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：23102

==========

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libswsscale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。

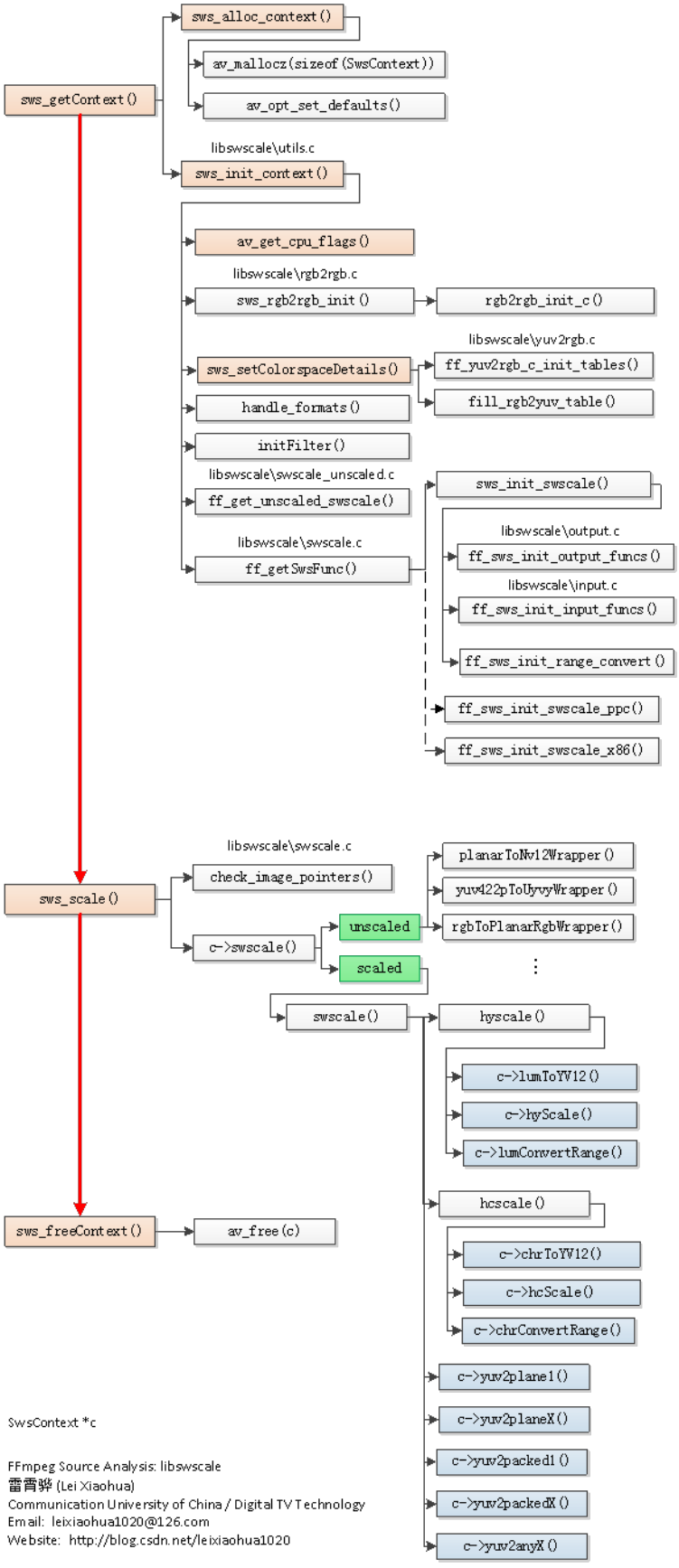
sws_freeContext()：释放一个SwsContext。

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()。

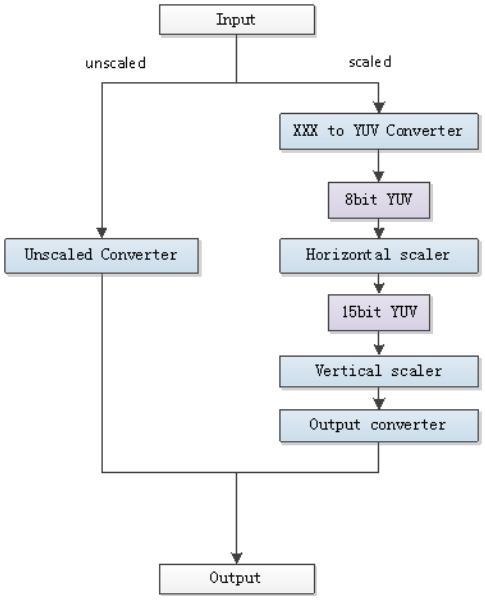
函数调用结构图

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
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
/* This struct should be aligned on at least a 32-byte boundary. */
typedef struct SwsContext {
    /**
     * info on struct for av_log
     */
    const AVClass *av_class;

    /**
     * Note that src, dst, srcStride, dstStride will be copied in the
     * sws_scale() wrapper so they can be freely modified here.
     */
    SwsFunc swscale;
    int srcW;                     ///< Width  of source      luma/alpha planes.
    int srcH;                     ///< Height of source      luma/alpha planes.
    int dstH;                     ///< Height of destination luma/alpha planes.
    int chrSrcW;                  ///< Width  of source      chroma     planes.
    int chrSrcH;                  ///< Height of source      chroma     planes.
    int chrDstW;                  ///< Width  of destination chroma     planes.
    int chrDstH;                  ///< Height of destination chroma     planes.
    int lumXInc, chrXInc;
    int lumYInc, chrYInc;
    enum AVPixelFormat dstFormat; ///< Destination pixel format.
    enum AVPixelFormat srcFormat; ///< Source      pixel format.
    int dstFormatBpp;             ///< Number of bits per pixel of the destination pixel format.
    int srcFormatBpp;             ///< Number of bits per pixel of the source      pixel format.
    int dstBpc, srcBpc;
    int chrSrcHSubSample;         ///< Binary logarithm of horizontal subsampling factor between luma/alpha and chroma planes in sour
      image.
    int chrSrcVSubSample;         ///< Binary logarithm of vertical   subsampling factor between luma/alpha and chroma planes in sour
      image.
    int chrDstHSubSample;         ///< Binary logarithm of horizontal subsampling factor between luma/alpha and chroma planes in dest
```

```
    int chrDstHSubSample;          ///< Binary logarithm of horizontal subsampling factor between luma/alpha and chroma planes in dest
ation image.
    int chrDstVSubSample;          ///< Binary logarithm of vertical   subsampling factor between luma/alpha and chroma planes in dest
ation image.
    int vChrDrop;                  ///< Binary logarithm of extra vertical subsampling factor in source image chroma planes specified
 user.
    int sliceDir;                  ///< Direction that slices are fed to the scaler (1 = top-to-bottom, -1 = bottom-to-top).
    double param[2];               ///< Input parameters for scaling algorithms that need them.

    /* The cascaded_* fields allow spliting a scaler task into multiple
     * sequential steps, this is for example used to limit the maximum
     * downscaling factor that needs to be supported in one scaler.
     */
    struct SwsContext *cascaded_context[2];
    int cascaded_tmpStride[4];
    uint8_t *cascaded_tmp[4];

    uint32_t pal_yuv[256];
    uint32_t pal_rgb[256];

    /**
     * @name Scaled horizontal lines ring buffer.
     * The horizontal scaler keeps just enough scaled lines in a ring buffer
     * so they may be passed to the vertical scaler. The pointers to the
     * allocated buffers for each line are duplicated in sequence in the ring
     * buffer to simplify indexing and avoid wrapping around between lines
     * inside the vertical scaler code. The wrapping is done before the
     * vertical scaler is called.
     */
    //@{
    int16_t **lumPixBuf;           ///< Ring buffer for scaled horizontal luma   plane lines to be fed to the vertical scaler.
    int16_t **chrUPixBuf;          ///< Ring buffer for scaled horizontal chroma plane lines to be fed to the vertical scaler.
    int16_t **chrVPixBuf;          ///< Ring buffer for scaled horizontal chroma plane lines to be fed to the vertical scaler.
    int16_t **alpPixBuf;           ///< Ring buffer for scaled horizontal alpha  plane lines to be fed to the vertical scaler.
    int vLumBufSize;               ///< Number of vertical luma/alpha lines allocated in the ring buffer.
    int vChrBufSize;               ///< Number of vertical chroma     lines allocated in the ring buffer.
    int lastInLumBuf;              ///< Last scaled horizontal luma/alpha line from source in the ring buffer.
    int lastInChrBuf;              ///< Last scaled horizontal chroma     line from source in the ring buffer.
    int lumBufIndex;               ///< Index in ring buffer of the last scaled horizontal luma/alpha line from source.
    int chrBufIndex;               ///< Index in ring buffer of the last scaled horizontal chroma     line from source.
    //@}

    uint8_t *formatConvBuffer;

    /**
     * @name Horizontal and vertical filters.
     * To better understand the following fields, here is a pseudo-code of
     * their usage in filtering a horizontal line:
     * @code
     * for (i = 0; i < width; i++) {
     *     dst[i] = 0;
     *     for (j = 0; j < filterSize; j++)
     *         dst[i] += src[ filterPos[i] + j ] * filter[ filterSize * i + j ];
     *     dst[i] >>= FRAC_BITS; // The actual implementation is fixed-point.
     * }
     * @endcode
     */
    //@{
    int16_t *hLumFilter;           ///< Array of horizontal filter coefficients for luma/alpha planes.
    int16_t *hChrFilter;           ///< Array of horizontal filter coefficients for chroma     planes.
    int16_t *vLumFilter;           ///< Array of vertical   filter coefficients for luma/alpha planes.
    int16_t *vChrFilter;           ///< Array of vertical   filter coefficients for chroma     planes.
    int32_t *hLumFilterPos;        ///< Array of horizontal filter starting positions for each dst[i] for luma/alpha planes.
    int32_t *hChrFilterPos;        ///< Array of horizontal filter starting positions for each dst[i] for chroma     planes.
    int32_t *vLumFilterPos;        ///< Array of vertical   filter starting positions for each dst[i] for luma/alpha planes.
    int32_t *vChrFilterPos;        ///< Array of vertical   filter starting positions for each dst[i] for chroma     planes.
    int hLumFilterSize;            ///< Horizontal filter size for luma/alpha pixels.
    int hChrFilterSize;            ///< Horizontal filter size for chroma     pixels.
    int vLumFilterSize;            ///< Vertical   filter size for luma/alpha pixels.
    int vChrFilterSize;            ///< Vertical   filter size for chroma     pixels.
    //@}

    int lumMmxextFilterCodeSize;  ///< Runtime-generated MMXEXT horizontal fast bilinear scaler code size for luma/alpha planes.
    int chrMmxextFilterCodeSize;  ///< Runtime-generated MMXEXT horizontal fast bilinear scaler code size for chroma planes.
    uint8_t *lumMmxextFilterCode; ///< Runtime-generated MMXEXT horizontal fast bilinear scaler code for luma/alpha planes.
    uint8_t *chrMmxextFilterCode; ///< Runtime-generated MMXEXT horizontal fast bilinear scaler code for chroma planes.

    int canMMXEXTBeUsed;

    int dstY;                      ///< Last destination vertical line output from last slice.
    int flags;                     ///< Flags passed by the user to select scaler algorithm, optimizations, subsampling, etc...
    void *yuvTable;             // pointer to the yuv->rgb table start so it can be freed()
    // alignment ensures the offset can be added in a single
    // instruction on e.g. ARM
    DECLARE_ALIGNED(16, int, table_gV)[256 + 2*YUVRGB_TABLE_HEADROOM];
    uint8_t *table_rV[256 + 2*YUVRGB_TABLE_HEADROOM];
    uint8_t *table_gU[256 + 2*YUVRGB_TABLE_HEADROOM];
    uint8_t *table_bU[256 + 2*YUVRGB_TABLE_HEADROOM];
    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
#define RY_IDX 0
#define GY_IDX 1
```

```
#define BY_IDX 2
#define RU_IDX 3
#define GU_IDX 4
#define BU_IDX 5
#define RV_IDX 6
#define GV_IDX 7
#define BV_IDX 8
#define RGB2YUV_SHIFT 15

    int *dither_error[4];

    //Colorspace stuff
    int contrast, brightness, saturation;    // for sws_getColorspaceDetails
    int srcColorspaceTable[4];
    int dstColorspaceTable[4];
    int srcRange;                ///< 0 = MPG YUV range, 1 = JPG YUV range (source      image).
    int dstRange;                ///< 0 = MPG YUV range, 1 = JPG YUV range (destination image).
    int src0Alpha;
    int dst0Alpha;
    int srcXYZ;
    int dstXYZ;
    int src_h_chr_pos;
    int dst_h_chr_pos;
    int src_v_chr_pos;
    int dst_v_chr_pos;
    int yuv2rgb_y_offset;
    int yuv2rgb_y_coeff;
    int yuv2rgb_v2r_coeff;
    int yuv2rgb_v2g_coeff;
    int yuv2rgb_u2g_coeff;
    int yuv2rgb_u2b_coeff;

#define RED_DITHER            "0*8"
#define GREEN_DITHER          "1*8"
#define BLUE_DITHER           "2*8"
#define Y_COEFF               "3*8"
#define VR_COEFF              "4*8"
#define UB_COEFF              "5*8"
#define VG_COEFF              "6*8"
#define UG_COEFF              "7*8"
#define Y_OFFSET              "8*8"
#define U_OFFSET              "9*8"
#define V_OFFSET              "10*8"
#define LUM_MMX_FILTER_OFFSET "11*8"
#define CHR_MMX_FILTER_OFFSET "11*8+4*4*"AV_STRINGIFY(MAX_FILTER_SIZE)
#define DSTW_OFFSET           "11*8+4*4*"AV_STRINGIFY(MAX_FILTER_SIZE)"*2"
#define ESP_OFFSET            "11*8+4*4*"AV_STRINGIFY(MAX_FILTER_SIZE)"*2+8"
#define VROUNDER_OFFSET       "11*8+4*4*"AV_STRINGIFY(MAX_FILTER_SIZE)"*2+16"
#define U_TEMP                "11*8+4*4*"AV_STRINGIFY(MAX_FILTER_SIZE)"*2+24"
#define V_TEMP                "11*8+4*4*"AV_STRINGIFY(MAX_FILTER_SIZE)"*2+32"
#define Y_TEMP                "11*8+4*4*"AV_STRINGIFY(MAX_FILTER_SIZE)"*2+40"
#define ALP_MMX_FILTER_OFFSET "11*8+4*4*"AV_STRINGIFY(MAX_FILTER_SIZE)"*2+48"
#define UV_OFF_PX             "11*8+4*4*"AV_STRINGIFY(MAX_FILTER_SIZE)"*3+48"
#define UV_OFF_BYTE           "11*8+4*4*"AV_STRINGIFY(MAX_FILTER_SIZE)"*3+56"
#define DITHER16              "11*8+4*4*"AV_STRINGIFY(MAX_FILTER_SIZE)"*3+64"
#define DITHER32              "11*8+4*4*"AV_STRINGIFY(MAX_FILTER_SIZE)"*3+80"
#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

    DECLARE_ALIGNED(8, uint64_t, redDither);
    DECLARE_ALIGNED(8, uint64_t, greenDither);
    DECLARE_ALIGNED(8, uint64_t, blueDither);

    DECLARE_ALIGNED(8, uint64_t, yCoeff);
    DECLARE_ALIGNED(8, uint64_t, vrCoeff);
    DECLARE_ALIGNED(8, uint64_t, ubCoeff);
    DECLARE_ALIGNED(8, uint64_t, vgCoeff);
    DECLARE_ALIGNED(8, uint64_t, ugCoeff);
    DECLARE_ALIGNED(8, uint64_t, yOffset);
    DECLARE_ALIGNED(8, uint64_t, uOffset);
    DECLARE_ALIGNED(8, uint64_t, vOffset);
    int32_t lumMmxFilter[4 * MAX_FILTER_SIZE];
    int32_t chrMmxFilter[4 * MAX_FILTER_SIZE];
    int dstW;                    ///< Width  of destination luma/alpha planes.
    DECLARE_ALIGNED(8, uint64_t, esp);
    DECLARE_ALIGNED(8, uint64_t, vRounder);
    DECLARE_ALIGNED(8, uint64_t, u_temp);
    DECLARE_ALIGNED(8, uint64_t, v_temp);
    DECLARE_ALIGNED(8, uint64_t, y_temp);
    int32_t alpMmxFilter[4 * MAX_FILTER_SIZE];
    // alignment of these values is not necessary, but merely here
    // to maintain the same offset across x8632 and x86-64. Once we
    // use proper offset macros in the asm, they can be removed.
    DECLARE_ALIGNED(8, ptrdiff_t, uv_off); ///< offset (in pixels) between u and v planes
    DECLARE_ALIGNED(8, ptrdiff_t, uv_offx2); ///< offset (in bytes) between u and v planes
    DECLARE_ALIGNED(8, uint16_t, dither16)[8];
    DECLARE_ALIGNED(8, uint32_t, dither32)[8];

    const uint8_t *chrDither8, *lumDither8;

#if HAVE_ALTIVEC
```

```
    vector signed short   CY;
    vector signed short   CRV;
    vector signed short   CBU;
    vector signed short   CGU;
    vector signed short   CGV;
    vector signed short   OY;
    vector unsigned short CSHIFT;
    vector signed short  *vYCoeffsBank, *vCCoeffsBank;
#endif

    int use_mmx_vfilter;

/* pre defined color-spaces gamma */
#define XYZ_GAMMA (2.6f)
#define RGB_GAMMA (2.2f)
    int16_t *xyzgamma;
    int16_t *rgbgamma;
    int16_t *xyzgammainv;
    int16_t *rgbgammainv;
    int16_t xyz2rgb_matrix[3][4];
    int16_t rgb2xyz_matrix[3][4];

    /* function pointers for swscale() */
    yuv2planar1_fn yuv2plane1;
    yuv2planarX_fn yuv2planeX;
    yuv2interleavedX_fn yuv2nv12cX;
    yuv2packed1_fn yuv2packed1;
    yuv2packed2_fn yuv2packed2;
    yuv2packedX_fn yuv2packedX;
    yuv2anyX_fn yuv2anyX;

    /// Unscaled conversion of luma plane to YV12 for horizontal scaler.
    void (*lumToYV12)(uint8_t *dst, const uint8_t *src, const uint8_t *src2, const uint8_t *src3,
                      int width, uint32_t *pal);
    /// Unscaled conversion of alpha plane to YV12 for horizontal scaler.
    void (*alpToYV12)(uint8_t *dst, const uint8_t *src, const uint8_t *src2, const uint8_t *src3,
                      int width, uint32_t *pal);
    /// Unscaled conversion of chroma planes to YV12 for horizontal scaler.
    void (*chrToYV12)(uint8_t *dstU, uint8_t *dstV,
                      const uint8_t *src1, const uint8_t *src2, const uint8_t *src3,
                      int width, uint32_t *pal);

    /**
     * Functions to read planar input, such as planar RGB, and convert
     * internally to Y/UV/A.
     */
    /** @{ */
    void (*readLumPlanar)(uint8_t *dst, const uint8_t *src[4], int width, int32_t *rgb2yuv);
    void (*readChrPlanar)(uint8_t *dstU, uint8_t *dstV, const uint8_t *src[4],
                          int width, int32_t *rgb2yuv);
    void (*readAlpPlanar)(uint8_t *dst, const uint8_t *src[4], int width, int32_t *rgb2yuv);
    /** @} */

    /**
     * Scale one horizontal line of input data using a bilinear filter
     * to produce one line of output data. Compared to SwsContext->hScale(),
     * please take note of the following caveats when using these:
     * - Scaling is done using only 7bit instead of 14bit coefficients.
     * - You can use no more than 5 input pixels to produce 4 output
     *   pixels. Therefore, this filter should not be used for downscaling
     *   by more than ~20% in width (because that equals more than 5/4th
     *   downscaling and thus more than 5 pixels input per 4 pixels output).
     * - In general, bilinear filters create artifacts during downscaling
     *   (even when <20%), because one output pixel will span more than one
     *   input pixel, and thus some pixels will need edges of both neighbor
     *   pixels to interpolate the output pixel. Since you can use at most
     *   two input pixels per output pixel in bilinear scaling, this is
     *   impossible and thus downscaling by any size will create artifacts.
     * To enable this type of scaling, set SWS_FLAG_FAST_BILINEAR
     * in SwsContext->flags.
     */
    /** @{ */
    void (*hyscale_fast)(struct SwsContext *c,
                         int16_t *dst, int dstWidth,
                         const uint8_t *src, int srcW, int xInc);
    void (*hcscale_fast)(struct SwsContext *c,
                         int16_t *dst1, int16_t *dst2, int dstWidth,
                         const uint8_t *src1, const uint8_t *src2,
                         int srcW, int xInc);
    /** @} */

    /**
     * Scale one horizontal line of input data using a filter over the input
     * lines, to produce one (differently sized) line of output data.
     *
     * @param dst        pointer to destination buffer for horizontally scaled
     *                   data. If the number of bits per component of one
     *                   destination pixel (SwsContext->dstBpc) is <= 10, data
     *                   will be 15bpc in 16bits (int16_t) width. Else (i.e.
     *                   SwsContext->dstBpc == 16), data will be 19bpc in
     *                   32bits (int32_t) width.
```

```
     * @param dstW       width of destination image
     * @param src        pointer to source data to be scaled. If the number of
     *                   bits per component of a source pixel (SwsContext->srcBpc)
     *                   is 8, this is 8bpc in 8bits (uint8_t) width. Else
     *                   (i.e. SwsContext->dstBpc > 8), this is native depth
     *                   in 16bits (uint16_t) width. In other words, for 9-bit
     *                   YUV input, this is 9bpc, for 10-bit YUV input, this is
     *                   10bpc, and for 16-bit RGB or YUV, this is 16bpc.
     * @param filter     filter coefficients to be used per output pixel for
     *                   scaling. This contains 14bpp filtering coefficients.
     *                   Guaranteed to contain dstW * filterSize entries.
     * @param filterPos  position of the first input pixel to be used for
     *                   each output pixel during scaling. Guaranteed to
     *                   contain dstW entries.
     * @param filterSize the number of input coefficients to be used (and
     *                   thus the number of input pixels to be used) for
     *                   creating a single output pixel. Is aligned to 4
     *                   (and input coefficients thus padded with zeroes)
     *                   to simplify creating SIMD code.
     */
    /** @{ */
    void (*hyScale)(struct SwsContext *c, int16_t *dst, int dstW,
                    const uint8_t *src, const int16_t *filter,
                    const int32_t *filterPos, int filterSize);
    void (*hcScale)(struct SwsContext *c, int16_t *dst, int dstW,
                    const uint8_t *src, const int16_t *filter,
                    const int32_t *filterPos, int filterSize);
    /** @} */

    /// Color range conversion function for luma plane if needed.
    void (*lumConvertRange)(int16_t *dst, int width);
    /// Color range conversion function for chroma planes if needed.
    void (*chrConvertRange)(int16_t *dst1, int16_t *dst2, int width);

    int needs_hcscale; ///< Set if there are chroma planes to be converted.

    SwsDither dither;
} SwsContext;
```

这个结构体的定义确实比较复杂，里面包含了libswscale所需要的全部变量。一一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。

```cpp
/**
 * Allocate and return an SwsContext. You need it to perform
 * scaling/conversion operations using sws_scale().
 *
 * @param srcW the width of the source image
 * @param srcH the height of the source image
 * @param srcFormat the source image format
 * @param dstW the width of the destination image
 * @param dstH the height of the destination image
 * @param dstFormat the destination image format
 * @param flags specify which algorithm and options to use for rescaling
 * @return a pointer to an allocated context, or NULL in case of error
 * @note this function is to be removed after a saner alternative is
 *       written
 */
struct SwsContext *sws_getContext(int srcW, int srcH, enum AVPixelFormat srcFormat,
                                  int dstW, int dstH, enum AVPixelFormat dstFormat,
                                  int flags, SwsFilter *srcFilter,
                                  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libswscale\utils.c，如下所示。

```cpp
SwsContext *sws_getContext(int srcW, int srcH, enum AVPixelFormat srcFormat,
                           int dstW, int dstH, enum AVPixelFormat dstFormat,
                           int flags, SwsFilter *srcFilter,
                           SwsFilter *dstFilter, const double *param)
{
    SwsContext *c;

    if (!(c = sws_alloc_context()))
        return NULL;

    c->flags     = flags;
    c->srcW      = srcW;
    c->srcH      = srcH;
    c->dstW      = dstW;
    c->dstH      = dstH;
    c->srcFormat = srcFormat;
    c->dstFormat = dstFormat;

    if (param) {
        c->param[0] = param[0];
        c->param[1] = param[1];
    }

    if (sws_init_context(c, srcFilter, dstFilter) < 0) {
        sws_freeContext(c);
        return NULL;
    }

    return c;
}
```

从sws_getContext()的定义中可以看出，它首先调用了一个函数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。

```cpp
/**
 * Allocate an empty SwsContext. This must be filled and passed to
 * sws_init_context(). For filling see AVOptions, options.c and
 * sws_setColorspaceDetails().
 */
struct SwsContext *sws_alloc_context(void);
```

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

```cpp
SwsContext *sws_alloc_context(void)
{
    SwsContext *c = av_mallocz(sizeof(SwsContext));

    av_assert0(offsetof(SwsContext, redDither) + DITHER32_INT == offsetof(SwsContext, dither32));

    if (c) {
        c->av_class = &sws_context_class;
        av_opt_set_defaults(c);
    }

    return c;
}
```

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。

```cpp
/**
 * Initialize the swscaler context sws_context.
 *
 * @return zero or positive value on success, a negative value on
 * error
 */
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。

```cpp
av_cold int sws_init_context(SwsContext *c, SwsFilter *srcFilter,
                             SwsFilter *dstFilter)
{
    int i, j;
    int usesVFilter, usesHFilter;
    int unscaled;
    SwsFilter dummyFilter = { NULL, NULL, NULL, NULL };
    int srcW              = c->srcW;
    int srcH              = c->srcH;
    int dstW              = c->dstW;
    int dstH              = c->dstH;
    int dst_stride        = FFALIGN(dstW * sizeof(int16_t) + 66, 16);
    int flags, cpu_flags;
    enum AVPixelFormat srcFormat = c->srcFormat;
    enum AVPixelFormat dstFormat = c->dstFormat;
    const AVPixFmtDescriptor *desc_src;
    const AVPixFmtDescriptor *desc_dst;
    int ret = 0;
    //获取
    cpu_flags = av_get_cpu_flags();
    flags     = c->flags;
    emms_c();
    if (!rgb15to16)
        sws_rgb2rgb_init();
    //如果输入的宽高和输出的宽高一样，则做特殊处理
    unscaled = (srcW == dstW && srcH == dstH);
    //如果是JPEG标准（Y取值0-255），则需要设置这两项
    c->srcRange |= handle_jpeg(&c->srcFormat);
    c->dstRange |= handle_jpeg(&c->dstFormat);

    if(srcFormat!=c->srcFormat || dstFormat!=c->dstFormat)
        av_log(c, AV_LOG_WARNING, "deprecated pixel format used, make sure you did set range correctly\n");
    //设置Colorspace
    if (!c->contrast && !c->saturation && !c->dstFormatBpp)
        sws_setColorspaceDetails(c, ff_yuv2rgb_coeffs[SWS_CS_DEFAULT], c->srcRange,
                                 ff_yuv2rgb_coeffs[SWS_CS_DEFAULT],
                                 c->dstRange, 0, 1 << 16, 1 << 16);

    handle_formats(c);
    srcFormat = c->srcFormat;
    dstFormat = c->dstFormat;
    desc_src = av_pix_fmt_desc_get(srcFormat);
    desc_dst = av_pix_fmt_desc_get(dstFormat);
    //转换大小端？
    if (!(unscaled && sws_isSupportedEndiannessConversion(srcFormat) &&
          av_pix_fmt_swap_endianness(srcFormat) == dstFormat)) {
    //检查输入格式是否支持
    if (!sws_isSupportedInput(srcFormat)) {
        av_log(c, AV_LOG_ERROR, "%s is not supported as input pixel format\n",
               av_get_pix_fmt_name(srcFormat));
        return AVERROR(EINVAL);
    }
    //检查输出格式是否支持
    if (!sws_isSupportedOutput(dstFormat)) {
        av_log(c, AV_LOG_ERROR, "%s is not supported as output pixel format\n",
               av_get_pix_fmt_name(dstFormat));
        return AVERROR(EINVAL);
    }
    }
    //检查拉伸的方法
    i = flags & (SWS_POINT         |
                 SWS_AREA          |
                 SWS_BILINEAR      |
                 SWS_FAST_BILINEAR |
                 SWS_BICUBIC       |
                 SWS_X             |
                 SWS_GAUSS         |
                 SWS_LANCZOS       |
                 SWS_SINC          |
                 SWS_SPLINE        |
                 SWS_BICUBLIN);

    /* provide a default scaler if not set by caller */
    //如果没有指定，就使用默认的
    if (!i) {
        if (dstW < srcW && dstH < srcH)
```

```
            flags |= SWS_BICUBIC;
        else if (dstW > srcW && dstH > srcH)
            flags |= SWS_BICUBIC;
        else
            flags |= SWS_BICUBIC;
        c->flags = flags;
    } else if (i & (i - 1)) {
        av_log(c, AV_LOG_ERROR,
               "Exactly one scaler algorithm must be chosen, got %X\n", i);
        return AVERROR(EINVAL);
    }
    /* sanity check */
    //检查宽高参数
    if (srcW < 1 || srcH < 1 || dstW < 1 || dstH < 1) {
        /* FIXME check if these are enough and try to lower them after
         * fixing the relevant parts of the code */
        av_log(c, AV_LOG_ERROR, "%dx%d -> %dx%d is invalid scaling dimension\n",
               srcW, srcH, dstW, dstH);
        return AVERROR(EINVAL);
    }

    if (!dstFilter)
        dstFilter = &dummyFilter;
    if (!srcFilter)
        srcFilter = &dummyFilter;

    c->lumXInc      = (((int64_t)srcW << 16) + (dstW >> 1)) / dstW;
    c->lumYInc      = (((int64_t)srcH << 16) + (dstH >> 1)) / dstH;
    c->dstFormatBpp = av_get_bits_per_pixel(desc_dst);
    c->srcFormatBpp = av_get_bits_per_pixel(desc_src);
    c->vRounder     = 4 * 0x0001000100010001ULL;

    usesVFilter = (srcFilter->lumV && srcFilter->lumV->length > 1) ||
                  (srcFilter->chrV && srcFilter->chrV->length > 1) ||
                  (dstFilter->lumV && dstFilter->lumV->length > 1) ||
                  (dstFilter->chrV && dstFilter->chrV->length > 1);
    usesHFilter = (srcFilter->lumH && srcFilter->lumH->length > 1) ||
                  (srcFilter->chrH && srcFilter->chrH->length > 1) ||
                  (dstFilter->lumH && dstFilter->lumH->length > 1) ||
                  (dstFilter->chrH && dstFilter->chrH->length > 1);

    av_pix_fmt_get_chroma_sub_sample(srcFormat, &c->chrSrcHSubSample, &c->chrSrcVSubSample);
    av_pix_fmt_get_chroma_sub_sample(dstFormat, &c->chrDstHSubSample, &c->chrDstVSubSample);

    if (isAnyRGB(dstFormat) && !(flags&SWS_FULL_CHR_H_INT)) {
        if (dstW&1) {
            av_log(c, AV_LOG_DEBUG, "Forcing full internal H chroma due to odd output size\n");
            flags |= SWS_FULL_CHR_H_INT;
            c->flags = flags;
        }

        if (   c->chrSrcHSubSample == 0
            && c->chrSrcVSubSample == 0
            && c->dither != SWS_DITHER_BAYER //SWS_FULL_CHR_H_INT is currently not supported with SWS_DITHER_BAYER
            && !(c->flags & SWS_FAST_BILINEAR)
        ) {
            av_log(c, AV_LOG_DEBUG, "Forcing full internal H chroma due to input having non subsampled chroma\n");
            flags |= SWS_FULL_CHR_H_INT;
            c->flags = flags;
        }
    }

    if (c->dither == SWS_DITHER_AUTO) {
        if (flags & SWS_ERROR_DIFFUSION)
            c->dither = SWS_DITHER_ED;
    }

    if(dstFormat == AV_PIX_FMT_BGR4_BYTE ||
       dstFormat == AV_PIX_FMT_RGB4_BYTE ||
       dstFormat == AV_PIX_FMT_BGR8 ||
       dstFormat == AV_PIX_FMT_RGB8) {
        if (c->dither == SWS_DITHER_AUTO)
            c->dither = (flags & SWS_FULL_CHR_H_INT) ? SWS_DITHER_ED : SWS_DITHER_BAYER;
        if (!(flags & SWS_FULL_CHR_H_INT)) {
            if (c->dither == SWS_DITHER_ED || c->dither == SWS_DITHER_A_DITHER || c->dither == SWS_DITHER_X_DITHER) {
                av_log(c, AV_LOG_DEBUG,
                    "Desired dithering only supported in full chroma interpolation for destination format '%s'\n",
                    av_get_pix_fmt_name(dstFormat));
                flags   |= SWS_FULL_CHR_H_INT;
                c->flags = flags;
            }
        }
        if (flags & SWS_FULL_CHR_H_INT) {
            if (c->dither == SWS_DITHER_BAYER) {
                av_log(c, AV_LOG_DEBUG,
                    "Ordered dither is not supported in full chroma interpolation for destination format '%s'\n",
                    av_get_pix_fmt_name(dstFormat));
                c->dither = SWS_DITHER_ED;
            }
        }
    }
    if (isPlanarRGB(dstFormat)) {
```

```
    if (isPlanarRGB(dstFormat)) {
        if (!(flags & SWS_FULL_CHR_H_INT)) {
            av_log(c, AV_LOG_DEBUG,
                   "%s output is not supported with half chroma resolution, switching to full\n",
                   av_get_pix_fmt_name(dstFormat));
            flags   |= SWS_FULL_CHR_H_INT;
            c->flags = flags;
        }
    }

    /* reuse chroma for 2 pixels RGB/BGR unless user wants full
     * chroma interpolation */
    if (flags & SWS_FULL_CHR_H_INT &&
        isAnyRGB(dstFormat)        &&
        !isPlanarRGB(dstFormat)    &&
        dstFormat != AV_PIX_FMT_RGBA  &&
        dstFormat != AV_PIX_FMT_ARGB  &&
        dstFormat != AV_PIX_FMT_BGRA  &&
        dstFormat != AV_PIX_FMT_ABGR  &&
        dstFormat != AV_PIX_FMT_RGB24 &&
        dstFormat != AV_PIX_FMT_BGR24 &&
        dstFormat != AV_PIX_FMT_BGR4_BYTE &&
        dstFormat != AV_PIX_FMT_RGB4_BYTE &&
        dstFormat != AV_PIX_FMT_BGR8 &&
        dstFormat != AV_PIX_FMT_RGB8
    ) {
        av_log(c, AV_LOG_WARNING,
               "full chroma interpolation for destination format '%s' not yet implemented\n",
               av_get_pix_fmt_name(dstFormat));
        flags   &= ~SWS_FULL_CHR_H_INT;
        c->flags = flags;
    }
    if (isAnyRGB(dstFormat) && !(flags & SWS_FULL_CHR_H_INT))
        c->chrDstHSubSample = 1;

    // drop some chroma lines if the user wants it
    c->vChrDrop          = (flags & SWS_SRC_V_CHR_DROP_MASK) >>
                           SWS_SRC_V_CHR_DROP_SHIFT;
    c->chrSrcVSubSample += c->vChrDrop;

    /* drop every other pixel for chroma calculation unless user
     * wants full chroma */
    if (isAnyRGB(srcFormat) && !(flags & SWS_FULL_CHR_H_INP)   &&
        srcFormat != AV_PIX_FMT_RGB8 && srcFormat != AV_PIX_FMT_BGR8 &&
        srcFormat != AV_PIX_FMT_RGB4 && srcFormat != AV_PIX_FMT_BGR4 &&
        srcFormat != AV_PIX_FMT_RGB4_BYTE && srcFormat != AV_PIX_FMT_BGR4_BYTE &&
        srcFormat != AV_PIX_FMT_GBRP9BE   && srcFormat != AV_PIX_FMT_GBRP9LE  &&
        srcFormat != AV_PIX_FMT_GBRP10BE  && srcFormat != AV_PIX_FMT_GBRP10LE &&
        srcFormat != AV_PIX_FMT_GBRP12BE  && srcFormat != AV_PIX_FMT_GBRP12LE &&
        srcFormat != AV_PIX_FMT_GBRP14BE  && srcFormat != AV_PIX_FMT_GBRP14LE &&
        srcFormat != AV_PIX_FMT_GBRP16BE  && srcFormat != AV_PIX_FMT_GBRP16LE &&
        ((dstW >> c->chrDstHSubSample) <= (srcW >> 1) ||
         (flags & SWS_FAST_BILINEAR)))
        c->chrSrcHSubSample = 1;

    // Note the FF_CEIL_RSHIFT is so that we always round toward +inf.
    c->chrSrcW = FF_CEIL_RSHIFT(srcW, c->chrSrcHSubSample);
    c->chrSrcH = FF_CEIL_RSHIFT(srcH, c->chrSrcVSubSample);
    c->chrDstW = FF_CEIL_RSHIFT(dstW, c->chrDstHSubSample);
    c->chrDstH = FF_CEIL_RSHIFT(dstH, c->chrDstVSubSample);

    FF_ALLOC_OR_GOTO(c, c->formatConvBuffer, FFALIGN(srcW*2+78, 16) * 2, fail);

    c->srcBpc = 1 + desc_src->comp[0].depth_minus1;
    if (c->srcBpc < 8)
        c->srcBpc = 8;
    c->dstBpc = 1 + desc_dst->comp[0].depth_minus1;
    if (c->dstBpc < 8)
        c->dstBpc = 8;
    if (isAnyRGB(srcFormat) || srcFormat == AV_PIX_FMT_PAL8)
        c->srcBpc = 16;
    if (c->dstBpc == 16)
        dst_stride <<= 1;

    if (INLINE_MMXEXT(cpu_flags) && c->srcBpc == 8 && c->dstBpc <= 14) {
        c->canMMXEXTBeUsed = dstW >= srcW && (dstW & 31) == 0 &&
                             c->chrDstW >= c->chrSrcW &&
                             (srcW & 15) == 0;
        if (!c->canMMXEXTBeUsed && dstW >= srcW && c->chrDstW >= c->chrSrcW && (srcW & 15) == 0

            && (flags & SWS_FAST_BILINEAR)) {
            if (flags & SWS_PRINT_INFO)
                av_log(c, AV_LOG_INFO,
                       "output width is not a multiple of 32 -> no MMXEXT scaler\n");
        }
        if (usesHFilter || isNBPS(c->srcFormat) || is16BPS(c->srcFormat) || isAnyRGB(c->srcFormat))
            c->canMMXEXTBeUsed = 0;
    } else
        c->canMMXEXTBeUsed = 0;

    c->chrXInc = (((int64_t)c->chrSrcW << 16) + (c->chrDstW >> 1)) / c->chrDstW;
    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->canMMXEXTBeUsed) {
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_MMXEXT_INLINE
310. // can't downscale !!!
311.         if (c->canMMXEXTBeUsed && (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. #ifdef 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);
```

```
            FF_ALLOCZ_OR_GOTO(c, c->hChrFilter,    (c->chrDstW     / 4 + 8) * sizeof(int16_t), fail);
            FF_ALLOCZ_OR_GOTO(c, c->hLumFilterPos, (dstW       / 2 / 8 + 8) * sizeof(int32_t), fail);
            FF_ALLOCZ_OR_GOTO(c, c->hChrFilterPos, (c->chrDstW / 2 / 4 + 8) * sizeof(int32_t), fail);

            ff_init_hscaler_mmxext(      dstW, c->lumXInc, c->lumMmxextFilterCode,
                                   c->hLumFilter, (uint32_t*)c->hLumFilterPos, 8);
            ff_init_hscaler_mmxext(c->chrDstW, c->chrXInc, c->chrMmxextFilterCode,
                                   c->hChrFilter, (uint32_t*)c->hChrFilterPos, 4);

#if USE_MMAP
            if (   mprotect(c->lumMmxextFilterCode, c->lumMmxextFilterCodeSize, PROT_EXEC | PROT_READ) == -1
                || mprotect(c->chrMmxextFilterCode, c->chrMmxextFilterCodeSize, PROT_EXEC | PROT_READ) == -1) {
                av_log(c, AV_LOG_ERROR, "mprotect failed, cannot use fast bilinear scaler\n");
                goto fail;
            }
#endif
        } else
#endif /* HAVE_MMXEXT_INLINE */
        {
            const int filterAlign = X86_MMX(cpu_flags)     ? 4 :
                                    PPC_ALTIVEC(cpu_flags) ? 8 : 1;

            if ((ret = initFilter(&c->hLumFilter, &c->hLumFilterPos,
                                  &c->hLumFilterSize, c->lumXInc,
                                  srcW, dstW, filterAlign, 1 << 14,
                                  (flags & SWS_BICUBLIN) ? (flags | SWS_BICUBIC) : flags,
                                  cpu_flags, srcFilter->lumH, dstFilter->lumH,
                                  c->param,
                                  get_local_pos(c, 0, 0, 0),
                                  get_local_pos(c, 0, 0, 0))) < 0)
                goto fail;
            if ((ret = initFilter(&c->hChrFilter, &c->hChrFilterPos,
                                  &c->hChrFilterSize, c->chrXInc,
                                  c->chrSrcW, c->chrDstW, filterAlign, 1 << 14,
                                  (flags & SWS_BICUBLIN) ? (flags | SWS_BILINEAR) : flags,
                                  cpu_flags, srcFilter->chrH, dstFilter->chrH,
                                  c->param,
                                  get_local_pos(c, c->chrSrcHSubSample, c->src_h_chr_pos, 0),
                                  get_local_pos(c, c->chrDstHSubSample, c->dst_h_chr_pos, 0))) < 0)
                goto fail;
        }
    } // initialize horizontal stuff

    /* precalculate vertical scaler filter coefficients */
    {
        const int filterAlign = X86_MMX(cpu_flags)     ? 2 :
                                PPC_ALTIVEC(cpu_flags) ? 8 : 1;

        if ((ret = initFilter(&c->vLumFilter, &c->vLumFilterPos, &c->vLumFilterSize,
                              c->lumYInc, srcH, dstH, filterAlign, (1 << 12),
                              (flags & SWS_BICUBLIN) ? (flags | SWS_BICUBIC) : flags,
                              cpu_flags, srcFilter->lumV, dstFilter->lumV,
                              c->param,
                              get_local_pos(c, 0, 0, 1),
                              get_local_pos(c, 0, 0, 1))) < 0)
            goto fail;
        if ((ret = initFilter(&c->vChrFilter, &c->vChrFilterPos, &c->vChrFilterSize,
                              c->chrYInc, c->chrSrcH, c->chrDstH,
                              filterAlign, (1 << 12),
                              (flags & SWS_BICUBLIN) ? (flags | SWS_BILINEAR) : flags,
                              cpu_flags, srcFilter->chrV, dstFilter->chrV,
                              c->param,
                              get_local_pos(c, c->chrSrcVSubSample, c->src_v_chr_pos, 1),
                              get_local_pos(c, c->chrDstVSubSample, c->dst_v_chr_pos, 1))) < 0)

            goto fail;

#if HAVE_ALTIVEC
        FF_ALLOC_OR_GOTO(c, c->vYCoeffsBank, sizeof(vector signed short) * c->vLumFilterSize * c->dstH,    fail);
        FF_ALLOC_OR_GOTO(c, c->vCCoeffsBank, sizeof(vector signed short) * c->vChrFilterSize * c->chrDstH, fail);

        for (i = 0; i < c->vLumFilterSize * c->dstH; i++) {
            int j;
            short *p = (short *)&c->vYCoeffsBank[i];
            for (j = 0; j < 8; j++)
                p[j] = c->vLumFilter[i];
        }

        for (i = 0; i < c->vChrFilterSize * c->chrDstH; i++) {
            int j;
            short *p = (short *)&c->vCCoeffsBank[i];
            for (j = 0; j < 8; j++)
                p[j] = c->vChrFilter[i];
        }
#endif
    }

    // calculate buffer sizes so that they won't run out while handling these damn slices
    c->vLumBufSize = c->vLumFilterSize;
    c->vChrBufSize = c->vChrFilterSize;
    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%s ",
516.                scaler,
517.                av_get_pix_fmt_name(srcFormat),
518. #ifdef DITHER1XBPP
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_MMXEXT(cpu_flags))
529.             cpucaps = "MMXEXT";
530.         else if (INLINE_AMD3DNOW(cpu_flags))
531.             cpucaps = "3DNOW";
532.         else if (INLINE_MMX(cpu_flags))
```

```
            cpucaps = "MMX";
        else if (PPC_ALTIVEC(cpu_flags))
            cpucaps = "AltiVec";
        else
            cpucaps = "C";

        av_log(c, AV_LOG_INFO, "using %s\n", cpucaps);

        av_log(c, AV_LOG_VERBOSE, "%dx%d -> %dx%d\n", srcW, srcH, dstW, dstH);
        av_log(c, AV_LOG_DEBUG,
               "lum srcW=%d srcH=%d dstW=%d dstH=%d xInc=%d yInc=%d\n",
               c->srcW, c->srcH, c->dstW, c->dstH, c->lumXInc, c->lumYInc);
        av_log(c, AV_LOG_DEBUG,
               "chr srcW=%d srcH=%d dstW=%d dstH=%d xInc=%d yInc=%d\n",
               c->chrSrcW, c->chrSrcH, c->chrDstW, c->chrDstH,
               c->chrXInc, c->chrYInc);
    }

    /* unscaled special cases */
    //不拉伸的情况
    if (unscaled && !usesHFilter && !usesVFilter &&
        (c->srcRange == c->dstRange || isAnyRGB(dstFormat))) {
        //不许拉伸的情况下，初始化相应的函数
        ff_get_unscaled_swscale(c);

        if (c->swscale) {
            if (flags & SWS_PRINT_INFO)
                av_log(c, AV_LOG_INFO,
                       "using unscaled %s -> %s special converter\n",
                       av_get_pix_fmt_name(srcFormat), av_get_pix_fmt_name(dstFormat));
            return 0;
        }
    }
    //关键：设置SwsContext中的swscale()指针
    c->swscale = ff_getSwsFunc(c);
    return 0;
fail: // FIXME replace things by appropriate error codes
    if (ret == RETCODE_USE_CASCADE)  {
        int tmpW = sqrt(srcW * (int64_t)dstW);
        int tmpH = sqrt(srcH * (int64_t)dstH);
        enum AVPixelFormat tmpFormat = AV_PIX_FMT_YUV420P;

        if (srcW*(int64_t)srcH <= 4LL*dstW*dstH)
            return AVERROR(EINVAL);

        ret = av_image_alloc(c->cascaded_tmp, c->cascaded_tmpStride,
                             tmpW, tmpH, tmpFormat, 64);
        if (ret < 0)
            return ret;

        c->cascaded_context[0] = sws_getContext(srcW, srcH, srcFormat,
                                                tmpW, tmpH, tmpFormat,
                                                flags, srcFilter, NULL, c->param);
        if (!c->cascaded_context[0])
            return -1;

        c->cascaded_context[1] = sws_getContext(tmpW, tmpH, tmpFormat,
                                                dstW, dstH, dstFormat,
                                                flags, NULL, dstFilter, c->param);
        if (!c->cascaded_context[1])
            return -1;
        return 0;
    }
    return -1;
}
```

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。

```cpp
av_cold void sws_rgb2rgb_init(void){
    rgb2rgb_init_c();
    if (ARCH_X86)
        rgb2rgb_init_x86();
}
```

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
static av_cold void rgb2rgb_init_c(void)
{
    rgb15to16          = rgb15to16_c;
    rgb15tobgr24       = rgb15tobgr24_c;
    rgb15to32          = rgb15to32_c;
    rgb16tobgr24       = rgb16tobgr24_c;
    rgb16to32          = rgb16to32_c;
    rgb16to15          = rgb16to15_c;
    rgb24tobgr16       = rgb24tobgr16_c;
    rgb24tobgr15       = rgb24tobgr15_c;
    rgb24tobgr32       = rgb24tobgr32_c;
    rgb32to16          = rgb32to16_c;
    rgb32to15          = rgb32to15_c;
    rgb32tobgr24       = rgb32tobgr24_c;
    rgb24to15          = rgb24to15_c;
    rgb24to16          = rgb24to16_c;
    rgb24tobgr24       = rgb24tobgr24_c;
    shuffle_bytes_2103 = shuffle_bytes_2103_c;
    rgb32tobgr16       = rgb32tobgr16_c;
    rgb32tobgr15       = rgb32tobgr15_c;
    yv12toyuy2         = yv12toyuy2_c;
    yv12touyvy         = yv12touyvy_c;
    yuv422ptoyuy2      = yuv422ptoyuy2_c;
    yuv422ptouyvy      = yuv422ptouyvy_c;
    yuy2toyv12         = yuy2toyv12_c;
    planar2x           = planar2x_c;
    ff_rgb24toyv12     = ff_rgb24toyv12_c;
    interleaveBytes    = interleaveBytes_c;
    deinterleaveBytes  = deinterleaveBytes_c;
    vu9_to_vu12        = vu9_to_vu12_c;
    yvu9_to_yuy2       = yvu9_to_yuy2_c;

    uyvytoyuv420       = uyvytoyuv420_c;
    uyvytoyuv422       = uyvytoyuv422_c;
    yuyvtoyuv420       = yuyvtoyuv420_c;
    yuyvtoyuv422       = yuyvtoyuv422_c;
}
```

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。

rgb24tobgr24_c()

rgb24tobgr24_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
static inline void rgb24tobgr24_c(const uint8_t *src, uint8_t *dst, int src_size)
{
    unsigned i;

    for (i = 0; i < src_size; i += 3) {
        register uint8_t x = src[i + 2];
        dst[i + 1]         = src[i + 1];
        dst[i + 2]         = src[i + 0];
        dst[i + 0]         = x;
    }
}
```

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
static inline void rgb24to16_c(const uint8_t *src, uint8_t *dst, int src_size)
{
    uint16_t *d        = (uint16_t *)dst;
    const uint8_t *s   = src;
    const uint8_t *end = s + src_size;

    while (s < end) {
        const int r = *s++;
        const int g = *s++;
        const int b = *s++;
        *d++        = (b >> 3) | ((g & 0xFC) << 3) | ((r & 0xF8) << 8);
    }
}
```

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
static void yuyvtoyuv422_c(uint8_t *ydst, uint8_t *udst, uint8_t *vdst,
                           const uint8_t *src, int width, int height,
                           int lumStride, int chromStride, int srcStride)
{
    int y;
    const int chromWidth = FF_CEIL_RSHIFT(width, 1);

    for (y = 0; y < height; y++) {
        extract_even_c(src, ydst, width);
        extract_odd2_c(src, udst, vdst, chromWidth);

        src  += srcStride;
        ydst += lumStride;
        udst += chromStride;
        vdst += chromStride;
    }
}
```

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
static void extract_even_c(const uint8_t *src, uint8_t *dst, int count)
{
    dst   +=  count;
    src   +=  count * 2;
    count  = -count;
    while (count < 0) {
        dst[count] = src[2 * count];
        count++;
    }
}
```

extract_odd2_c()定义如下所示。

```cpp
static void extract_even2_c(const uint8_t *src, uint8_t *dst0, uint8_t *dst1,
                            int count)
{
    dst0  +=  count;
    dst1  +=  count;
    src   +=  count * 4;
    count  = -count;
    while (count < 0) {
        dst0[count] = src[4 * count + 0];
        dst1[count] = src[4 * count + 2];
        count++;
    }
}
```

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
av_cold void rgb2rgb_init_x86(void)
{
#if HAVE_INLINE_ASM
    int cpu_flags = av_get_cpu_flags();

    if (INLINE_MMX(cpu_flags))
        rgb2rgb_init_mmx();
    if (INLINE_AMD3DNOW(cpu_flags))
        rgb2rgb_init_3dnow();
    if (INLINE_MMXEXT(cpu_flags))
        rgb2rgb_init_mmxext();
    if (INLINE_SSE2(cpu_flags))
        rgb2rgb_init_sse2();
    if (INLINE_AVX(cpu_flags))
        rgb2rgb_init_avx();
#endif /* HAVE_INLINE_ASM */
}
```

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。

```cpp
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：

```cpp
/**
 * @param dstRange flag indicating the while-black range of the output (1=jpeg / 0=mpeg)
 * @param srcRange flag indicating the while-black range of the input (1=jpeg / 0=mpeg)
 * @param table the yuv2rgb coefficients describing the output yuv space, normally ff_yuv2rgb_coeffs[x]
 * @param inv_table the yuv2rgb coefficients describing the input yuv space, normally ff_yuv2rgb_coeffs[x]
 * @param brightness 16.16 fixed point brightness correction
 * @param contrast 16.16 fixed point contrast correction
 * @param saturation 16.16 fixed point saturation correction
 * @return -1 if not supported
 */
int sws_setColorspaceDetails(struct SwsContext *c, const int inv_table[4],
                             int srcRange, const int table[4], int dstRange,
                             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。

```cpp
int sws_setColorspaceDetails(struct SwsContext *c, const int inv_table[4],
                             int srcRange, const int table[4], int dstRange,
                             int brightness, int contrast, int saturation)
{
    const AVPixFmtDescriptor *desc_dst;
    const AVPixFmtDescriptor *desc_src;
    int need_reinit = 0;
    memmove(c->srcColorspaceTable, inv_table, sizeof(int) * 4);
    memmove(c->dstColorspaceTable, table, sizeof(int) * 4);

    handle_formats(c);
    desc_dst = av_pix_fmt_desc_get(c->dstFormat);
    desc_src = av_pix_fmt_desc_get(c->srcFormat);

    if(!isYUV(c->dstFormat) && !isGray(c->dstFormat))
        dstRange = 0;
    if(!isYUV(c->srcFormat) && !isGray(c->srcFormat))
        srcRange = 0;

    c->brightness = brightness;
    c->contrast   = contrast;
    c->saturation = saturation;
    if (c->srcRange != srcRange || c->dstRange != dstRange)
        need_reinit = 1;
    c->srcRange   = srcRange;
    c->dstRange   = dstRange;

    //The srcBpc check is possibly wrong but we seem to lack a definitive reference to test this
    //and what we have in ticket 2939 looks better with this check
    if (need_reinit && (c->srcBpc == 8 || !isYUV(c->srcFormat)))
        ff_sws_init_range_convert(c);

    if ((isYUV(c->dstFormat) || isGray(c->dstFormat)) && (isYUV(c->srcFormat) || isGray(c->srcFormat)))
        return -1;

    c->dstFormatBpp = av_get_bits_per_pixel(desc_dst);
    c->srcFormatBpp = av_get_bits_per_pixel(desc_src);

    if (!isYUV(c->dstFormat) && !isGray(c->dstFormat)) {
        ff_yuv2rgb_c_init_tables(c, inv_table, srcRange, brightness,
                                 contrast, saturation);
        // FIXME factorize

        if (ARCH_PPC)
            ff_yuv2rgb_init_tables_ppc(c, inv_table, brightness,
                                       contrast, saturation);
    }

    fill_rgb2yuv_table(c, table, dstRange);

    return 0;
}
```

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，并且在最后调用了一个函数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。

```cpp
/**
 * Return a pointer to yuv<->rgb coefficients for the given colorspace
 * suitable for sws_setColorspaceDetails().
 *
 * @param colorspace One of the SWS_CS_* macros. If invalid,
 * SWS_CS_DEFAULT is used.
 */
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。

```cpp
#define SWS_CS_ITU709         1
#define SWS_CS_FCC            4
#define SWS_CS_ITU601         5
#define SWS_CS_ITU624         5
#define SWS_CS_SMPTE170M      5
#define SWS_CS_SMPTE240M      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
const int *sws_getCoefficients(int colorspace)
{
    if (colorspace > 7 || colorspace < 0)
        colorspace = SWS_CS_DEFAULT;
    return ff_yuv2rgb_coeffs[colorspace];
}
```

可以看出它返回了一个名称为ff_yuv2rgb_coeffs的数组中的一个元素，该数组的定义如下所示。

```cpp
const int32_t ff_yuv2rgb_coeffs[8][4] = {
    { 117504, 138453, 13954, 34903 }, /* no sequence_display_extension */
    { 117504, 138453, 13954, 34903 }, /* ITU-R Rec. 709 (1990) */
    { 104597, 132201, 25675, 53279 }, /* unspecified */
    { 104597, 132201, 25675, 53279 }, /* reserved */
    { 104448, 132798, 24759, 53109 }, /* FCC */
    { 104597, 132201, 25675, 53279 }, /* ITU-R Rec. 624-4 System B, G */
    { 104597, 132201, 25675, 53279 }, /* SMPTE 170M */
    { 117579, 136230, 16907, 35559 }  /* SMPTE 240M (1987) */
};
```

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
i = flags & (SWS_POINT         |
             SWS_AREA          |
             SWS_BILINEAR      |
             SWS_FAST_BILINEAR |
             SWS_BICUBIC       |
             SWS_X             |
             SWS_GAUSS         |
             SWS_LANCZOS       |
             SWS_SINC          |
             SWS_SPLINE        |
             SWS_BICUBLIN);

/* provide a default scaler if not set by caller */
if (!i) {
    if (dstW < srcW && dstH < srcH)
        flags |= SWS_BICUBIC;
    else if (dstW > srcW && dstH > srcH)
        flags |= SWS_BICUBIC;
    else
        flags |= SWS_BICUBIC;
    c->flags = flags;
} else if (i & (i - 1)) {
    av_log(c, AV_LOG_ERROR,
           "Exactly one scaler algorithm must be chosen, got %X\n", i);
    return AVERROR(EINVAL);
}
/* sanity check */
if (srcW < 1 || srcH < 1 || dstW < 1 || dstH < 1) {
    /* FIXME check if these are enough and try to lower them after
     * fixing the relevant parts of the code */
    av_log(c, AV_LOG_ERROR, "%dx%d -> %dx%d is invalid scaling dimension\n",
           srcW, srcH, dstW, dstH);
    return AVERROR(EINVAL);
}
```

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。

```cpp
if (flags & SWS_PRINT_INFO) {
        const char *scaler = NULL, *cpucaps;

        for (i = 0; i < FF_ARRAY_ELEMS(scale_algorithms); i++) {
            if (flags & scale_algorithms[i].flag) {
                scaler = scale_algorithms[i].description;
                break;
            }
        }
        if (!scaler)
            scaler =  "ehh flags invalid?!";
        av_log(c, AV_LOG_INFO, "%s scaler, from %s to %s%s ",
               scaler,
               av_get_pix_fmt_name(srcFormat),
#ifdef DITHER1XBPP
               dstFormat == AV_PIX_FMT_BGR555   || dstFormat == AV_PIX_FMT_BGR565   ||
               dstFormat == AV_PIX_FMT_RGB444BE || dstFormat == AV_PIX_FMT_RGB444LE ||
               dstFormat == AV_PIX_FMT_BGR444BE || dstFormat == AV_PIX_FMT_BGR444LE ?
                                                           "dithered " : "",
#else
               "",
#endif
               av_get_pix_fmt_name(dstFormat));

        if (INLINE_MMXEXT(cpu_flags))
            cpucaps = "MMXEXT";
        else if (INLINE_AMD3DNOW(cpu_flags))
            cpucaps = "3DNOW";
        else if (INLINE_MMX(cpu_flags))
            cpucaps = "MMX";
        else if (PPC_ALTIVEC(cpu_flags))
            cpucaps = "AltiVec";
        else
            cpucaps = "C";

        av_log(c, AV_LOG_INFO, "using %s\n", cpucaps);

        av_log(c, AV_LOG_VERBOSE, "%dx%d -> %dx%d\n", srcW, srcH, dstW, dstH);
        av_log(c, AV_LOG_DEBUG,
               "lum srcW=%d srcH=%d dstW=%d dstH=%d xInc=%d yInc=%d\n",
               c->srcW, c->srcH, c->dstW, c->dstH, c->lumXInc, c->lumYInc);
        av_log(c, AV_LOG_DEBUG,
               "chr srcW=%d srcH=%d dstW=%d dstH=%d xInc=%d yInc=%d\n",
               c->chrSrcW, c->chrSrcH, c->chrDstW, c->chrDstH,
               c->chrXInc, c->chrYInc);
    }
```

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。

```cpp
void ff_get_unscaled_swscale(SwsContext *c)
{
    const enum AVPixelFormat srcFormat = c->srcFormat;
    const enum AVPixelFormat dstFormat = c->dstFormat;
    const int flags = c->flags;
    const int dstH = c->dstH;
    int needsDither;

    needsDither = isAnyRGB(dstFormat) &&
            c->dstFormatBpp < 24 &&
           (c->dstFormatBpp < c->srcFormatBpp || (!isAnyRGB(srcFormat)));

    /* yv12_to_nv12 */
    if ((srcFormat == AV_PIX_FMT_YUV420P || srcFormat == AV_PIX_FMT_YUVA420P) &&
        (dstFormat == AV_PIX_FMT_NV12 || dstFormat == AV_PIX_FMT_NV21)) {
        c->swscale = planarToNv12Wrapper;
    }
    /* nv12_to_yv12 */
    if (dstFormat == AV_PIX_FMT_YUV420P &&
        (srcFormat == AV_PIX_FMT_NV12 || srcFormat == AV_PIX_FMT_NV21)) {
        c->swscale = nv12ToPlanarWrapper;
    }
    /* yuv2bgr */
    if ((srcFormat == AV_PIX_FMT_YUV420P || srcFormat == AV_PIX_FMT_YUV422P ||
         srcFormat == AV_PIX_FMT_YUVA420P) && isAnyRGB(dstFormat) &&
        !(flags & SWS_ACCURATE_RND) && (c->dither == SWS_DITHER_BAYER || c->dither == SWS_DITHER_AUTO) && !(dstH & 1)) {
        c->swscale = ff_yuv2rgb_get_func_ptr(c);
    }

    if (srcFormat == AV_PIX_FMT_YUV410P && !(dstH & 3) &&
        (dstFormat == AV_PIX_FMT_YUV420P || dstFormat == AV_PIX_FMT_YUVA420P) &&
        !(flags & SWS_BITEXACT)) {
```

```
        c->swscale = yvu9ToYv12Wrapper;
    }

    /* bgr24toYV12 */
    if (srcFormat == AV_PIX_FMT_BGR24 &&
        (dstFormat == AV_PIX_FMT_YUV420P || dstFormat == AV_PIX_FMT_YUVA420P) &&
        !(flags & SWS_ACCURATE_RND))
        c->swscale = bgr24ToYv12Wrapper;

    /* RGB/BGR -> RGB/BGR (no dither needed forms) */
    if (isAnyRGB(srcFormat) && isAnyRGB(dstFormat) && findRgbConvFn(c)
        && (!needsDither || (c->flags&(SWS_FAST_BILINEAR|SWS_POINT))))
        c->swscale = rgbToRgbWrapper;

    if ((srcFormat == AV_PIX_FMT_GBRP && dstFormat == AV_PIX_FMT_GBRAP) ||
        (srcFormat == AV_PIX_FMT_GBRAP && dstFormat == AV_PIX_FMT_GBRP))
        c->swscale = planarRgbToplanarRgbWrapper;

#define isByteRGB(f) (            \
        f == AV_PIX_FMT_RGB32   || \
        f == AV_PIX_FMT_RGB32_1 || \
        f == AV_PIX_FMT_RGB24   || \
        f == AV_PIX_FMT_BGR32   || \
        f == AV_PIX_FMT_BGR32_1 || \
        f == AV_PIX_FMT_BGR24)

    if (srcFormat == AV_PIX_FMT_GBRP && isPlanar(srcFormat) && isByteRGB(dstFormat))
        c->swscale = planarRgbToRgbWrapper;

    if ((srcFormat == AV_PIX_FMT_RGB48LE  || srcFormat == AV_PIX_FMT_RGB48BE  ||
         srcFormat == AV_PIX_FMT_BGR48LE  || srcFormat == AV_PIX_FMT_BGR48BE  ||
         srcFormat == AV_PIX_FMT_RGBA64LE || srcFormat == AV_PIX_FMT_RGBA64BE ||
         srcFormat == AV_PIX_FMT_BGRA64LE || srcFormat == AV_PIX_FMT_BGRA64BE) &&
        (dstFormat == AV_PIX_FMT_GBRP9LE  || dstFormat == AV_PIX_FMT_GBRP9BE  ||
         dstFormat == AV_PIX_FMT_GBRP10LE || dstFormat == AV_PIX_FMT_GBRP10BE ||
         dstFormat == AV_PIX_FMT_GBRP12LE || dstFormat == AV_PIX_FMT_GBRP12BE ||
         dstFormat == AV_PIX_FMT_GBRP14LE || dstFormat == AV_PIX_FMT_GBRP14BE ||
         dstFormat == AV_PIX_FMT_GBRP16LE || dstFormat == AV_PIX_FMT_GBRP16BE ||
         dstFormat == AV_PIX_FMT_GBRAP16LE || dstFormat == AV_PIX_FMT_GBRAP16BE ))
        c->swscale = Rgb16ToPlanarRgb16Wrapper;

    if ((srcFormat == AV_PIX_FMT_GBRP9LE  || srcFormat == AV_PIX_FMT_GBRP9BE  ||
         srcFormat == AV_PIX_FMT_GBRP16LE || srcFormat == AV_PIX_FMT_GBRP16BE ||
         srcFormat == AV_PIX_FMT_GBRP10LE || srcFormat == AV_PIX_FMT_GBRP10BE ||
         srcFormat == AV_PIX_FMT_GBRP12LE || srcFormat == AV_PIX_FMT_GBRP12BE ||
         srcFormat == AV_PIX_FMT_GBRP14LE || srcFormat == AV_PIX_FMT_GBRP14BE ||
         srcFormat == AV_PIX_FMT_GBRAP16LE || srcFormat == AV_PIX_FMT_GBRAP16BE) &&
        (dstFormat == AV_PIX_FMT_RGB48LE  || dstFormat == AV_PIX_FMT_RGB48BE  ||
         dstFormat == AV_PIX_FMT_BGR48LE  || dstFormat == AV_PIX_FMT_BGR48BE  ||
         dstFormat == AV_PIX_FMT_RGBA64LE || dstFormat == AV_PIX_FMT_RGBA64BE ||
         dstFormat == AV_PIX_FMT_BGRA64LE || dstFormat == AV_PIX_FMT_BGRA64BE))
        c->swscale = planarRgb16ToRgb16Wrapper;

    if (av_pix_fmt_desc_get(srcFormat)->comp[0].depth_minus1 == 7 &&
        isPackedRGB(srcFormat) && dstFormat == AV_PIX_FMT_GBRP)
        c->swscale = rgbToPlanarRgbWrapper;

    if (isBayer(srcFormat)) {
        if (dstFormat == AV_PIX_FMT_RGB24)
            c->swscale = bayer_to_rgb24_wrapper;
        else if (dstFormat == AV_PIX_FMT_YUV420P)
            c->swscale = bayer_to_yv12_wrapper;
        else if (!isBayer(dstFormat)) {
            av_log(c, AV_LOG_ERROR, "unsupported bayer conversion\n");
            av_assert0(0);
        }
    }

    /* bswap 16 bits per pixel/component packed formats */
    if (IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_BAYER_BGGR16) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_BAYER_RGGB16) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_BAYER_GBRG16) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_BAYER_GRBG16) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_BGR444) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_BGR48)  ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_BGRA64) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_BGR555) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_BGR565) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_BGRA64) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_GRAY16) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YA16)   ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_GBRP9)  ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_GBRP10) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_GBRP12) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_GBRP14) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_GBRP16) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_GBRAP16) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_RGB444) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_RGB48)  ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_RGBA64) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_RGB555) ||
```

```
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_RGB565) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_RGBA64) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_XYZ12)  ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV420P9)  ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV420P10) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV420P12) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV420P14) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV420P16) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV422P9)  ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV422P10) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV422P12) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV422P14) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV422P16) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV444P9)  ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV444P10) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV444P12) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV444P14) ||
        IS_DIFFERENT_ENDIANESS(srcFormat, dstFormat, AV_PIX_FMT_YUV444P16))
        c->swscale = packed_16bpc_bswap;

    if (usePal(srcFormat) && isByteRGB(dstFormat))
        c->swscale = palToRgbWrapper;

    if (srcFormat == AV_PIX_FMT_YUV422P) {
        if (dstFormat == AV_PIX_FMT_YUYV422)
            c->swscale = yuv422pToYuy2Wrapper;
        else if (dstFormat == AV_PIX_FMT_UYVY422)
            c->swscale = yuv422pToUyvyWrapper;
    }

    /* LQ converters if -sws 0 or -sws 4*/
    if (c->flags&(SWS_FAST_BILINEAR|SWS_POINT)) {
        /* yv12_to_yuy2 */
        if (srcFormat == AV_PIX_FMT_YUV420P || srcFormat == AV_PIX_FMT_YUVA420P) {
            if (dstFormat == AV_PIX_FMT_YUYV422)
                c->swscale = planarToYuy2Wrapper;
            else if (dstFormat == AV_PIX_FMT_UYVY422)
                c->swscale = planarToUyvyWrapper;
        }
    }
    if (srcFormat == AV_PIX_FMT_YUYV422 &&
       (dstFormat == AV_PIX_FMT_YUV420P || dstFormat == AV_PIX_FMT_YUVA420P))
        c->swscale = yuyvToYuv420Wrapper;
    if (srcFormat == AV_PIX_FMT_UYVY422 &&
       (dstFormat == AV_PIX_FMT_YUV420P || dstFormat == AV_PIX_FMT_YUVA420P))
        c->swscale = uyvyToYuv420Wrapper;
    if (srcFormat == AV_PIX_FMT_YUYV422 && dstFormat == AV_PIX_FMT_YUV422P)
        c->swscale = yuyvToYuv422Wrapper;
    if (srcFormat == AV_PIX_FMT_UYVY422 && dstFormat == AV_PIX_FMT_YUV422P)
        c->swscale = uyvyToYuv422Wrapper;

#define isPlanarGray(x) (isGray(x) && (x) != AV_PIX_FMT_YA8 && (x) != AV_PIX_FMT_YA16LE && (x) != AV_PIX_FMT_YA16BE)
    /* simple copy */
    if ( srcFormat == dstFormat ||
        (srcFormat == AV_PIX_FMT_YUVA420P && dstFormat == AV_PIX_FMT_YUV420P) ||
        (srcFormat == AV_PIX_FMT_YUV420P && dstFormat == AV_PIX_FMT_YUVA420P) ||
        (isPlanarYUV(srcFormat) && isPlanarGray(dstFormat)) ||
        (isPlanarYUV(dstFormat) && isPlanarGray(srcFormat)) ||
        (isPlanarGray(dstFormat) && isPlanarGray(srcFormat)) ||
        (isPlanarYUV(srcFormat) && isPlanarYUV(dstFormat) &&
         c->chrDstHSubSample == c->chrSrcHSubSample &&
         c->chrDstVSubSample == c->chrSrcVSubSample &&
         dstFormat != AV_PIX_FMT_NV12 && dstFormat != AV_PIX_FMT_NV21 &&
         srcFormat != AV_PIX_FMT_NV12 && srcFormat != AV_PIX_FMT_NV21))
    {
        if (isPacked(c->srcFormat))
            c->swscale = packedCopyWrapper;
        else /* Planar YUV or gray */
            c->swscale = planarCopyWrapper;
    }

    if (ARCH_PPC)
        ff_get_unscaled_swscale_ppc(c);
//     if (ARCH_ARM)
//         ff_get_unscaled_swscale_arm(c);
}
```

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
static int yuyvToYuv422Wrapper(SwsContext *c, const uint8_t *src[],
                               int srcStride[], int srcSliceY, int srcSliceH,
                               uint8_t *dstParam[], int dstStride[])
{
    uint8_t *ydst = dstParam[0] + dstStride[0] * srcSliceY;
    uint8_t *udst = dstParam[1] + dstStride[1] * srcSliceY;
    uint8_t *vdst = dstParam[2] + dstStride[2] * srcSliceY;

    yuyvtoyuv422(ydst, udst, vdst, src[0], c->srcW, srcSliceH, dstStride[0],
                 dstStride[1], srcStride[0]);

    return srcSliceH;
}
```

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
SwsFunc ff_getSwsFunc(SwsContext *c)
{
    sws_init_swscale(c);

    if (ARCH_PPC)
        ff_sws_init_swscale_ppc(c);
    if (ARCH_X86)
        ff_sws_init_swscale_x86(c);

    return swscale;
}
```

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
static av_cold void sws_init_swscale(SwsContext *c)
{
    enum AVPixelFormat srcFormat = c->srcFormat;

    ff_sws_init_output_funcs(c, &c->yuv2plane1, &c->yuv2planeX,
                             &c->yuv2nv12cX, &c->yuv2packed1,
                             &c->yuv2packed2, &c->yuv2packedX, &c->yuv2anyX);

    ff_sws_init_input_funcs(c);


    if (c->srcBpc == 8) {
        if (c->dstBpc <= 14) {
            c->hyScale = c->hcScale = hScale8To15_c;
            if (c->flags & SWS_FAST_BILINEAR) {
                c->hyscale_fast = ff_hyscale_fast_c;
                c->hcscale_fast = ff_hcscale_fast_c;
            }
        } else {
            c->hyScale = c->hcScale = hScale8To19_c;
        }
    } else {
        c->hyScale = c->hcScale = c->dstBpc > 14 ? hScale16To19_c
                                                 : hScale16To15_c;
    }

    ff_sws_init_range_convert(c);

    if (!(isGray(srcFormat) || isGray(c->dstFormat) ||
          srcFormat == AV_PIX_FMT_MONOBLACK || srcFormat == AV_PIX_FMT_MONOWHITE))
        c->needs_hcscale = 1;
}
```

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。

```cpp
av_cold void ff_sws_init_output_funcs(SwsContext *c,
                                      yuv2planar1_fn *yuv2plane1,
                                      yuv2planarX_fn *yuv2planeX,
                                      yuv2interleavedX_fn *yuv2nv12cX,
                                      yuv2packed1_fn *yuv2packed1,
                                      yuv2packed2_fn *yuv2packed2,
                                      yuv2packedX_fn *yuv2packedX,
                                      yuv2anyX_fn *yuv2anyX)
{
    enum AVPixelFormat dstFormat = c->dstFormat;
    const AVPixFmtDescriptor *desc = av_pix_fmt_desc_get(dstFormat);

    if (is16BPS(dstFormat)) {
        *yuv2planeX = isBE(dstFormat) ? yuv2planeX_16BE_c  : yuv2planeX_16LE_c;
        *yuv2plane1 = isBE(dstFormat) ? yuv2plane1_16BE_c  : yuv2plane1_16LE_c;
    } else if (is9_OR_10BPS(dstFormat)) {
        if (desc->comp[0].depth_minus1 == 8) {
            *yuv2planeX = isBE(dstFormat) ? yuv2planeX_9BE_c  : yuv2planeX_9LE_c;
            *yuv2plane1 = isBE(dstFormat) ? yuv2plane1_9BE_c  : yuv2plane1_9LE_c;
        } else if (desc->comp[0].depth_minus1 == 9) {
            *yuv2planeX = isBE(dstFormat) ? yuv2planeX_10BE_c  : yuv2planeX_10LE_c;
            *yuv2plane1 = isBE(dstFormat) ? yuv2plane1_10BE_c  : yuv2plane1_10LE_c;
        } else if (desc->comp[0].depth_minus1 == 11) {
            *yuv2planeX = isBE(dstFormat) ? yuv2planeX_12BE_c  : yuv2planeX_12LE_c;
            *yuv2plane1 = isBE(dstFormat) ? yuv2plane1_12BE_c  : yuv2plane1_12LE_c;
        } else if (desc->comp[0].depth_minus1 == 13) {
            *yuv2planeX = isBE(dstFormat) ? yuv2planeX_14BE_c  : yuv2planeX_14LE_c;
            *yuv2plane1 = isBE(dstFormat) ? yuv2plane1_14BE_c  : yuv2plane1_14LE_c;
        } else
            av_assert0(0);
    } else {
        *yuv2plane1 = yuv2plane1_8_c;
        *yuv2planeX = yuv2planeX_8_c;
        if (dstFormat == AV_PIX_FMT_NV12 || dstFormat == AV_PIX_FMT_NV21)
            *yuv2nv12cX = yuv2nv12cX_c;
    }

    if(c->flags & SWS_FULL_CHR_H_INT) {
        switch (dstFormat) {
            case AV_PIX_FMT_RGBA:
#if CONFIG_SMALL
                *yuv2packedX = yuv2rgba32_full_X_c;
                *yuv2packed2 = yuv2rgba32_full_2_c;
                *yuv2packed1 = yuv2rgba32_full_1_c;
#else
#if CONFIG_SWSCALE_ALPHA
                if (c->alpPixBuf) {
                    *yuv2packedX = yuv2rgba32_full_X_c;
                    *yuv2packed2 = yuv2rgba32_full_2_c;
                    *yuv2packed1 = yuv2rgba32_full_1_c;
                } else
#endif /* CONFIG_SWSCALE_ALPHA */
                {
                    *yuv2packedX = yuv2rgbx32_full_X_c;
                    *yuv2packed2 = yuv2rgbx32_full_2_c;
                    *yuv2packed1 = yuv2rgbx32_full_1_c;
                }
#endif /* !CONFIG_SMALL */
                break;
            case AV_PIX_FMT_ARGB:
#if CONFIG_SMALL
                *yuv2packedX = yuv2argb32_full_X_c;
                *yuv2packed2 = yuv2argb32_full_2_c;
                *yuv2packed1 = yuv2argb32_full_1_c;
#else
#if CONFIG_SWSCALE_ALPHA
                if (c->alpPixBuf) {
                    *yuv2packedX = yuv2argb32_full_X_c;
                    *yuv2packed2 = yuv2argb32_full_2_c;
                    *yuv2packed1 = yuv2argb32_full_1_c;
                } else
#endif /* CONFIG_SWSCALE_ALPHA */
                {
                    *yuv2packedX = yuv2xrgb32_full_X_c;
                    *yuv2packed2 = yuv2xrgb32_full_2_c;
                    *yuv2packed1 = yuv2xrgb32_full_1_c;
```

```
                    *yuv2packed1 = yuv2xrgb32_full_1_c;
                }
#endif /* !CONFIG_SMALL */
                break;
            case AV_PIX_FMT_BGRA:
#if CONFIG_SMALL
                *yuv2packedX = yuv2bgra32_full_X_c;
                *yuv2packed2 = yuv2bgra32_full_2_c;
                *yuv2packed1 = yuv2bgra32_full_1_c;
#else
#if CONFIG_SWSCALE_ALPHA
                if (c->alpPixBuf) {
                    *yuv2packedX = yuv2bgra32_full_X_c;
                    *yuv2packed2 = yuv2bgra32_full_2_c;
                    *yuv2packed1 = yuv2bgra32_full_1_c;
                } else
#endif /* CONFIG_SWSCALE_ALPHA */
                {
                    *yuv2packedX = yuv2bgrx32_full_X_c;
                    *yuv2packed2 = yuv2bgrx32_full_2_c;
                    *yuv2packed1 = yuv2bgrx32_full_1_c;
                }
#endif /* !CONFIG_SMALL */
                break;
            case AV_PIX_FMT_ABGR:
#if CONFIG_SMALL
                *yuv2packedX = yuv2abgr32_full_X_c;
                *yuv2packed2 = yuv2abgr32_full_2_c;
                *yuv2packed1 = yuv2abgr32_full_1_c;
#else
#if CONFIG_SWSCALE_ALPHA
                if (c->alpPixBuf) {
                    *yuv2packedX = yuv2abgr32_full_X_c;
                    *yuv2packed2 = yuv2abgr32_full_2_c;
                    *yuv2packed1 = yuv2abgr32_full_1_c;
                } else
#endif /* CONFIG_SWSCALE_ALPHA */
                {
                    *yuv2packedX = yuv2xbgr32_full_X_c;
                    *yuv2packed2 = yuv2xbgr32_full_2_c;
                    *yuv2packed1 = yuv2xbgr32_full_1_c;
                }
#endif /* !CONFIG_SMALL */
                break;
            case AV_PIX_FMT_RGB24:
            *yuv2packedX = yuv2rgb24_full_X_c;
            *yuv2packed2 = yuv2rgb24_full_2_c;
            *yuv2packed1 = yuv2rgb24_full_1_c;
            break;
        case AV_PIX_FMT_BGR24:
            *yuv2packedX = yuv2bgr24_full_X_c;
            *yuv2packed2 = yuv2bgr24_full_2_c;
            *yuv2packed1 = yuv2bgr24_full_1_c;
            break;
        case AV_PIX_FMT_BGR4_BYTE:
            *yuv2packedX = yuv2bgr4_byte_full_X_c;
            *yuv2packed2 = yuv2bgr4_byte_full_2_c;
            *yuv2packed1 = yuv2bgr4_byte_full_1_c;
            break;
        case AV_PIX_FMT_RGB4_BYTE:
            *yuv2packedX = yuv2rgb4_byte_full_X_c;
            *yuv2packed2 = yuv2rgb4_byte_full_2_c;
            *yuv2packed1 = yuv2rgb4_byte_full_1_c;
            break;
        case AV_PIX_FMT_BGR8:
            *yuv2packedX = yuv2bgr8_full_X_c;
            *yuv2packed2 = yuv2bgr8_full_2_c;
            *yuv2packed1 = yuv2bgr8_full_1_c;
            break;
        case AV_PIX_FMT_RGB8:
            *yuv2packedX = yuv2rgb8_full_X_c;
            *yuv2packed2 = yuv2rgb8_full_2_c;
            *yuv2packed1 = yuv2rgb8_full_1_c;
            break;
        case AV_PIX_FMT_GBRP:
        case AV_PIX_FMT_GBRP9BE:
        case AV_PIX_FMT_GBRP9LE:
        case AV_PIX_FMT_GBRP10BE:
        case AV_PIX_FMT_GBRP10LE:
        case AV_PIX_FMT_GBRP12BE:
        case AV_PIX_FMT_GBRP12LE:
        case AV_PIX_FMT_GBRP14BE:
        case AV_PIX_FMT_GBRP14LE:
        case AV_PIX_FMT_GBRP16BE:
        case AV_PIX_FMT_GBRP16LE:
        case AV_PIX_FMT_GBRAP:
            *yuv2anyX = yuv2gbrp_full_X_c;
            break;
        }
        if (!*yuv2packedX && !*yuv2anyX)
            goto YUV_PACKED;
    } 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
```

```
#endif /* CONFIG_SWSCALE_ALPHA */
                {
                    *yuv2packed1 = yuv2rgbx32_1_c;
                    *yuv2packed2 = yuv2rgbx32_2_c;
                    *yuv2packedX = yuv2rgbx32_X_c;
                }
#endif /* !CONFIG_SMALL */
            break;
        case AV_PIX_FMT_RGB32_1:
        case AV_PIX_FMT_BGR32_1:
#if CONFIG_SMALL
                *yuv2packed1 = yuv2rgb32_1_1_c;
                *yuv2packed2 = yuv2rgb32_1_2_c;
                *yuv2packedX = yuv2rgb32_1_X_c;
#else
#if CONFIG_SWSCALE_ALPHA
                if (c->alpPixBuf) {
                    *yuv2packed1 = yuv2rgba32_1_1_c;
                    *yuv2packed2 = yuv2rgba32_1_2_c;
                    *yuv2packedX = yuv2rgba32_1_X_c;
                } else
#endif /* CONFIG_SWSCALE_ALPHA */
                {
                    *yuv2packed1 = yuv2rgbx32_1_1_c;
                    *yuv2packed2 = yuv2rgbx32_1_2_c;
                    *yuv2packedX = yuv2rgbx32_1_X_c;
                }
#endif /* !CONFIG_SMALL */
                break;
        case AV_PIX_FMT_RGB24:
            *yuv2packed1 = yuv2rgb24_1_c;
            *yuv2packed2 = yuv2rgb24_2_c;
            *yuv2packedX = yuv2rgb24_X_c;
            break;
        case AV_PIX_FMT_BGR24:
            *yuv2packed1 = yuv2bgr24_1_c;
            *yuv2packed2 = yuv2bgr24_2_c;
            *yuv2packedX = yuv2bgr24_X_c;
            break;
        case AV_PIX_FMT_RGB565LE:
        case AV_PIX_FMT_RGB565BE:
        case AV_PIX_FMT_BGR565LE:
        case AV_PIX_FMT_BGR565BE:
            *yuv2packed1 = yuv2rgb16_1_c;
            *yuv2packed2 = yuv2rgb16_2_c;
            *yuv2packedX = yuv2rgb16_X_c;
            break;
        case AV_PIX_FMT_RGB555LE:
        case AV_PIX_FMT_RGB555BE:
        case AV_PIX_FMT_BGR555LE:
        case AV_PIX_FMT_BGR555BE:
            *yuv2packed1 = yuv2rgb15_1_c;
            *yuv2packed2 = yuv2rgb15_2_c;
            *yuv2packedX = yuv2rgb15_X_c;
            break;
        case AV_PIX_FMT_RGB444LE:
        case AV_PIX_FMT_RGB444BE:
        case AV_PIX_FMT_BGR444LE:
        case AV_PIX_FMT_BGR444BE:
            *yuv2packed1 = yuv2rgb12_1_c;
            *yuv2packed2 = yuv2rgb12_2_c;
            *yuv2packedX = yuv2rgb12_X_c;
            break;
        case AV_PIX_FMT_RGB8:
        case AV_PIX_FMT_BGR8:
            *yuv2packed1 = yuv2rgb8_1_c;
            *yuv2packed2 = yuv2rgb8_2_c;
            *yuv2packedX = yuv2rgb8_X_c;
            break;
        case AV_PIX_FMT_RGB4:
        case AV_PIX_FMT_BGR4:
            *yuv2packed1 = yuv2rgb4_1_c;
            *yuv2packed2 = yuv2rgb4_2_c;
            *yuv2packedX = yuv2rgb4_X_c;
            break;
        case AV_PIX_FMT_RGB4_BYTE:
        case AV_PIX_FMT_BGR4_BYTE:
            *yuv2packed1 = yuv2rgb4b_1_c;
            *yuv2packed2 = yuv2rgb4b_2_c;
            *yuv2packedX = yuv2rgb4b_X_c;
            break;
        }
    }
    switch (dstFormat) {
    case AV_PIX_FMT_MONOWHITE:
        *yuv2packed1 = yuv2monowhite_1_c;
        *yuv2packed2 = yuv2monowhite_2_c;
        *yuv2packedX = yuv2monowhite_X_c;
        break;
    case AV_PIX_FMT_MONOBLACK:
        *yuv2packed1 = yuv2monoblack_1_c;
```

```
        *yuv2packed2 = yuv2monoblack_2_c;
        *yuv2packedX = yuv2monoblack_X_c;
        break;
    case AV_PIX_FMT_YUYV422:
        *yuv2packed1 = yuv2yuyv422_1_c;
        *yuv2packed2 = yuv2yuyv422_2_c;
        *yuv2packedX = yuv2yuyv422_X_c;
        break;
    case AV_PIX_FMT_YVYU422:
        *yuv2packed1 = yuv2yvyu422_1_c;
        *yuv2packed2 = yuv2yvyu422_2_c;
        *yuv2packedX = yuv2yvyu422_X_c;
        break;
    case AV_PIX_FMT_UYVY422:
        *yuv2packed1 = yuv2uyvy422_1_c;
        *yuv2packed2 = yuv2uyvy422_2_c;
        *yuv2packedX = yuv2uyvy422_X_c;
        break;
    }
}
```

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。

```cpp
av_cold void ff_sws_init_input_funcs(SwsContext *c)
{
    enum AVPixelFormat srcFormat = c->srcFormat;

    c->chrToYV12 = NULL;
    switch (srcFormat) {
    case AV_PIX_FMT_YUYV422:
        c->chrToYV12 = yuy2ToUV_c;
        break;
    case AV_PIX_FMT_YVYU422:
        c->chrToYV12 = yvy2ToUV_c;
        break;
    case AV_PIX_FMT_UYVY422:
        c->chrToYV12 = uyvyToUV_c;
        break;
    case AV_PIX_FMT_NV12:
        c->chrToYV12 = nv12ToUV_c;
        break;
    case AV_PIX_FMT_NV21:
        c->chrToYV12 = nv21ToUV_c;
        break;
    case AV_PIX_FMT_RGB8:
    case AV_PIX_FMT_BGR8:
    case AV_PIX_FMT_PAL8:
    case AV_PIX_FMT_BGR4_BYTE:
    case AV_PIX_FMT_RGB4_BYTE:
        c->chrToYV12 = palToUV_c;
        break;
    case AV_PIX_FMT_GBRP9LE:
        c->readChrPlanar = planar_rgb9le_to_uv;
        break;
    case AV_PIX_FMT_GBRP10LE:
        c->readChrPlanar = planar_rgb10le_to_uv;
        break;
    case AV_PIX_FMT_GBRP12LE:
        c->readChrPlanar = planar_rgb12le_to_uv;
        break;
    case AV_PIX_FMT_GBRP14LE:
        c->readChrPlanar = planar_rgb14le_to_uv;
        break;
    case AV_PIX_FMT_GBRAP16LE:
    case AV_PIX_FMT_GBRP16LE:
        c->readChrPlanar = planar_rgb16le_to_uv;
        break;
    case AV_PIX_FMT_GBRP9BE:
        c->readChrPlanar = planar_rgb9be_to_uv;
        break;
    case AV_PIX_FMT_GBRP10BE:
```

```
        c->readChrPlanar = planar_rgb10be_to_uv;
        break;
    case AV_PIX_FMT_GBRP12BE:
        c->readChrPlanar = planar_rgb12be_to_uv;
        break;
    case AV_PIX_FMT_GBRP14BE:
        c->readChrPlanar = planar_rgb14be_to_uv;
        break;
    case AV_PIX_FMT_GBRAP16BE:
    case AV_PIX_FMT_GBRP16BE:
        c->readChrPlanar = planar_rgb16be_to_uv;
        break;
    case AV_PIX_FMT_GBRAP:
    case AV_PIX_FMT_GBRP:
        c->readChrPlanar = planar_rgb_to_uv;
        break;
#if HAVE_BIGENDIAN
    case AV_PIX_FMT_YUV444P9LE:
    case AV_PIX_FMT_YUV422P9LE:
    case AV_PIX_FMT_YUV420P9LE:
    case AV_PIX_FMT_YUV422P10LE:
    case AV_PIX_FMT_YUV444P10LE:
    case AV_PIX_FMT_YUV420P10LE:
    case AV_PIX_FMT_YUV422P12LE:
    case AV_PIX_FMT_YUV444P12LE:
    case AV_PIX_FMT_YUV420P12LE:
    case AV_PIX_FMT_YUV422P14LE:
    case AV_PIX_FMT_YUV444P14LE:
    case AV_PIX_FMT_YUV420P14LE:
    case AV_PIX_FMT_YUV420P16LE:
    case AV_PIX_FMT_YUV422P16LE:
    case AV_PIX_FMT_YUV444P16LE:

    case AV_PIX_FMT_YUVA444P9LE:
    case AV_PIX_FMT_YUVA422P9LE:
    case AV_PIX_FMT_YUVA420P9LE:
    case AV_PIX_FMT_YUVA444P10LE:
    case AV_PIX_FMT_YUVA422P10LE:
    case AV_PIX_FMT_YUVA420P10LE:
    case AV_PIX_FMT_YUVA420P16LE:
    case AV_PIX_FMT_YUVA422P16LE:
    case AV_PIX_FMT_YUVA444P16LE:
        c->chrToYV12 = bswap16UV_c;
        break;
#else
    case AV_PIX_FMT_YUV444P9BE:
    case AV_PIX_FMT_YUV422P9BE:
    case AV_PIX_FMT_YUV420P9BE:
    case AV_PIX_FMT_YUV444P10BE:
    case AV_PIX_FMT_YUV422P10BE:
    case AV_PIX_FMT_YUV420P10BE:
    case AV_PIX_FMT_YUV444P12BE:
    case AV_PIX_FMT_YUV422P12BE:
    case AV_PIX_FMT_YUV420P12BE:
    case AV_PIX_FMT_YUV444P14BE:
    case AV_PIX_FMT_YUV422P14BE:
    case AV_PIX_FMT_YUV420P14BE:
    case AV_PIX_FMT_YUV420P16BE:
    case AV_PIX_FMT_YUV422P16BE:
    case AV_PIX_FMT_YUV444P16BE:

    case AV_PIX_FMT_YUVA444P9BE:
    case AV_PIX_FMT_YUVA422P9BE:
    case AV_PIX_FMT_YUVA420P9BE:
    case AV_PIX_FMT_YUVA444P10BE:
    case AV_PIX_FMT_YUVA422P10BE:
    case AV_PIX_FMT_YUVA420P10BE:
    case AV_PIX_FMT_YUVA420P16BE:
    case AV_PIX_FMT_YUVA422P16BE:
    case AV_PIX_FMT_YUVA444P16BE:
        c->chrToYV12 = bswap16UV_c;
        break;
#endif
    }
    if (c->chrSrcHSubSample) {
        switch (srcFormat) {
        case AV_PIX_FMT_RGBA64BE:
            c->chrToYV12 = rgb64BEToUV_half_c;
            break;
        case AV_PIX_FMT_RGBA64LE:
            c->chrToYV12 = rgb64LEToUV_half_c;
            break;
        case AV_PIX_FMT_BGRA64BE:
            c->chrToYV12 = bgr64BEToUV_half_c;
            break;
        case AV_PIX_FMT_BGRA64LE:
            c->chrToYV12 = bgr64LEToUV_half_c;
            break;
        case AV_PIX_FMT_RGB48BE:
            c->chrToYV12 = rgb48BEToUV_half_c;
            break;
```

```
        case AV_PIX_FMT_RGB48LE:
            c->chrToYV12 = rgb48LEToUV_half_c;
            break;
        case AV_PIX_FMT_BGR48BE:
            c->chrToYV12 = bgr48BEToUV_half_c;
            break;
        case AV_PIX_FMT_BGR48LE:
            c->chrToYV12 = bgr48LEToUV_half_c;
            break;
        case AV_PIX_FMT_RGB32:
            c->chrToYV12 = bgr32ToUV_half_c;
            break;
        case AV_PIX_FMT_RGB32_1:
            c->chrToYV12 = bgr321ToUV_half_c;
            break;
        case AV_PIX_FMT_BGR24:
            c->chrToYV12 = bgr24ToUV_half_c;
            break;
        case AV_PIX_FMT_BGR565LE:
            c->chrToYV12 = bgr16leToUV_half_c;
            break;
        case AV_PIX_FMT_BGR565BE:
            c->chrToYV12 = bgr16beToUV_half_c;
            break;
        case AV_PIX_FMT_BGR555LE:
            c->chrToYV12 = bgr15leToUV_half_c;
            break;
        case AV_PIX_FMT_BGR555BE:
            c->chrToYV12 = bgr15beToUV_half_c;
            break;
        case AV_PIX_FMT_GBRAP:
        case AV_PIX_FMT_GBRP:
            c->chrToYV12 = gbr24pToUV_half_c;
            break;
        case AV_PIX_FMT_BGR444LE:
            c->chrToYV12 = bgr12leToUV_half_c;
            break;
        case AV_PIX_FMT_BGR444BE:
            c->chrToYV12 = bgr12beToUV_half_c;
            break;
        case AV_PIX_FMT_BGR32:
            c->chrToYV12 = rgb32ToUV_half_c;
            break;
        case AV_PIX_FMT_BGR32_1:
            c->chrToYV12 = rgb321ToUV_half_c;
            break;
        case AV_PIX_FMT_RGB24:
            c->chrToYV12 = rgb24ToUV_half_c;
            break;
        case AV_PIX_FMT_RGB565LE:
            c->chrToYV12 = rgb16leToUV_half_c;
            break;
        case AV_PIX_FMT_RGB565BE:
            c->chrToYV12 = rgb16beToUV_half_c;
            break;
        case AV_PIX_FMT_RGB555LE:
            c->chrToYV12 = rgb15leToUV_half_c;
            break;
        case AV_PIX_FMT_RGB555BE:
            c->chrToYV12 = rgb15beToUV_half_c;
            break;
        case AV_PIX_FMT_RGB444LE:
            c->chrToYV12 = rgb12leToUV_half_c;
            break;
        case AV_PIX_FMT_RGB444BE:
            c->chrToYV12 = rgb12beToUV_half_c;
            break;
        }
    } else {
        switch (srcFormat) {
        case AV_PIX_FMT_RGBA64BE:
            c->chrToYV12 = rgb64BEToUV_c;
            break;
        case AV_PIX_FMT_RGBA64LE:
            c->chrToYV12 = rgb64LEToUV_c;
            break;
        case AV_PIX_FMT_BGRA64BE:
            c->chrToYV12 = bgr64BEToUV_c;
            break;
        case AV_PIX_FMT_BGRA64LE:
            c->chrToYV12 = bgr64LEToUV_c;
            break;
        case AV_PIX_FMT_RGB48BE:
            c->chrToYV12 = rgb48BEToUV_c;
            break;
        case AV_PIX_FMT_RGB48LE:
            c->chrToYV12 = rgb48LEToUV_c;
            break;
        case AV_PIX_FMT_BGR48BE:
            c->chrToYV12 = bgr48BEToUV_c;
            break;
        case AV_PIX_FMT_BGR48LE:
```

```
            case AV_PIX_FMT_BGR48LE:
                c->chrToYV12 = bgr48LEToUV_c;
                break;
            case AV_PIX_FMT_RGB32:
                c->chrToYV12 = bgr32ToUV_c;
                break;
            case AV_PIX_FMT_RGB32_1:
                c->chrToYV12 = bgr321ToUV_c;
                break;
            case AV_PIX_FMT_BGR24:
                c->chrToYV12 = bgr24ToUV_c;
                break;
            case AV_PIX_FMT_BGR565LE:
                c->chrToYV12 = bgr16leToUV_c;
                break;
            case AV_PIX_FMT_BGR565BE:
                c->chrToYV12 = bgr16beToUV_c;
                break;
            case AV_PIX_FMT_BGR555LE:
                c->chrToYV12 = bgr15leToUV_c;
                break;
            case AV_PIX_FMT_BGR555BE:
                c->chrToYV12 = bgr15beToUV_c;
                break;
            case AV_PIX_FMT_BGR444LE:
                c->chrToYV12 = bgr12leToUV_c;
                break;
            case AV_PIX_FMT_BGR444BE:
                c->chrToYV12 = bgr12beToUV_c;
                break;
            case AV_PIX_FMT_BGR32:
                c->chrToYV12 = rgb32ToUV_c;
                break;
            case AV_PIX_FMT_BGR32_1:
                c->chrToYV12 = rgb321ToUV_c;
                break;
            case AV_PIX_FMT_RGB24:
                c->chrToYV12 = rgb24ToUV_c;
                break;
            case AV_PIX_FMT_RGB565LE:
                c->chrToYV12 = rgb16leToUV_c;
                break;
            case AV_PIX_FMT_RGB565BE:
                c->chrToYV12 = rgb16beToUV_c;
                break;
            case AV_PIX_FMT_RGB555LE:
                c->chrToYV12 = rgb15leToUV_c;
                break;
            case AV_PIX_FMT_RGB555BE:
                c->chrToYV12 = rgb15beToUV_c;
                break;
            case AV_PIX_FMT_RGB444LE:
                c->chrToYV12 = rgb12leToUV_c;
                break;
            case AV_PIX_FMT_RGB444BE:
                c->chrToYV12 = rgb12beToUV_c;
                break;
            }
        }

        c->lumToYV12 = NULL;
        c->alpToYV12 = NULL;
        switch (srcFormat) {
        case AV_PIX_FMT_GBRP9LE:
            c->readLumPlanar = planar_rgb9le_to_y;
            break;
        case AV_PIX_FMT_GBRP10LE:
            c->readLumPlanar = planar_rgb10le_to_y;
            break;
        case AV_PIX_FMT_GBRP12LE:
            c->readLumPlanar = planar_rgb12le_to_y;
            break;
        case AV_PIX_FMT_GBRP14LE:
            c->readLumPlanar = planar_rgb14le_to_y;
            break;
        case AV_PIX_FMT_GBRAP16LE:
        case AV_PIX_FMT_GBRP16LE:
            c->readLumPlanar = planar_rgb16le_to_y;
            break;
        case AV_PIX_FMT_GBRP9BE:
            c->readLumPlanar = planar_rgb9be_to_y;
            break;
        case AV_PIX_FMT_GBRP10BE:
            c->readLumPlanar = planar_rgb10be_to_y;
            break;
        case AV_PIX_FMT_GBRP12BE:
            c->readLumPlanar = planar_rgb12be_to_y;
            break;
        case AV_PIX_FMT_GBRP14BE:
            c->readLumPlanar = planar_rgb14be_to_y;
            break;
        case AV_PIX_FMT_GBRAP16BE:
```

```
    case AV_PIX_FMT_GBRP16BE:
        c->readLumPlanar = planar_rgb16be_to_y;
        break;
    case AV_PIX_FMT_GBRAP:
        c->readAlpPlanar = planar_rgb_to_a;
    case AV_PIX_FMT_GBRP:
        c->readLumPlanar = planar_rgb_to_y;
        break;
#if HAVE_BIGENDIAN
    case AV_PIX_FMT_YUV444P9LE:
    case AV_PIX_FMT_YUV422P9LE:
    case AV_PIX_FMT_YUV420P9LE:
    case AV_PIX_FMT_YUV444P10LE:
    case AV_PIX_FMT_YUV422P10LE:
    case AV_PIX_FMT_YUV420P10LE:
    case AV_PIX_FMT_YUV444P12LE:
    case AV_PIX_FMT_YUV422P12LE:
    case AV_PIX_FMT_YUV420P12LE:
    case AV_PIX_FMT_YUV444P14LE:
    case AV_PIX_FMT_YUV422P14LE:
    case AV_PIX_FMT_YUV420P14LE:
    case AV_PIX_FMT_YUV420P16LE:
    case AV_PIX_FMT_YUV422P16LE:
    case AV_PIX_FMT_YUV444P16LE:

    case AV_PIX_FMT_GRAY16LE:
        c->lumToYV12 = bswap16Y_c;
        break;
    case AV_PIX_FMT_YUVA444P9LE:
    case AV_PIX_FMT_YUVA422P9LE:
    case AV_PIX_FMT_YUVA420P9LE:
    case AV_PIX_FMT_YUVA444P10LE:
    case AV_PIX_FMT_YUVA422P10LE:
    case AV_PIX_FMT_YUVA420P10LE:
    case AV_PIX_FMT_YUVA420P16LE:
    case AV_PIX_FMT_YUVA422P16LE:
    case AV_PIX_FMT_YUVA444P16LE:
        c->lumToYV12 = bswap16Y_c;
        c->alpToYV12 = bswap16Y_c;
        break;
#else
    case AV_PIX_FMT_YUV444P9BE:
    case AV_PIX_FMT_YUV422P9BE:
    case AV_PIX_FMT_YUV420P9BE:
    case AV_PIX_FMT_YUV444P10BE:
    case AV_PIX_FMT_YUV422P10BE:
    case AV_PIX_FMT_YUV420P10BE:
    case AV_PIX_FMT_YUV444P12BE:
    case AV_PIX_FMT_YUV422P12BE:
    case AV_PIX_FMT_YUV420P12BE:
    case AV_PIX_FMT_YUV444P14BE:
    case AV_PIX_FMT_YUV422P14BE:
    case AV_PIX_FMT_YUV420P14BE:
    case AV_PIX_FMT_YUV420P16BE:
    case AV_PIX_FMT_YUV422P16BE:
    case AV_PIX_FMT_YUV444P16BE:

    case AV_PIX_FMT_GRAY16BE:
        c->lumToYV12 = bswap16Y_c;
        break;
    case AV_PIX_FMT_YUVA444P9BE:
    case AV_PIX_FMT_YUVA422P9BE:
    case AV_PIX_FMT_YUVA420P9BE:
    case AV_PIX_FMT_YUVA444P10BE:
    case AV_PIX_FMT_YUVA422P10BE:
    case AV_PIX_FMT_YUVA420P10BE:
    case AV_PIX_FMT_YUVA420P16BE:
    case AV_PIX_FMT_YUVA422P16BE:
    case AV_PIX_FMT_YUVA444P16BE:
        c->lumToYV12 = bswap16Y_c;
        c->alpToYV12 = bswap16Y_c;
        break;
#endif
    case AV_PIX_FMT_YA16LE:
        c->lumToYV12 = read_ya16le_gray_c;
        c->alpToYV12 = read_ya16le_alpha_c;
        break;
    case AV_PIX_FMT_YA16BE:
        c->lumToYV12 = read_ya16be_gray_c;
        c->alpToYV12 = read_ya16be_alpha_c;
        break;
    case AV_PIX_FMT_YUYV422:
    case AV_PIX_FMT_YVYU422:
    case AV_PIX_FMT_YA8:
        c->lumToYV12 = yuy2ToY_c;
        break;
    case AV_PIX_FMT_UYVY422:
        c->lumToYV12 = uyvyToY_c;
        break;
    case AV_PIX_FMT_BGR24:
        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)) {
```

```
            if (HAVE_BIGENDIAN == !isBE(srcFormat))
                c->alpToYV12 = bswap16Y_c;
        }
        switch (srcFormat) {
        case AV_PIX_FMT_BGRA64LE:
        case AV_PIX_FMT_BGRA64BE:
        case AV_PIX_FMT_RGBA64LE:
        case AV_PIX_FMT_RGBA64BE:  c->alpToYV12 = rgba64ToA_c; break;
        case AV_PIX_FMT_BGRA:
        case AV_PIX_FMT_RGBA:
            c->alpToYV12 = rgbaToA_c;
            break;
        case AV_PIX_FMT_ABGR:
        case AV_PIX_FMT_ARGB:
            c->alpToYV12 = abgrToA_c;
            break;
        case AV_PIX_FMT_YA8:
            c->alpToYV12 = uyvyToY_c;
            break;
        case AV_PIX_FMT_PAL8 :
            c->alpToYV12 = palToA_c;
            break;
        }
    }
}
```

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
case AV_PIX_FMT_RGB24:
    c->lumToYV12 = rgb24ToY_c;
    break;
```

rgb24ToY_c()

rgb24ToY_c()的定义如下。

```cpp
static void rgb24ToY_c(uint8_t *_dst, const uint8_t *src, const uint8_t *unused1, const uint8_t *unused2, int width,
                       uint32_t *rgb2yuv)
{
    int16_t *dst = (int16_t *)_dst;
    int32_t ry = rgb2yuv[RY_IDX], gy = rgb2yuv[GY_IDX], by = rgb2yuv[BY_IDX];
    int i;
    for (i = 0; i < width; i++) {
        int r = src[i * 3 + 0];
        int g = src[i * 3 + 1];
        int b = src[i * 3 + 2];

        dst[i] = ((ry*r + gy*g + by*b + (32<<(RGB2YUV_SHIFT-1)) + (1<<(RGB2YUV_SHIFT-7)))>>(RGB2YUV_SHIFT-6));
    }
}
```

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
case AV_PIX_FMT_RGB24:
    c->chrToYV12 = rgb24ToUV_half_c;
    break;
```

rgb24ToUV_half_c()

rgb24ToUV_half_c()定义如下。

```cpp
static void rgb24ToUV_half_c(uint8_t *_dstU, uint8_t *_dstV, const uint8_t *unused0, const uint8_t *src1,
                             const uint8_t *src2, int width, uint32_t *rgb2yuv)
{
    int16_t *dstU = (int16_t *)_dstU;
    int16_t *dstV = (int16_t *)_dstV;
    int i;
    int32_t ru = rgb2yuv[RU_IDX], gu = rgb2yuv[GU_IDX], bu = rgb2yuv[BU_IDX];
    int32_t rv = rgb2yuv[RV_IDX], gv = rgb2yuv[GV_IDX], bv = rgb2yuv[BV_IDX];
    av_assert1(src1 == src2);
    for (i = 0; i < width; i++) {
        int r = src1[6 * i + 0] + src1[6 * i + 3];
        int g = src1[6 * i + 1] + src1[6 * i + 4];
        int b = src1[6 * i + 2] + src1[6 * i + 5];

        dstU[i] = (ru*r + gu*g + bu*b + (256<<RGB2YUV_SHIFT) + (1<<(RGB2YUV_SHIFT-6)))>>(RGB2YUV_SHIFT-5);
        dstV[i] = (rv*r + gv*g + bv*b + (256<<RGB2YUV_SHIFT) + (1<<(RGB2YUV_SHIFT-6)))>>(RGB2YUV_SHIFT-5);
    }
}
```

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
case AV_PIX_FMT_GBRP:
    c->readLumPlanar = planar_rgb_to_y;
    break;
```

planar_rgb_to_y()

planar_rgb_to_y()定义如下。

```cpp
static void planar_rgb_to_y(uint8_t *_dst, const uint8_t *src[4], int width, int32_t *rgb2yuv)
{
    uint16_t *dst = (uint16_t *)_dst;
    int32_t ry = rgb2yuv[RY_IDX], gy = rgb2yuv[GY_IDX], by = rgb2yuv[BY_IDX];
    int i;
    for (i = 0; i < width; i++) {
        int g = src[0][i];
        int b = src[1][i];
        int r = src[2][i];

        dst[i] = (ry*r + gy*g + by*b + (0x801<<(RGB2YUV_SHIFT-7))) >> (RGB2YUV_SHIFT-6);
    }
}
```

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
av_cold void ff_sws_init_range_convert(SwsContext *c)
{
    c->lumConvertRange = NULL;
    c->chrConvertRange = NULL;
    if (c->srcRange != c->dstRange && !isAnyRGB(c->dstFormat)) {
        if (c->dstBpc <= 14) {
            if (c->srcRange) {
                c->lumConvertRange = lumRangeFromJpeg_c;
                c->chrConvertRange = chrRangeFromJpeg_c;
            } else {
                c->lumConvertRange = lumRangeToJpeg_c;
                c->chrConvertRange = chrRangeToJpeg_c;
            }
        } else {
            if (c->srcRange) {
                c->lumConvertRange = lumRangeFromJpeg16_c;
                c->chrConvertRange = chrRangeFromJpeg16_c;
            } else {
                c->lumConvertRange = lumRangeToJpeg16_c;
                c->chrConvertRange = chrRangeToJpeg16_c;
            }
        }
    }
}
```

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
static void lumRangeFromJpeg_c(int16_t *dst, int width)
{
    int i;
    for (i = 0; i < width; i++)
        dst[i] = (dst[i] * 14071 + 33561947) >> 14;
}
```

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
static void chrRangeFromJpeg_c(int16_t *dstU, int16_t *dstV, int width)
{
    int i;
    for (i = 0; i < width; i++) {
        dstU[i] = (dstU[i] * 1799 + 4081085) >> 11; // 1469
        dstV[i] = (dstV[i] * 1799 + 4081085) >> 11; // 1469
    }
}
```

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
static void lumRangeToJpeg_c(int16_t *dst, int width)
{
    int i;
    for (i = 0; i < width; i++)
        dst[i] = (FFMIN(dst[i], 30189) * 19077 - 39057361) >> 14;
}
```

chrRangeToJpeg_c()

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

```cpp
static void chrRangeToJpeg_c(int16_t *dstU, int16_t *dstV, int width)
{
    int i;
    for (i = 0; i < width; i++) {
        dstU[i] = (FFMIN(dstU[i], 30775) * 4663 - 9289992) >> 12; // -264
        dstV[i] = (FFMIN(dstV[i], 30775) * 4663 - 9289992) >> 12; // -264
    }
}
```

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

文章标签： swscale ffmpeg 源代码 YUV RGB

个人分类： FFMPEG

所属专栏： FFmpeg