStride, int srcStride)  
4. {  
5.     int y;  
6.     const int chromWidth = FF_CEIL_RSHIFT(width, 1);  
7.  
8.     for (y = 0; y < height; y++) {  
9.         extract_even_c(src, ydst, width);  
10.        extract_odd2_c(src, udst, vdst, chromWidth);  
11.  
12.        src += srcStride;  
13.        ydst += lumStride;  
14.        udst += chromStride;  
15.        vdst += chromStride;  
16.    }  
17. }
```

该函数将YUYV像素数据分离成为Y, U, V三个分量的像素数据。其中extract_even_c()用于获取一行像素中序数为偶数的像素，对应提取了YUYV像素格式中的“Y”。extract_odd2_c()用于获取一行像素中序数为奇数的像素，并且把这些像素值再次按照奇偶的不同，存储于两个数组中。对应提取了YUYV像素格式中的“U”和“V”。extract_even_c()定义如下所示。

```
[cpp]    
1. static void extract_even_c(const uint8_t *src, uint8_t *dst, int count)  
2. {  
3.     dst += count;  
4.     src += count * 2;  
5.     count = -count;  
6.     while (count < 0) {  
7.         dst[count] = src[2 * count];  
8.         count++;  
9.     }  
10. }
```

extract_odd2_c()定义如下所示。

```
[cpp]    
1. static void extract_even2_c(const uint8_t *src, uint8_t *dst0, uint8_t *dst1,  
2.                          int count)  
3. {  
4.     dst0 += count;  
5.     dst1 += count;  
6.     src += count * 4;  
7.     count = -count;  
8.     while (count < 0) {  
9.         dst0[count] = src[4 * count + 0];  
10.        dst1[count] = src[4 * count + 2];  
11.        count++;  
12.    }  
13. }
```

rgb2rgb_init_x86()

rgb2rgb_init_x86()用于初始化基于X86汇编语言的RGB互转的代码。由于对汇编不是很熟，不再作详细分析，出于和rgb2rgb_init_c()相对比的目的，列出它的代码。它的代码位于libswscale\x86\rgb2rgb.c，如下所示。

PS：所有和汇编有关的代码都位于libswscale目录的x86子目录下。

```
[cpp]    
1. av_cold void rgb2rgb_init_x86(void)  
2. {  
3.     #if HAVE_INLINE_ASM  
4.         int cpu_flags = av_get_cpu_flags();  
5.  
6.         if (INLINE_MMX(cpu_flags))  
7.             rgb2rgb_init_mmx();  
8.         if (INLINE_AMD3DNOW(cpu_flags))  
9.             rgb2rgb_init_3dnow();  
10.        if (INLINE_MMXEXT(cpu_flags))  
11.            rgb2rgb_init_mmxext();  
12.        if (INLINE_SSE2(cpu_flags))  
13.            rgb2rgb_init_sse2();  
14.        if (INLINE_AVX(cpu_flags))  
15.            rgb2rgb_init_avx();  
16.    #endif /* HAVE_INLINE_ASM */  
17. }
```

可以看出，rgb2rgb_init_x86()首先调用了av_get_cpu_flags()获取CPU支持的特性，根据特性调用rgb2rgb_init_mmx()，rgb2rgb_init_3dnow()，rgb2rgb_init_mmxext()，rgb2rgb_init_sse2()，rgb2rgb_init_avx()等函数。

2.判断图像是否需要拉伸。

这一步主要通过比较输入图像和输出图像的宽高实现。系统使用一个unscaled变量记录图像是否需要拉伸，如下所示。

```
1. unscaled = (srcW == dstW && srcH == dstH);
```

3.初始化颜色空间。

初始化颜色空间通过函数sws_setColorspaceDetails()完成。sws_setColorspaceDetails()是FFmpeg的一个API函数，它的声明如下所示：

```
1. /**
2.  * @param dstRange flag indicating the while-black range of the output (1=jpeg / 0=mpeg)
3.  * @param srcRange flag indicating the while-black range of the input (1=jpeg / 0=mpeg)
4.  * @param table the yuv2rgb coefficients describing the output yuv space, normally ff_yuv2rgb_coefs[x]
5.  * @param inv_table the yuv2rgb coefficients describing the input yuv space, normally ff_yuv2rgb_coefs[x]
6.  * @param brightness 16.16 fixed point brightness correction
7.  * @param contrast 16.16 fixed point contrast correction
8.  * @param saturation 16.16 fixed point saturation correction
9.  * @return -1 if not supported
10. */
11. int sws_setColorspaceDetails(struct SwsContext *c, const int inv_table[4],
12.                             int srcRange, const int table[4], int dstRange,
13.                             int brightness, int contrast, int saturation);
```

简单解释一下几个参数的含义：

c：需要设定的SwsContext。

inv_table：描述输出YUV颜色空间的参数表。

srcRange：输入图像的取值范围（“1”代表JPEG标准，取值范围是0-255；“0”代表MPEG标准，取值范围是16-235）。

table：描述输入YUV颜色空间的参数表。

dstRange：输出图像的取值范围。

brightness：未研究。

contrast：未研究。

saturation：未研究。

如果返回-1代表设置不成功。

其中描述颜色空间的参数表可以通过sws_getCoefficients()获取。该函数在后文中再详细记录。

sws_setColorspaceDetails()的定义位于libswscale\utils.c，如下所示。

```

1.  int sws_setColorspaceDetails(struct SwsContext *c, const int inv_table[4],
2.                             int srcRange, const int table[4], int dstRange,
3.                             int brightness, int contrast, int saturation)
4.  {
5.      const AVPixFmtDescriptor *desc_dst;
6.      const AVPixFmtDescriptor *desc_src;
7.      int need_reinit = 0;
8.      memmove(c->srcColorspaceTable, inv_table, sizeof(int) * 4);
9.      memmove(c->dstColorspaceTable, table, sizeof(int) * 4);
10.
11.      handle_formats(c);
12.      desc_dst = av_pix_fmt_desc_get(c->dstFormat);
13.      desc_src = av_pix_fmt_desc_get(c->srcFormat);
14.
15.      if(!isYUV(c->dstFormat) && !isGray(c->dstFormat))
16.          dstRange = 0;
17.      if(!isYUV(c->srcFormat) && !isGray(c->srcFormat))
18.          srcRange = 0;
19.
20.      c->brightness = brightness;
21.      c->contrast = contrast;
22.      c->saturation = saturation;
23.      if (c->srcRange != srcRange || c->dstRange != dstRange)
24.          need_reinit = 1;
25.      c->srcRange = srcRange;
26.      c->dstRange = dstRange;
27.
28.      //The srcBpc check is possibly wrong but we seem to lack a definitive reference to test this
29.      //and what we have in ticket 2939 looks better with this check
30.      if (need_reinit && (c->srcBpc == 8 || !isYUV(c->srcFormat)))
31.          ff_sws_init_range_convert(c);
32.
33.      if ((isYUV(c->dstFormat) || isGray(c->dstFormat)) && (isYUV(c->srcFormat) || isGray(c->srcFormat)))
34.          return -1;
35.
36.      c->dstFormatBpp = av_get_bits_per_pixel(desc_dst);
37.      c->srcFormatBpp = av_get_bits_per_pixel(desc_src);
38.
39.      if (!isYUV(c->dstFormat) && !isGray(c->dstFormat)) {
40.          ff_yuv2rgb_c_init_tables(c, inv_table, srcRange, brightness,
41.                                  contrast, saturation);
42.          // FIXME factorize
43.
44.          if (ARCH_PPC)
45.              ff_yuv2rgb_init_tables_ppc(c, inv_table, brightness,
46.                                          contrast, saturation);
47.      }
48.
49.      fill_rgb2yuv_table(c, table, dstRange);
50.
51.      return 0;
52.  }

```

从sws_setColorspaceDetails()定义中可以看出，该函数将输入的参数分别赋值给了相应的变量，并且在最后调用了函数fill_rgb2yuv_table()。fill_rgb2yuv_table()函数还没有弄懂，暂时不记录。

sws_getCoefficients()

sws_getCoefficients()用于获取描述颜色空间的参数表。它的声明如下。

```

1.  /**
2.   * Return a pointer to yuv<->rgb coefficients for the given colorspace
3.   * suitable for sws_setColorspaceDetails().
4.   *
5.   * @param colorspace One of the SWS_CS_* macros. If invalid,
6.   * SWS_CS_DEFAULT is used.
7.   */
8.  const int *sws_getCoefficients(int colorspace);

```

其中colorspace可以取值如下变量。默认的取值SWS_CS_DEFAULT等同于SWS_CS_ITU601或者SWS_CS_SMPTE170M。

```



1.  #define SWS_CS_ITU709      1
2.  #define SWS_CS_FCC        4
3.  #define SWS_CS_ITU601     5
4.  #define SWS_CS_ITU624     5
5.  #define SWS_CS_SMPTE170M  5
6.  #define SWS_CS_SMPTE240M  7
7.  #define SWS_CS_DEFAULT    5

```

下面看一下sws_getCoefficients()的定义，位于libswscale/yuv2rgb.c，如下所示。



```
[cpp]    
1.  const int *sws_getCoefficients(int colorspace)  
2.  {  
3.      if (colorspace > 7 || colorspace < 0)  
4.          colorspace = SWS_CS_DEFAULT;  
5.      return ff_yuv2rgb_coefs[colorSpace];  
6.  }
```

可以看出它返回了一个名称为ff_yuv2rgb_coefs的数组中的一个元素，该数组的定义如下所示。

```
[cpp]    
1.  const int32_t ff_yuv2rgb_coefs[8][4] = {  
2.      { 117504, 138453, 13954, 34903 }, /* no sequence_display_extension */  
3.      { 117504, 138453, 13954, 34903 }, /* ITU-R Rec. 709 (1990) */  
4.      { 104597, 132201, 25675, 53279 }, /* unspecified */  
5.      { 104597, 132201, 25675, 53279 }, /* reserved */  
6.      { 104448, 132798, 24759, 53109 }, /* FCC */  
7.      { 104597, 132201, 25675, 53279 }, /* ITU-R Rec. 624-4 System B, G */  
8.      { 104597, 132201, 25675, 53279 }, /* SMPTE 170M */  
9.      { 117579, 136230, 16907, 35559 }, /* SMPTE 240M (1987) */  
10. };
```

4.一些输入参数的检测。

例如：如果没有设置图像拉伸方法的话，默认设置为SWS_BICUBIC；如果输入和输出图像的宽高小于等于0的话，也会返回错误信息。有关这方面的代码比较多，简单举个例子。

```
[cpp]    
1.  i = flags & (SWS_POINT |  
2.      SWS_AREA |  
3.      SWS_BILINEAR |  
4.      SWS_FAST_BILINEAR |  
5.      SWS_BICUBIC |  
6.      SWS_X |  
7.      SWS_GAUSS |  
8.      SWS_LANCZOS |  
9.      SWS_SINC |  
10.     SWS_SPLINE |  
11.     SWS_BICUBLIN);  
12.  
13. /* provide a default scaler if not set by caller */  
14. if (!i) {  
15.     if (dstW < srcW && dstH < srcH)  
16.         flags |= SWS_BICUBIC;  
17.     else if (dstW > srcW && dstH > srcH)  
18.         flags |= SWS_BICUBIC;  
19.     else  
20.         flags |= SWS_BICUBIC;  
21.     c->flags = flags;  
22. } else if (i & (i - 1)) {  
23.     av_log(c, AV_LOG_ERROR,  
24.         "Exactly one scaler algorithm must be chosen, got %X\n", i);  
25.     return AVERROR(EINVAL);  
26. }  
27. /* sanity check */  
28. if (srcW < 1 || srcH < 1 || dstW < 1 || dstH < 1) {  
29.     /* FIXME check if these are enough and try to lower them after  
30.      * fixing the relevant parts of the code */  
31.     av_log(c, AV_LOG_ERROR, "%dx%d -> %dx%d is invalid scaling dimension\n",  
32.         srcW, srcH, dstW, dstH);  
33.     return AVERROR(EINVAL);  
34. }
```

5.初始化Filter。这一步根据拉伸方法的不同，初始化不同的Filter。

这一部分的工作在函数initFilter()中完成，暂时不详细分析。

6.如果flags中设置了“打印信息”选项SWS_PRINT_INFO，则输出信息。

SwsContext初始化的时候，可以给flags设置SWS_PRINT_INFO标记。这样SwsContext初始化完成的时候就可以打印出一些配置信息。与打印相关的代码如下所示。

```

1.  if (flags & SWS_PRINT_INFO) {
2.      const char *scaler = NULL, *cpucaps;
3.
4.      for (i = 0; i < FF_ARRAY_ELEMS(scale_algorithms); i++) {
5.          if (flags & scale_algorithms[i].flag) {
6.              scaler = scale_algorithms[i].description;
7.              break;
8.          }
9.      }
10.     if (!scaler)
11.         scaler = "ehh flags invalid?!";
12.     av_log(c, AV_LOG_INFO, "%s scaler, from %s to %s\n",
13.         scaler,
14.         av_get_pix_fmt_name(srcFormat),
15. #ifdef DITHER1XBPP
16.         dstFormat == AV_PIX_FMT_BGR555 || dstFormat == AV_PIX_FMT_BGR565 ||
17.         dstFormat == AV_PIX_FMT_RGB444BE || dstFormat == AV_PIX_FMT_RGB444LE ||
18.         dstFormat == AV_PIX_FMT_BGR444BE || dstFormat == AV_PIX_FMT_BGR444LE ?
19.             "dithered " : "",
20. #else
21.         "",
22. #endif
23.         av_get_pix_fmt_name(dstFormat));
24.
25.     if (INLINE_MMEXT(cpu_flags))
26.         cpucaps = "MMEXT";
27.     else if (INLINE_AMD3DNOW(cpu_flags))
28.         cpucaps = "3DNOW";
29.     else if (INLINE_MMX(cpu_flags))
30.         cpucaps = "MMX";
31.     else if (PPC_ALTIVEC(cpu_flags))
32.         cpucaps = "Altivec";
33.     else
34.         cpucaps = "C";
35.
36.     av_log(c, AV_LOG_INFO, "using %s\n", cpucaps);
37.
38.     av_log(c, AV_LOG_VERBOSE, "%dx%d -> %dx%d\n", srcW, srcH, dstW, dstH);
39.     av_log(c, AV_LOG_DEBUG,
40.         "lum srcW=%d srcH=%d dstW=%d dstH=%d xInc=%d yInc=%d\n",
41.         c->srcW, c->srcH, c->dstW, c->dstH, c->lumXInc, c->lumYInc);
42.     av_log(c, AV_LOG_DEBUG,
43.         "chr srcW=%d srcH=%d dstW=%d dstH=%d xInc=%d yInc=%d\n",
44.         c->chrSrcW, c->chrSrcH, c->chrDstW, c->chrDstH,
45.         c->chrXInc, c->chrYInc);
46. }

```

7. 如果不需要拉伸的话，就会调用ff_get_unscaled_swscale()将特定的像素转换函数的指针赋值给SwsContext中的swscale指针。

ff_get_unscaled_swscale()

ff_get_unscaled_swscale()的定义如下所示。该函数根据输入图像像素格式和输出图像像素格式，选择不同的像素格式转换函数。

```

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_YUV422P)
149.             c->swscale = yuv422pToYuy2Wrapper;
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 yuy2 */
157.         if (srcFormat == AV_PIX_FMT_YUV420P || srcFormat == AV_PIX_FMT_YUVA420P) {
158.             if (dstFormat == AV_PIX_FMT_YUV422P)
159.                 c->swscale = planarToYuy2Wrapper;
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->chrDstSubSample == c->chrSrcSubSample &&
185.          c->chrDstVSubSample == 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ff_get_unscaled_swscale()源代码中可以看出，赋值给SwsContext的swscale指针的函数名称大多数为XXXWrapper()。实际上这些函数封装了一些基本的像素格式转换函数。例如yuyvToYuv422Wrapper()的定义如下所示。

```
[cpp]
1. static int yuyvToYuv422Wrapper(SwsContext *c, const uint8_t *src[],
2.                               int srcStride[], int srcSliceY, int srcSliceH,
3.                               uint8_t *dstParam[], int dstStride[])
4. {
5.     uint8_t *ydst = dstParam[0] + dstStride[0] * srcSliceY;
6.     uint8_t *udst = dstParam[1] + dstStride[1] * srcSliceY;
7.     uint8_t *vdst = dstParam[2] + dstStride[2] * srcSliceY;
8.
9.     yuyvtoyuv422(ydst, udst, vdst, src[0], c->srcW, srcSliceH, dstStride[0],
10.                  dstStride[1], srcStride[0]);
11.
12.     return srcSliceH;
13. }
```

从yuyvToYuv422Wrapper()的定义中可以看出，它调用了yuyvtoyuv422()。而yuyvtoyuv422()则是rgb2rgb.c中的一个函数，用于将YUVU转换为YUV422（该函数在前文中已经记录）。

8.如果需要拉伸的话，就会调用ff_getSwsFunc()将通用的swscale()赋值给SwsContext中的swscale指针，然后返回。

上一步骤（图像不用缩放）实际上是一种不太常见的情况，更多的情况下会执行本步骤。这个时候就会调用ff_getSwsFunc()获取图像的缩放函数。

ff_getSwsFunc()

ff_getSwsFunc()用于获取通用的swscale()函数。该函数的定义如下。

```
[cpp]
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. }
```

从源代码中可以看出ff_getSwsFunc()调用了函数sws_init_swscale()。如果系统支持X86汇编的话，还会调用ff_sws_init_swscale_x86()。

sws_init_swscale()

sws_init_swscale()的定义位于libswscale\swscale.c，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. }
```


从函数中可以看出, `sws_init_swscale()`主要调用了3个函数: `ff_sws_init_output_funcs()`, `ff_sws_init_input_funcs()`, `ff_sws_init_range_convert()`。其中, `ff_sws_init_output_funcs()`用于初始化输出的函数, `ff_sws_init_input_funcs()`用于初始化输入的函数, `ff_sws_init_range_convert()`用于初始化像素值范围转换的函数。

ff_sws_init_output_funcs()

ff_sws_init_output_funcs()用于初始化“输出函数”。“输出函数”在libswscale中的作用就是将处理后的一行像素数据输出出来。ff_sws_init_output_funcs()的定义位于libswscale/output.c，如下所示。

```

1. av_cold void ff_sws_init_output_funcs(SwsContext *c,
2.                                     yuv2planar1_fn *yuv2plane1,
3.                                     yuv2planarX_fn *yuv2planeX,
4.                                     yuv2interleavedX_fn *yuv2nv12cX,
5.                                     yuv2packed1_fn *yuv2packed1,
6.                                     yuv2packed2_fn *yuv2packed2,
7.                                     yuv2packedX_fn *yuv2packedX,
8.                                     yuv2anyX_fn *yuv2anyX)
9. {
10.     enum AVPixelFormat dstFormat = c->dstFormat;
11.     const AVPixFmtDescriptor *desc = av_pix_fmt_desc_get(dstFormat);
12.
13.     if (is16BPS(dstFormat)) {
14.         *yuv2planeX = isBE(dstFormat) ? yuv2planeX_16BE_c : yuv2planeX_16LE_c;
15.         *yuv2plane1 = isBE(dstFormat) ? yuv2plane1_16BE_c : yuv2plane1_16LE_c;
16.     } else if (is9_OR_10BPS(dstFormat)) {
17.         if (desc->comp[0].depth_minus1 == 8) {
18.             *yuv2planeX = isBE(dstFormat) ? yuv2planeX_9BE_c : yuv2planeX_9LE_c;
19.             *yuv2plane1 = isBE(dstFormat) ? yuv2plane1_9BE_c : yuv2plane1_9LE_c;
20.         } else if (desc->comp[0].depth_minus1 == 9) {
21.             *yuv2planeX = isBE(dstFormat) ? yuv2planeX_10BE_c : yuv2planeX_10LE_c;
22.             *yuv2plane1 = isBE(dstFormat) ? yuv2plane1_10BE_c : yuv2plane1_10LE_c;
23.         } else if (desc->comp[0].depth_minus1 == 11) {
24.             *yuv2planeX = isBE(dstFormat) ? yuv2planeX_12BE_c : yuv2planeX_12LE_c;
25.             *yuv2plane1 = isBE(dstFormat) ? yuv2plane1_12BE_c : yuv2plane1_12LE_c;
26.         } else if (desc->comp[0].depth_minus1 == 13) {
27.             *yuv2planeX = isBE(dstFormat) ? yuv2planeX_14BE_c : yuv2planeX_14LE_c;
28.             *yuv2plane1 = isBE(dstFormat) ? yuv2plane1_14BE_c : yuv2plane1_14LE_c;
29.         } else
30.             av_assert0(0);
31.     } else {
32.         *yuv2plane1 = yuv2plane1_8_c;
33.         *yuv2planeX = yuv2planeX_8_c;
34.         if (dstFormat == AV_PIX_FMT_NV12 || dstFormat == AV_PIX_FMT_NV21)
35.             *yuv2nv12cX = yuv2nv12cX_c;
36.     }
37.
38.     if (c->flags & SWS_FULL_CHR_H_INT) {
39.         switch (dstFormat) {
40.             case AV_PIX_FMT_RGBA:
41. #if CONFIG_SMALL
42.                 *yuv2packedX = yuv2rgba32_full_X_c;
43.                 *yuv2packed2 = yuv2rgba32_full_2_c;
44.                 *yuv2packed1 = yuv2rgba32_full_1_c;
45. #else
46. #if CONFIG_SWSCALE_ALPHA
47.                 if (c->alpPixBuf) {
48.                     *yuv2packedX = yuv2rgba32_full_X_c;
49.                     *yuv2packed2 = yuv2rgba32_full_2_c;
50.                     *yuv2packed1 = yuv2rgba32_full_1_c;
51.                 } else
52. #endif /* CONFIG_SWSCALE_ALPHA */
53.                 {
54.                     *yuv2packedX = yuv2rgb32_full_X_c;
55.                     *yuv2packed2 = yuv2rgb32_full_2_c;
56.                     *yuv2packed1 = yuv2rgb32_full_1_c;
57.                 }
58. #endif /* !CONFIG_SMALL */
59.                 break;
60.             case AV_PIX_FMT_ARGB:
61. #if CONFIG_SMALL
62.                 *yuv2packedX = yuv2argb32_full_X_c;
63.                 *yuv2packed2 = yuv2argb32_full_2_c;
64.                 *yuv2packed1 = yuv2argb32_full_1_c;
65. #else
66. #if CONFIG_SWSCALE_ALPHA
67.                 if (c->alpPixBuf) {
68.                     *yuv2packedX = yuv2argb32_full_X_c;
69.                     *yuv2packed2 = yuv2argb32_full_2_c;
70.                     *yuv2packed1 = yuv2argb32_full_1_c;
71.                 } else
72. #endif /* CONFIG_SWSCALE_ALPHA */
73.                 {
74.                     *yuv2packedX = yuv2xrgb32_full_X_c;
75.                     *yuv2packed2 = yuv2xrgb32_full_2_c;
76.                     *yuv2packed1 = yuv2xrgb32_full_1_c;
77.                 }
78. #endif
79.         }
80.     }
81. }

```

```

70.         *yuv2packed1 = yuv2xrgb32_full_1_c;
71.     }
72. #endif /* !CONFIG_SMALL */
73.     break;
74.     case AV_PIX_FMT_BGRA:
75. #if CONFIG_SMALL
76.         *yuv2packedX = yuv2bgra32_full_X_c;
77.         *yuv2packed2 = yuv2bgra32_full_2_c;
78.         *yuv2packed1 = yuv2bgra32_full_1_c;
79.     #else
80. #if CONFIG_SWSCALE_ALPHA
81.         if (c->alpPixBuf) {
82.             *yuv2packedX = yuv2bgra32_full_X_c;
83.             *yuv2packed2 = yuv2bgra32_full_2_c;
84.             *yuv2packed1 = yuv2bgra32_full_1_c;
85.         } else
86. #endif /* CONFIG_SWSCALE_ALPHA */
87.         {
88.             *yuv2packedX = yuv2bgrx32_full_X_c;
89.             *yuv2packed2 = yuv2bgrx32_full_2_c;
90.             *yuv2packed1 = yuv2bgrx32_full_1_c;
91.         }
92. #endif /* !CONFIG_SMALL */
93.     break;
94.     case AV_PIX_FMT_ABGR:
95. #if CONFIG_SMALL
96.         *yuv2packedX = yuv2abgr32_full_X_c;
97.         *yuv2packed2 = yuv2abgr32_full_2_c;
98.         *yuv2packed1 = yuv2abgr32_full_1_c;
99.     #else
100. #if CONFIG_SWSCALE_ALPHA
101.         if (c->alpPixBuf) {
102.             *yuv2packedX = yuv2abgr32_full_X_c;
103.             *yuv2packed2 = yuv2abgr32_full_2_c;
104.             *yuv2packed1 = yuv2abgr32_full_1_c;
105.         } else
106. #endif /* CONFIG_SWSCALE_ALPHA */
107.         {
108.             *yuv2packedX = yuv2xbgr32_full_X_c;
109.             *yuv2packed2 = yuv2xbgr32_full_2_c;
110.             *yuv2packed1 = yuv2xbgr32_full_1_c;
111.         }
112. #endif /* !CONFIG_SMALL */
113.     break;
114.     case AV_PIX_FMT_RGB24:
115.         *yuv2packedX = yuv2rgb24_full_X_c;
116.         *yuv2packed2 = yuv2rgb24_full_2_c;
117.         *yuv2packed1 = yuv2rgb24_full_1_c;
118.     break;
119.     case AV_PIX_FMT_BGR24:
120.         *yuv2packedX = yuv2bgr24_full_X_c;
121.         *yuv2packed2 = yuv2bgr24_full_2_c;
122.         *yuv2packed1 = yuv2bgr24_full_1_c;
123.     break;
124.     case AV_PIX_FMT_BGR4_BYTE:
125.         *yuv2packedX = yuv2bgr4_byte_full_X_c;
126.         *yuv2packed2 = yuv2bgr4_byte_full_2_c;
127.         *yuv2packed1 = yuv2bgr4_byte_full_1_c;
128.     break;
129.     case AV_PIX_FMT_RGB4_BYTE:
130.         *yuv2packedX = yuv2rgb4_byte_full_X_c;
131.         *yuv2packed2 = yuv2rgb4_byte_full_2_c;
132.         *yuv2packed1 = yuv2rgb4_byte_full_1_c;
133.     break;
134.     case AV_PIX_FMT_BGR8:
135.         *yuv2packedX = yuv2bgr8_full_X_c;
136.         *yuv2packed2 = yuv2bgr8_full_2_c;
137.         *yuv2packed1 = yuv2bgr8_full_1_c;
138.     break;
139.     case AV_PIX_FMT_RGB8:
140.         *yuv2packedX = yuv2rgb8_full_X_c;
141.         *yuv2packed2 = yuv2rgb8_full_2_c;
142.         *yuv2packed1 = yuv2rgb8_full_1_c;
143.     break;
144.     case AV_PIX_FMT_GBRP:
145.     case AV_PIX_FMT_GBRP9BE:
146.     case AV_PIX_FMT_GBRP9LE:
147.     case AV_PIX_FMT_GBRP10BE:
148.     case AV_PIX_FMT_GBRP10LE:
149.     case AV_PIX_FMT_GBRP12BE:
150.     case AV_PIX_FMT_GBRP12LE:
151.     case AV_PIX_FMT_GBRP14BE:
152.     case AV_PIX_FMT_GBRP14LE:
153.     case AV_PIX_FMT_GBRP16BE:
154.     case AV_PIX_FMT_GBRP16LE:
155.     case AV_PIX_FMT_GBRAP:
156.         *yuv2anyX = yuv2gbrp_full_X_c;
157.         break;
158.     }
159.     if (!*yuv2packedX && !*yuv2anyX)
160.         goto YUV_PACKED;
161. } else {

```

```

168.     YUV_PACKED:
169.         switch (dstFormat) {
170.             case AV_PIX_FMT_RGBA64LE:
171. #if CONFIG_SWSCALE_ALPHA
172.                 if (c->alpPixBuf) {
173.                     *yuv2packed1 = yuv2rgba64le_1_c;
174.                     *yuv2packed2 = yuv2rgba64le_2_c;
175.                     *yuv2packedX = yuv2rgba64le_X_c;
176.                 } else
177. #endif /* CONFIG_SWSCALE_ALPHA */
178.                 {
179.                     *yuv2packed1 = yuv2rgbx64le_1_c;
180.                     *yuv2packed2 = yuv2rgbx64le_2_c;
181.                     *yuv2packedX = yuv2rgbx64le_X_c;
182.                 }
183.                 break;
184.             case AV_PIX_FMT_RGBA64BE:
185. #if CONFIG_SWSCALE_ALPHA
186.                 if (c->alpPixBuf) {
187.                     *yuv2packed1 = yuv2rgba64be_1_c;
188.                     *yuv2packed2 = yuv2rgba64be_2_c;
189.                     *yuv2packedX = yuv2rgba64be_X_c;
190.                 } else
191. #endif /* CONFIG_SWSCALE_ALPHA */
192.                 {
193.                     *yuv2packed1 = yuv2rgbx64be_1_c;
194.                     *yuv2packed2 = yuv2rgbx64be_2_c;
195.                     *yuv2packedX = yuv2rgbx64be_X_c;
196.                 }
197.                 break;
198.             case AV_PIX_FMT_BGRA64LE:
199. #if CONFIG_SWSCALE_ALPHA
200.                 if (c->alpPixBuf) {
201.                     *yuv2packed1 = yuv2bgra64le_1_c;
202.                     *yuv2packed2 = yuv2bgra64le_2_c;
203.                     *yuv2packedX = yuv2bgra64le_X_c;
204.                 } else
205. #endif /* CONFIG_SWSCALE_ALPHA */
206.                 {
207.                     *yuv2packed1 = yuv2bgrx64le_1_c;
208.                     *yuv2packed2 = yuv2bgrx64le_2_c;
209.                     *yuv2packedX = yuv2bgrx64le_X_c;
210.                 }
211.                 break;
212.             case AV_PIX_FMT_BGRA64BE:
213. #if CONFIG_SWSCALE_ALPHA
214.                 if (c->alpPixBuf) {
215.                     *yuv2packed1 = yuv2bgra64be_1_c;
216.                     *yuv2packed2 = yuv2bgra64be_2_c;
217.                     *yuv2packedX = yuv2bgra64be_X_c;
218.                 } else
219. #endif /* CONFIG_SWSCALE_ALPHA */
220.                 {
221.                     *yuv2packed1 = yuv2bgrx64be_1_c;
222.                     *yuv2packed2 = yuv2bgrx64be_2_c;
223.                     *yuv2packedX = yuv2bgrx64be_X_c;
224.                 }
225.                 break;
226.             case AV_PIX_FMT_RGB48LE:
227.                 *yuv2packed1 = yuv2rgb48le_1_c;
228.                 *yuv2packed2 = yuv2rgb48le_2_c;
229.                 *yuv2packedX = yuv2rgb48le_X_c;
230.                 break;
231.             case AV_PIX_FMT_RGB48BE:
232.                 *yuv2packed1 = yuv2rgb48be_1_c;
233.                 *yuv2packed2 = yuv2rgb48be_2_c;
234.                 *yuv2packedX = yuv2rgb48be_X_c;
235.                 break;
236.             case AV_PIX_FMT_BGR48LE:
237.                 *yuv2packed1 = yuv2bgr48le_1_c;
238.                 *yuv2packed2 = yuv2bgr48le_2_c;
239.                 *yuv2packedX = yuv2bgr48le_X_c;
240.                 break;
241.             case AV_PIX_FMT_BGR48BE:
242.                 *yuv2packed1 = yuv2bgr48be_1_c;
243.                 *yuv2packed2 = yuv2bgr48be_2_c;
244.                 *yuv2packedX = yuv2bgr48be_X_c;
245.                 break;
246.             case AV_PIX_FMT_RGB32:
247.             case AV_PIX_FMT_BGR32:
248. #if CONFIG_SMALL
249.                 *yuv2packed1 = yuv2rgb32_1_c;
250.                 *yuv2packed2 = yuv2rgb32_2_c;
251.                 *yuv2packedX = yuv2rgb32_X_c;
252.             #else
253. #if CONFIG_SWSCALE_ALPHA
254.                 if (c->alpPixBuf) {
255.                     *yuv2packed1 = yuv2rgba32_1_c;
256.                     *yuv2packed2 = yuv2rgba32_2_c;
257.                     *yuv2packedX = yuv2rgba32_X_c;
258.                 } else

```

```

259. #endif /* CONFIG_SWSCALE_ALPHA */
260. {
261.     *yuv2packed1 = yuv2rgb32_1_c;
262.     *yuv2packed2 = yuv2rgb32_2_c;
263.     *yuv2packedX = yuv2rgb32_X_c;
264. }
265. #endif /* !CONFIG_SMALL */
266.     break;
267.     case AV_PIX_FMT_RGB32_1:
268.     case AV_PIX_FMT_BGR32_1:
269. #if CONFIG_SMALL
270.         *yuv2packed1 = yuv2rgb32_1_1_c;
271.         *yuv2packed2 = yuv2rgb32_1_2_c;
272.         *yuv2packedX = yuv2rgb32_1_X_c;
273. #else
274. #if CONFIG_SWSCALE_ALPHA
275.         if (c->alpPixBuf) {
276.             *yuv2packed1 = yuv2rgba32_1_1_c;
277.             *yuv2packed2 = yuv2rgba32_1_2_c;
278.             *yuv2packedX = yuv2rgba32_1_X_c;
279.         } else
280. #endif /* CONFIG_SWSCALE_ALPHA */
281.         {
282.             *yuv2packed1 = yuv2rgb32_1_1_c;
283.             *yuv2packed2 = yuv2rgb32_1_2_c;
284.             *yuv2packedX = yuv2rgb32_1_X_c;
285.         }
286. #endif /* !CONFIG_SMALL */
287.     break;
288.     case AV_PIX_FMT_RGB24:
289.         *yuv2packed1 = yuv2rgb24_1_c;
290.         *yuv2packed2 = yuv2rgb24_2_c;
291.         *yuv2packedX = yuv2rgb24_X_c;
292.     break;
293.     case AV_PIX_FMT_BGR24:
294.         *yuv2packed1 = yuv2bgr24_1_c;
295.         *yuv2packed2 = yuv2bgr24_2_c;
296.         *yuv2packedX = yuv2bgr24_X_c;
297.     break;
298.     case AV_PIX_FMT_RGB565LE:
299.     case AV_PIX_FMT_RGB565BE:
300.     case AV_PIX_FMT_BGR565LE:
301.     case AV_PIX_FMT_BGR565BE:
302.         *yuv2packed1 = yuv2rgb16_1_c;
303.         *yuv2packed2 = yuv2rgb16_2_c;
304.         *yuv2packedX = yuv2rgb16_X_c;
305.     break;
306.     case AV_PIX_FMT_RGB555LE:
307.     case AV_PIX_FMT_RGB555BE:
308.     case AV_PIX_FMT_BGR555LE:
309.     case AV_PIX_FMT_BGR555BE:
310.         *yuv2packed1 = yuv2rgb15_1_c;
311.         *yuv2packed2 = yuv2rgb15_2_c;
312.         *yuv2packedX = yuv2rgb15_X_c;
313.     break;
314.     case AV_PIX_FMT_RGB444LE:
315.     case AV_PIX_FMT_RGB444BE:
316.     case AV_PIX_FMT_BGR444LE:
317.     case AV_PIX_FMT_BGR444BE:
318.         *yuv2packed1 = yuv2rgb12_1_c;
319.         *yuv2packed2 = yuv2rgb12_2_c;
320.         *yuv2packedX = yuv2rgb12_X_c;
321.     break;
322.     case AV_PIX_FMT_RGB8:
323.     case AV_PIX_FMT_BGR8:
324.         *yuv2packed1 = yuv2rgb8_1_c;
325.         *yuv2packed2 = yuv2rgb8_2_c;
326.         *yuv2packedX = yuv2rgb8_X_c;
327.     break;
328.     case AV_PIX_FMT_RGB4:
329.     case AV_PIX_FMT_BGR4:
330.         *yuv2packed1 = yuv2rgb4_1_c;
331.         *yuv2packed2 = yuv2rgb4_2_c;
332.         *yuv2packedX = yuv2rgb4_X_c;
333.     break;
334.     case AV_PIX_FMT_RGBA_BYTE:
335.     case AV_PIX_FMT_BGRA_BYTE:
336.         *yuv2packed1 = yuv2rgb4b_1_c;
337.         *yuv2packed2 = yuv2rgb4b_2_c;
338.         *yuv2packedX = yuv2rgb4b_X_c;
339.     break;
340. }
341. }
342. switch (dstFormat) {
343.     case AV_PIX_FMT_MONOWHITE:
344.         *yuv2packed1 = yuv2monowhite_1_c;
345.         *yuv2packed2 = yuv2monowhite_2_c;
346.         *yuv2packedX = yuv2monowhite_X_c;
347.     break;
348.     case AV_PIX_FMT_MONOBLACK:
349.         *yuv2packed1 = yuv2monoblack_1_c;

```

```

350.         *yuv2packed2 = yuv2monoback_2_c;
351.         *yuv2packedX = yuv2monoback_X_c;
352.         break;
353.     case AV_PIX_FMT_YUYV422:
354.         *yuv2packed1 = yuv2yuyv422_1_c;
355.         *yuv2packed2 = yuv2yuyv422_2_c;
356.         *yuv2packedX = yuv2yuyv422_X_c;
357.         break;
358.     case AV_PIX_FMT_YVYU422:
359.         *yuv2packed1 = yuv2yvvy422_1_c;
360.         *yuv2packed2 = yuv2yvvy422_2_c;
361.         *yuv2packedX = yuv2yvvy422_X_c;
362.         break;
363.     case AV_PIX_FMT_UYVY422:
364.         *yuv2packed1 = yuv2uyvy422_1_c;
365.         *yuv2packed2 = yuv2uyvy422_2_c;
366.         *yuv2packedX = yuv2uyvy422_X_c;
367.         break;
368.     }
369. }

```

ff_sws_init_output_funcs()根据输出像素格式的不同，对以下几个函数指针进行赋值：

yuv2plane1：是yuv2planar1_fn类型的函数指针。该函数用于输出一行水平拉伸后的planar格式数据。数据没有使用垂直拉伸。

yuv2planeX：是yuv2planarX_fn类型的函数指针。该函数用于输出一行水平拉伸后的planar格式数据。数据使用垂直拉伸。

yuv2packed1：是yuv2packed1_fn类型的函数指针。该函数用于输出一行水平拉伸后的packed格式数据。数据没有使用垂直拉伸。

yuv2packed2：是yuv2packed2_fn类型的函数指针。该函数用于输出一行水平拉伸后的packed格式数据。数据使用两行数据进行垂直拉伸。

yuv2packedX：是yuv2packedX_fn类型的函数指针。该函数用于输出一行水平拉伸后的packed格式数据。数据使用垂直拉伸。

yuv2nv12cX：是yuv2interleavedX_fn类型的函数指针。还没有研究该函数。

yuv2anyX：是yuv2anyX_fn类型的函数指针。还没有研究该函数。

ff_sws_init_input_funcs()

ff_sws_init_input_funcs()用于初始化“输入函数”。“输入函数”在libswscale中的作用就是任意格式的像素转换为YUV格式以供后续的处理。ff_sws_init_input_funcs()的定义位于libswscale/input.c，如下所示。

```

1.  av_cold void ff_sws_init_input_funcs(SwsContext *c)
2.  {
3.      enum AVPixelFormat srcFormat = c->srcFormat;
4.
5.      c->chrToYV12 = NULL;
6.      switch (srcFormat) {
7.          case AV_PIX_FMT_YUYV422:
8.              c->chrToYV12 = yuy2ToUV_c;
9.              break;
10.         case AV_PIX_FMT_YVYU422:
11.             c->chrToYV12 = yvy2ToUV_c;
12.             break;
13.         case AV_PIX_FMT_UYVY422:
14.             c->chrToYV12 = uyvyToUV_c;
15.             break;
16.         case AV_PIX_FMT_NV12:
17.             c->chrToYV12 = nv12ToUV_c;
18.             break;
19.         case AV_PIX_FMT_NV21:
20.             c->chrToYV12 = nv21ToUV_c;
21.             break;
22.         case AV_PIX_FMT_RGB8:
23.         case AV_PIX_FMT_BGR8:
24.         case AV_PIX_FMT_PAL8:
25.         case AV_PIX_FMT_BGR4_BYTE:
26.         case AV_PIX_FMT_RGB4_BYTE:
27.             c->chrToYV12 = palToUV_c;
28.             break;
29.         case AV_PIX_FMT_GBRP9LE:
30.             c->readChrPlanar = planar_rgb9le_to_uv;
31.             break;
32.         case AV_PIX_FMT_GBRP10LE:
33.             c->readChrPlanar = planar_rgb10le_to_uv;
34.             break;
35.         case AV_PIX_FMT_GBRP12LE:
36.             c->readChrPlanar = planar_rgb12le_to_uv;
37.             break;
38.         case AV_PIX_FMT_GBRP14LE:
39.             c->readChrPlanar = planar_rgb14le_to_uv;
40.             break;
41.         case AV_PIX_FMT_GBRAP16LE:
42.         case AV_PIX_FMT_GBRP16LE:
43.             c->readChrPlanar = planar_rgb16le_to_uv;
44.             break;
45.         case AV_PIX_FMT_GBRP9BE:
46.             c->readChrPlanar = planar_rgb9be_to_uv;
47.             break;
48.         case AV_PIX_FMT_GBRP10BE:

```

```

49.         c->readChrPlanar = planar_rgb10be_to_uv;
50.         break;
51.     case AV_PIX_FMT_GBRP12BE:
52.         c->readChrPlanar = planar_rgb12be_to_uv;
53.         break;
54.     case AV_PIX_FMT_GBRP14BE:
55.         c->readChrPlanar = planar_rgb14be_to_uv;
56.         break;
57.     case AV_PIX_FMT_GBRAP16BE:
58.     case AV_PIX_FMT_GBRP16BE:
59.         c->readChrPlanar = planar_rgb16be_to_uv;
60.         break;
61.     case AV_PIX_FMT_GBRAP:
62.     case AV_PIX_FMT_GBRP:
63.         c->readChrPlanar = planar_rgb_to_uv;
64.         break;
65. #if HAVE_BIGENDIAN
66.     case AV_PIX_FMT_YUV444P9LE:
67.     case AV_PIX_FMT_YUV422P9LE:
68.     case AV_PIX_FMT_YUV420P9LE:
69.     case AV_PIX_FMT_YUV422P10LE:
70.     case AV_PIX_FMT_YUV444P10LE:
71.     case AV_PIX_FMT_YUV420P10LE:
72.     case AV_PIX_FMT_YUV422P12LE:
73.     case AV_PIX_FMT_YUV444P12LE:
74.     case AV_PIX_FMT_YUV420P12LE:
75.     case AV_PIX_FMT_YUV422P14LE:
76.     case AV_PIX_FMT_YUV444P14LE:
77.     case AV_PIX_FMT_YUV420P14LE:
78.     case AV_PIX_FMT_YUV420P16LE:
79.     case AV_PIX_FMT_YUV422P16LE:
80.     case AV_PIX_FMT_YUV444P16LE:
81.
82.     case AV_PIX_FMT_YUVA444P9LE:
83.     case AV_PIX_FMT_YUVA422P9LE:
84.     case AV_PIX_FMT_YUVA420P9LE:
85.     case AV_PIX_FMT_YUVA444P10LE:
86.     case AV_PIX_FMT_YUVA422P10LE:
87.     case AV_PIX_FMT_YUVA420P10LE:
88.     case AV_PIX_FMT_YUVA420P16LE:
89.     case AV_PIX_FMT_YUVA422P16LE:
90.     case AV_PIX_FMT_YUVA444P16LE:
91.         c->chrToYV12 = bswap16UV_c;
92.         break;
93. #else
94.     case AV_PIX_FMT_YUV444P9BE:
95.     case AV_PIX_FMT_YUV422P9BE:
96.     case AV_PIX_FMT_YUV420P9BE:
97.     case AV_PIX_FMT_YUV444P10BE:
98.     case AV_PIX_FMT_YUV422P10BE:
99.     case AV_PIX_FMT_YUV420P10BE:
100.    case AV_PIX_FMT_YUV444P12BE:
101.    case AV_PIX_FMT_YUV422P12BE:
102.    case AV_PIX_FMT_YUV420P12BE:
103.    case AV_PIX_FMT_YUV444P14BE:
104.    case AV_PIX_FMT_YUV422P14BE:
105.    case AV_PIX_FMT_YUV420P14BE:
106.    case AV_PIX_FMT_YUV420P16BE:
107.    case AV_PIX_FMT_YUV422P16BE:
108.    case AV_PIX_FMT_YUV444P16BE:
109.
110.    case AV_PIX_FMT_YUVA444P9BE:
111.    case AV_PIX_FMT_YUVA422P9BE:
112.    case AV_PIX_FMT_YUVA420P9BE:
113.    case AV_PIX_FMT_YUVA444P10BE:
114.    case AV_PIX_FMT_YUVA422P10BE:
115.    case AV_PIX_FMT_YUVA420P10BE:
116.    case AV_PIX_FMT_YUVA420P16BE:
117.    case AV_PIX_FMT_YUVA422P16BE:
118.    case AV_PIX_FMT_YUVA444P16BE:
119.        c->chrToYV12 = bswap16UV_c;
120.        break;
121. #endif
122. }
123. if (c->chrSrcSubSample) {
124.     switch (srcFormat) {
125.     case AV_PIX_FMT_RGBA64BE:
126.         c->chrToYV12 = rgb64BEToUV_half_c;
127.         break;
128.     case AV_PIX_FMT_RGBA64LE:
129.         c->chrToYV12 = rgb64LEToUV_half_c;
130.         break;
131.     case AV_PIX_FMT_BGRA64BE:
132.         c->chrToYV12 = bgr64BEToUV_half_c;
133.         break;
134.     case AV_PIX_FMT_BGRA64LE:
135.         c->chrToYV12 = bgr64LEToUV_half_c;
136.         break;
137.     case AV_PIX_FMT_RGB48BE:
138.         c->chrToYV12 = rgb48BEToUV_half_c;
139.         break;
140.     case AV_PIX_FMT_RGB48LE:
141.         c->chrToYV12 = rgb48LEToUV_half_c;

```

```

140.         case AV_PIX_FMT_RGB48LE:
141.             c->chrToYV12 = rgb48LEToUV_half_c;
142.             break;
143.         case AV_PIX_FMT_BGR48BE:
144.             c->chrToYV12 = bgr48BEToUV_half_c;
145.             break;
146.         case AV_PIX_FMT_BGR48LE:
147.             c->chrToYV12 = bgr48LEToUV_half_c;
148.             break;
149.         case AV_PIX_FMT_RGB32:
150.             c->chrToYV12 = bgr32ToUV_half_c;
151.             break;
152.         case AV_PIX_FMT_RGB32_1:
153.             c->chrToYV12 = bgr321ToUV_half_c;
154.             break;
155.         case AV_PIX_FMT_BGR24:
156.             c->chrToYV12 = bgr24ToUV_half_c;
157.             break;
158.         case AV_PIX_FMT_BGR565LE:
159.             c->chrToYV12 = bgr16leToUV_half_c;
160.             break;
161.         case AV_PIX_FMT_BGR565BE:
162.             c->chrToYV12 = bgr16beToUV_half_c;
163.             break;
164.         case AV_PIX_FMT_BGR555LE:
165.             c->chrToYV12 = bgr15leToUV_half_c;
166.             break;
167.         case AV_PIX_FMT_BGR555BE:
168.             c->chrToYV12 = bgr15beToUV_half_c;
169.             break;
170.         case AV_PIX_FMT_GBRAP:
171.         case AV_PIX_FMT_GBRP:
172.             c->chrToYV12 = gbr24pToUV_half_c;
173.             break;
174.         case AV_PIX_FMT_BGR444LE:
175.             c->chrToYV12 = bgr12leToUV_half_c;
176.             break;
177.         case AV_PIX_FMT_BGR444BE:
178.             c->chrToYV12 = bgr12beToUV_half_c;
179.             break;
180.         case AV_PIX_FMT_BGR32:
181.             c->chrToYV12 = rgb32ToUV_half_c;
182.             break;
183.         case AV_PIX_FMT_BGR32_1:
184.             c->chrToYV12 = rgb321ToUV_half_c;
185.             break;
186.         case AV_PIX_FMT_RGB24:
187.             c->chrToYV12 = rgb24ToUV_half_c;
188.             break;
189.         case AV_PIX_FMT_RGB565LE:
190.             c->chrToYV12 = rgb16leToUV_half_c;
191.             break;
192.         case AV_PIX_FMT_RGB565BE:
193.             c->chrToYV12 = rgb16beToUV_half_c;
194.             break;
195.         case AV_PIX_FMT_RGB555LE:
196.             c->chrToYV12 = rgb15leToUV_half_c;
197.             break;
198.         case AV_PIX_FMT_RGB555BE:
199.             c->chrToYV12 = rgb15beToUV_half_c;
200.             break;
201.         case AV_PIX_FMT_RGB444LE:
202.             c->chrToYV12 = rgb12leToUV_half_c;
203.             break;
204.         case AV_PIX_FMT_RGB444BE:
205.             c->chrToYV12 = rgb12beToUV_half_c;
206.             break;
207.     }
208. } else {
209.     switch (srcFormat) {
210.         case AV_PIX_FMT_RGBA64BE:
211.             c->chrToYV12 = rgb64BEToUV_c;
212.             break;
213.         case AV_PIX_FMT_RGBA64LE:
214.             c->chrToYV12 = rgb64LEToUV_c;
215.             break;
216.         case AV_PIX_FMT_BGRA64BE:
217.             c->chrToYV12 = bgr64BEToUV_c;
218.             break;
219.         case AV_PIX_FMT_BGRA64LE:
220.             c->chrToYV12 = bgr64LEToUV_c;
221.             break;
222.         case AV_PIX_FMT_RGB48BE:
223.             c->chrToYV12 = rgb48BEToUV_c;
224.             break;
225.         case AV_PIX_FMT_RGB48LE:
226.             c->chrToYV12 = rgb48LEToUV_c;
227.             break;
228.         case AV_PIX_FMT_BGR48BE:
229.             c->chrToYV12 = bgr48BEToUV_c;
230.             break;
231.         case AV_PIX_FMT_BGR48LE:

```

```

232.         c->chrToYV12 = bgr48LEToUV_c;
233.         break;
234.     case AV_PIX_FMT_RGB32:
235.         c->chrToYV12 = bgr32ToUV_c;
236.         break;
237.     case AV_PIX_FMT_RGB32_1:
238.         c->chrToYV12 = bgr321ToUV_c;
239.         break;
240.     case AV_PIX_FMT_BGR24:
241.         c->chrToYV12 = bgr24ToUV_c;
242.         break;
243.     case AV_PIX_FMT_BGR565LE:
244.         c->chrToYV12 = bgr16leToUV_c;
245.         break;
246.     case AV_PIX_FMT_BGR565BE:
247.         c->chrToYV12 = bgr16beToUV_c;
248.         break;
249.     case AV_PIX_FMT_BGR555LE:
250.         c->chrToYV12 = bgr15leToUV_c;
251.         break;
252.     case AV_PIX_FMT_BGR555BE:
253.         c->chrToYV12 = bgr15beToUV_c;
254.         break;
255.     case AV_PIX_FMT_BGR444LE:
256.         c->chrToYV12 = bgr12leToUV_c;
257.         break;
258.     case AV_PIX_FMT_BGR444BE:
259.         c->chrToYV12 = bgr12beToUV_c;
260.         break;
261.     case AV_PIX_FMT_BGR32:
262.         c->chrToYV12 = rgb32ToUV_c;
263.         break;
264.     case AV_PIX_FMT_BGR32_1:
265.         c->chrToYV12 = rgb321ToUV_c;
266.         break;
267.     case AV_PIX_FMT_RGB24:
268.         c->chrToYV12 = rgb24ToUV_c;
269.         break;
270.     case AV_PIX_FMT_RGB565LE:
271.         c->chrToYV12 = rgb16leToUV_c;
272.         break;
273.     case AV_PIX_FMT_RGB565BE:
274.         c->chrToYV12 = rgb16beToUV_c;
275.         break;
276.     case AV_PIX_FMT_RGB555LE:
277.         c->chrToYV12 = rgb15leToUV_c;
278.         break;
279.     case AV_PIX_FMT_RGB555BE:
280.         c->chrToYV12 = rgb15beToUV_c;
281.         break;
282.     case AV_PIX_FMT_RGB444LE:
283.         c->chrToYV12 = rgb12leToUV_c;
284.         break;
285.     case AV_PIX_FMT_RGB444BE:
286.         c->chrToYV12 = rgb12beToUV_c;
287.         break;
288.     }
289. }
290.
291. c->lumToYV12 = NULL;
292. c->alpToYV12 = NULL;
293. switch (srcFormat) {
294. case AV_PIX_FMT_GBRP9LE:
295.     c->readLumPlanar = planar_rgb9le_to_y;
296.     break;
297. case AV_PIX_FMT_GBRP10LE:
298.     c->readLumPlanar = planar_rgb10le_to_y;
299.     break;
300. case AV_PIX_FMT_GBRP12LE:
301.     c->readLumPlanar = planar_rgb12le_to_y;
302.     break;
303. case AV_PIX_FMT_GBRP14LE:
304.     c->readLumPlanar = planar_rgb14le_to_y;
305.     break;
306. case AV_PIX_FMT_GBRAP16LE:
307. case AV_PIX_FMT_GBRP16LE:
308.     c->readLumPlanar = planar_rgb16le_to_y;
309.     break;
310. case AV_PIX_FMT_GBRP9BE:
311.     c->readLumPlanar = planar_rgb9be_to_y;
312.     break;
313. case AV_PIX_FMT_GBRP10BE:
314.     c->readLumPlanar = planar_rgb10be_to_y;
315.     break;
316. case AV_PIX_FMT_GBRP12BE:
317.     c->readLumPlanar = planar_rgb12be_to_y;
318.     break;
319. case AV_PIX_FMT_GBRP14BE:
320.     c->readLumPlanar = planar_rgb14be_to_y;
321.     break;
322. case AV_PIX_FMT_GBRAP16BE:

```



```

323.     case AV_PIX_FMT_GBRP16BE:
324.         c->readLumPlanar = planar_rgb16be_to_y;
325.         break;
326.     case AV_PIX_FMT_GBRAP:
327.         c->readAlpPlanar = planar_rgb_to_a;
328.     case AV_PIX_FMT_GBRP:
329.         c->readLumPlanar = planar_rgb_to_y;
330.         break;
331. #if HAVE_BIGENDIAN
332.     case AV_PIX_FMT_YUV444P9LE:
333.     case AV_PIX_FMT_YUV422P9LE:
334.     case AV_PIX_FMT_YUV420P9LE:
335.     case AV_PIX_FMT_YUV444P10LE:
336.     case AV_PIX_FMT_YUV422P10LE:
337.     case AV_PIX_FMT_YUV420P10LE:
338.     case AV_PIX_FMT_YUV444P12LE:
339.     case AV_PIX_FMT_YUV422P12LE:
340.     case AV_PIX_FMT_YUV420P12LE:
341.     case AV_PIX_FMT_YUV444P14LE:
342.     case AV_PIX_FMT_YUV422P14LE:
343.     case AV_PIX_FMT_YUV420P14LE:
344.     case AV_PIX_FMT_YUV420P16LE:
345.     case AV_PIX_FMT_YUV422P16LE:
346.     case AV_PIX_FMT_YUV444P16LE:
347.
348.     case AV_PIX_FMT_GRAY16LE:
349.         c->lumToYV12 = bswap16Y_c;
350.         break;
351.     case AV_PIX_FMT_YUVA444P9LE:
352.     case AV_PIX_FMT_YUVA422P9LE:
353.     case AV_PIX_FMT_YUVA420P9LE:
354.     case AV_PIX_FMT_YUVA444P10LE:
355.     case AV_PIX_FMT_YUVA422P10LE:
356.     case AV_PIX_FMT_YUVA420P10LE:
357.     case AV_PIX_FMT_YUVA420P16LE:
358.     case AV_PIX_FMT_YUVA422P16LE:
359.     case AV_PIX_FMT_YUVA444P16LE:
360.         c->lumToYV12 = bswap16Y_c;
361.         c->alpToYV12 = bswap16Y_c;
362.         break;
363. #else
364.     case AV_PIX_FMT_YUV444P9BE:
365.     case AV_PIX_FMT_YUV422P9BE:
366.     case AV_PIX_FMT_YUV420P9BE:
367.     case AV_PIX_FMT_YUV444P10BE:
368.     case AV_PIX_FMT_YUV422P10BE:
369.     case AV_PIX_FMT_YUV420P10BE:
370.     case AV_PIX_FMT_YUV444P12BE:
371.     case AV_PIX_FMT_YUV422P12BE:
372.     case AV_PIX_FMT_YUV420P12BE:
373.     case AV_PIX_FMT_YUV444P14BE:
374.     case AV_PIX_FMT_YUV422P14BE:
375.     case AV_PIX_FMT_YUV420P14BE:
376.     case AV_PIX_FMT_YUV420P16BE:
377.     case AV_PIX_FMT_YUV422P16BE:
378.     case AV_PIX_FMT_YUV444P16BE:
379.
380.     case AV_PIX_FMT_GRAY16BE:
381.         c->lumToYV12 = bswap16Y_c;
382.         break;
383.     case AV_PIX_FMT_YUVA444P9BE:
384.     case AV_PIX_FMT_YUVA422P9BE:
385.     case AV_PIX_FMT_YUVA420P9BE:
386.     case AV_PIX_FMT_YUVA444P10BE:
387.     case AV_PIX_FMT_YUVA422P10BE:
388.     case AV_PIX_FMT_YUVA420P10BE:
389.     case AV_PIX_FMT_YUVA420P16BE:
390.     case AV_PIX_FMT_YUVA422P16BE:
391.     case AV_PIX_FMT_YUVA444P16BE:
392.         c->lumToYV12 = bswap16Y_c;
393.         c->alpToYV12 = bswap16Y_c;
394.         break;
395. #endif
396.     case AV_PIX_FMT_YA16LE:
397.         c->lumToYV12 = read_ya16le_gray_c;
398.         c->alpToYV12 = read_ya16le_alpha_c;
399.         break;
400.     case AV_PIX_FMT_YA16BE:
401.         c->lumToYV12 = read_ya16be_gray_c;
402.         c->alpToYV12 = read_ya16be_alpha_c;
403.         break;
404.     case AV_PIX_FMT_YUVV422:
405.     case AV_PIX_FMT_VYU422:
406.     case AV_PIX_FMT_YA8:
407.         c->lumToYV12 = yuy2ToY_c;
408.         break;
409.     case AV_PIX_FMT_UYVY422:
410.         c->lumToYV12 = uyvyToY_c;
411.         break;
412.     case AV_PIX_FMT_BGR24:
413.         c->lumToYV12 = bgr24ToY_c;

```

```

414.         break;
415.     case AV_PIX_FMT_BGR565LE:
416.         c->lumToYV12 = bgr16leToY_c;
417.         break;
418.     case AV_PIX_FMT_BGR565BE:
419.         c->lumToYV12 = bgr16beToY_c;
420.         break;
421.     case AV_PIX_FMT_BGR555LE:
422.         c->lumToYV12 = bgr15leToY_c;
423.         break;
424.     case AV_PIX_FMT_BGR555BE:
425.         c->lumToYV12 = bgr15beToY_c;
426.         break;
427.     case AV_PIX_FMT_BGR444LE:
428.         c->lumToYV12 = bgr12leToY_c;
429.         break;
430.     case AV_PIX_FMT_BGR444BE:
431.         c->lumToYV12 = bgr12beToY_c;
432.         break;
433.     case AV_PIX_FMT_RGB24:
434.         c->lumToYV12 = rgb24ToY_c;
435.         break;
436.     case AV_PIX_FMT_RGB565LE:
437.         c->lumToYV12 = rgb16leToY_c;
438.         break;
439.     case AV_PIX_FMT_RGB565BE:
440.         c->lumToYV12 = rgb16beToY_c;
441.         break;
442.     case AV_PIX_FMT_RGB555LE:
443.         c->lumToYV12 = rgb15leToY_c;
444.         break;
445.     case AV_PIX_FMT_RGB555BE:
446.         c->lumToYV12 = rgb15beToY_c;
447.         break;
448.     case AV_PIX_FMT_RGB444LE:
449.         c->lumToYV12 = rgb12leToY_c;
450.         break;
451.     case AV_PIX_FMT_RGB444BE:
452.         c->lumToYV12 = rgb12beToY_c;
453.         break;
454.     case AV_PIX_FMT_RGB8:
455.     case AV_PIX_FMT_BGR8:
456.     case AV_PIX_FMT_PAL8:
457.     case AV_PIX_FMT_BGR4_BYTE:
458.     case AV_PIX_FMT_RGB4_BYTE:
459.         c->lumToYV12 = palToY_c;
460.         break;
461.     case AV_PIX_FMT_MONOBLACK:
462.         c->lumToYV12 = monoblack2Y_c;
463.         break;
464.     case AV_PIX_FMT_MONOWHITE:
465.         c->lumToYV12 = monowhite2Y_c;
466.         break;
467.     case AV_PIX_FMT_RGB32:
468.         c->lumToYV12 = bgr32ToY_c;
469.         break;
470.     case AV_PIX_FMT_RGB32_1:
471.         c->lumToYV12 = bgr321ToY_c;
472.         break;
473.     case AV_PIX_FMT_BGR32:
474.         c->lumToYV12 = rgb32ToY_c;
475.         break;
476.     case AV_PIX_FMT_BGR32_1:
477.         c->lumToYV12 = rgb321ToY_c;
478.         break;
479.     case AV_PIX_FMT_RGB48BE:
480.         c->lumToYV12 = rgb48BEToY_c;
481.         break;
482.     case AV_PIX_FMT_RGB48LE:
483.         c->lumToYV12 = rgb48LEToY_c;
484.         break;
485.     case AV_PIX_FMT_BGR48BE:
486.         c->lumToYV12 = bgr48BEToY_c;
487.         break;
488.     case AV_PIX_FMT_BGR48LE:
489.         c->lumToYV12 = bgr48LEToY_c;
490.         break;
491.     case AV_PIX_FMT_RGBA64BE:
492.         c->lumToYV12 = rgb64BEToY_c;
493.         break;
494.     case AV_PIX_FMT_RGBA64LE:
495.         c->lumToYV12 = rgb64LEToY_c;
496.         break;
497.     case AV_PIX_FMT_BGRA64BE:
498.         c->lumToYV12 = bgr64BEToY_c;
499.         break;
500.     case AV_PIX_FMT_BGRA64LE:
501.         c->lumToYV12 = bgr64LEToY_c;
502.     }
503.     if (c->alpPixBuf) {
504.         if (is16BPS(srcFormat) || isNBPS(srcFormat)) {

```

```

505.         if (HAVE_BIGENDIAN == !isBE(srcFormat))
506.             c->alpToYV12 = bswap16Y_c;
507.     }
508.     switch (srcFormat) {
509.     case AV_PIX_FMT_BGRA64LE:
510.     case AV_PIX_FMT_BGRA64BE:
511.     case AV_PIX_FMT_RGBA64LE:
512.     case AV_PIX_FMT_RGBA64BE:  c->alpToYV12 = rgba64ToA_c; break;
513.     case AV_PIX_FMT_BGRA:
514.     case AV_PIX_FMT_RGBA:
515.         c->alpToYV12 = rgbaToA_c;
516.         break;
517.     case AV_PIX_FMT_ABGR:
518.     case AV_PIX_FMT_ARGB:
519.         c->alpToYV12 = abgrToA_c;
520.         break;
521.     case AV_PIX_FMT_YA8:
522.         c->alpToYV12 = uyvyToY_c;
523.         break;
524.     case AV_PIX_FMT_PAL8 :
525.         c->alpToYV12 = palToA_c;
526.         break;
527.     }
528. }
529. }

```

ff_sws_init_input_funcs()根据输入像素格式的不同，对以下几个函数指针进行赋值：

lumToYV12：转换得到Y分量。

chrToYV12：转换得到UV分量。

alpToYV12：转换得到Alpha分量。

readLumPlanar：读取planar格式的数据转换为Y。

readChrPlanar：读取planar格式的数据转换为UV。

下面看几个例子。

当输入像素格式为AV_PIX_FMT_RGB24的时候，lumToYV12()指针指向的函数是rgb24ToY_c()，如下所示。

```

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

```

rgb24ToY_c()

rgb24ToY_c()的定义如下。

```

[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. }

```

从源代码中可以看出，该函数主要完成了以下三步：

1. 取系数。通过读取rgb2yuv数组中存储的参数获得R，G，B每个分量的系数。
2. 取像素值。分别读取R，G，B每个分量的像素值。
3. 计算得到亮度值。根据R，G，B的系数和值，计算得到亮度值Y。

当输入像素格式为AV_PIX_FMT_RGB24的时候，chrToYV12 ()指针指向的函数是rgb24ToUV_half_c()，如下所示。



```

[cpp]
1.  case AV_PIX_FMT_RGB24:
2.      c->chrToYV12 = rgb24ToUV_half_c;
3.      break;



```

rgb24ToUV_half_c()

rgb24ToUV_half_c()定义如下。



```
[cpp]    
1. static void rgb24ToUV_half_c(uint8_t *_dstU, uint8_t *_dstV, const uint8_t *unused0, const uint8_t *src1,  
2. const uint8_t *src2, int width, uint32_t *rgb2yuv)  
3. {  
4.     int16_t *dstU = (int16_t *)_dstU;  
5.     int16_t *dstV = (int16_t *)_dstV;  
6.     int i;  
7.     int32_t ru = rgb2yuv[RU_IDX], gu = rgb2yuv[GU_IDX], bu = rgb2yuv[BU_IDX];  
8.     int32_t rv = rgb2yuv[RV_IDX], gv = rgb2yuv[GV_IDX], bv = rgb2yuv[BV_IDX];  
9.     av_assert1(src1 == src2);  
10.    for (i = 0; i < width; i++) {  
11.        int r = src1[6 * i + 0] + src1[6 * i + 3];  
12.        int g = src1[6 * i + 1] + src1[6 * i + 4];  
13.        int b = src1[6 * i + 2] + src1[6 * i + 5];  
14.  
15.        dstU[i] = (ru*r + gu*g + bu*b + (256<<RGB2YUV_SHIFT) + (1<<(RGB2YUV_SHIFT-6)))>>(RGB2YUV_SHIFT-5);  
16.        dstV[i] = (rv*r + gv*g + bv*b + (256<<RGB2YUV_SHIFT) + (1<<(RGB2YUV_SHIFT-6)))>>(RGB2YUV_SHIFT-5);  
17.    }  
18. }
```

rgb24ToUV_half_c()的过程相比rgb24ToY_c()要稍微复杂些。这主要是因为U、V取值的数量只有Y的一半。因此需要首先求出每2个像素点的平均值之后，再进行计算。当输入像素格式为AV_PIX_FMT_GBRP（注意这个是planar格式，三个分量分别为G，B，R）的时候，readLumPlanar指向的函数是planar_rgb_to_y()，如下所示。

```
[cpp]    
1. case AV_PIX_FMT_GBRP:  
2.     c->readLumPlanar = planar_rgb_to_y;  
3.     break;
```

planar_rgb_to_y()

planar_rgb_to_y()定义如下。

```
[cpp]    
1. static void planar_rgb_to_y(uint8_t *_dst, const uint8_t *src[4], int width, int32_t *rgb2yuv)  
2. {  
3.     uint16_t *dst = (uint16_t *)_dst;  
4.     int32_t ry = rgb2yuv[RY_IDX], gy = rgb2yuv[GY_IDX], by = rgb2yuv[BY_IDX];  
5.     int i;  
6.     for (i = 0; i < width; i++) {  
7.         int g = src[0][i];  
8.         int b = src[1][i];  
9.         int r = src[2][i];  
10.  
11.        dst[i] = (ry*r + gy*g + by*b + (0x801<<(RGB2YUV_SHIFT-7))) >> (RGB2YUV_SHIFT-6);  
12.    }  
13. }
```

可以看出处理planar格式的GBR数据和处理packed格式的RGB数据的方法是基本一样的，在这里不再重复。

ff_sws_init_range_convert()

ff_sws_init_range_convert()用于初始化像素值范围转换的函数，它的定义位于libswscale\swscale.c，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. }
```

ff_sws_init_range_convert()包含了两种像素取值范围的转换：

lumConvertRange：亮度分量取值范围的转换。

chrConvertRange：色度分量取值范围的转换。

从JPEG标准转换为MPEG标准的函数有：lumRangeFromJpeg_c()和chrRangeFromJpeg_c()。

lumRangeFromJpeg_c()

亮度转换（0-255转换为16-235）函数lumRangeFromJpeg_c()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. }
```

可以简单代入一个数字验证一下上述函数的正确性。该函数将亮度值“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。

后续几个函数都可以用上面描述的方法进行验证，就不再重复了。

chrRangeFromJpeg_c()



色度转换（0-255转换为16-240）函数chrRangeFromJpeg_c()如下所示。

```
[cpp]
1. static void chrRangeFromJpeg_c(int16_t *dstU, int16_t *dstV, int width)
2. {
3.     int i;
4.     for (i = 0; i < width; i++) {
5.         dstU[i] = (dstU[i] * 1799 + 4081085) >> 11; // 1469
6.         dstV[i] = (dstV[i] * 1799 + 4081085) >> 11; // 1469
7.     }
8. }
```

从MPEG标准转换为JPEG标准的函数有：lumRangeToJpeg_c()和chrRangeToJpeg_c()。



lumRangeToJpeg_c()

亮度转换（16-235转换为0-255）函数lumRangeToJpeg_c()定义如下所示。

```
[cpp]    
1. static void lumRangeToJpeg_c(int16_t *dst, int width)  
2. {  
3.     int i;  
4.     for (i = 0; i < width; i++)  
5.         dst[i] = (FFMIN(dst[i], 30189) * 19077 - 39057361) >> 14;  
6. }
```

chrRangeToJpeg_c()

色度转换（16-240转换为0-255）函数chrRangeToJpeg_c()定义如下所示。

```
[cpp]    
1. static void chrRangeToJpeg_c(int16_t *dstU, int16_t *dstV, int width)  
2. {  
3.     int i;  
4.     for (i = 0; i < width; i++) {  
5.         dstU[i] = (FFMIN(dstU[i], 30775) * 4663 - 9289992) >> 12; // -264  
6.         dstV[i] = (FFMIN(dstV[i], 30775) * 4663 - 9289992) >> 12; // -264  
7.     }  
8. }
```

至此sws_getContext()的源代码就基本上分析完毕了。

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/44305697>

文章标签： [swscale](#) [ffmpeg](#) [源代码](#) [YUV](#) [RGB](#)

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