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

2015年03月17日 20:02:33　阅读数：25817

==========

FFmpeg的库函数源代码分析文章列表：

【架构图】

FFmpeg 源代码结构图 - 解码

FFmpeg 源代码结构图 - 编码

【通用】

FFmpeg 源代码简单分析： av_register_all()

FFmpeg 源代码简单分析： avcodec_register_all()

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

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

FFmpeg 源代码简单分析： avio_open2()

FFmpeg 源代码简单分析： av_find_decoder() 和 av_find_encoder()

FFmpeg 源代码简单分析： avcodec_open2()

FFmpeg 源代码简单分析： avcodec_close()

【解码】

图解 FFMPEG 打开媒体的函数 avformat_open_input

FFmpeg 源代码简单分析： avformat_open_input()

FFmpeg 源代码简单分析： avformat_find_stream_info()

FFmpeg 源代码简单分析： av_read_frame()

FFmpeg 源代码简单分析： avcodec_decode_video2()

FFmpeg 源代码简单分析： avformat_close_input()

【编码】

FFmpeg 源代码简单分析： avformat_alloc_output_context2()

FFmpeg 源代码简单分析： avformat_write_header()

FFmpeg 源代码简单分析： avcodec_encode_video()

FFmpeg 源代码简单分析： av_write_frame()

FFmpeg 源代码简单分析： av_write_trailer()

【其它】

FFmpeg 源代码简单分析：日志输出系统（ av_log() 等）

FFmpeg 源代码简单分析：结构体成员管理系统 -AVClass

FFmpeg 源代码简单分析：结构体成员管理系统 -AVOption

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

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

FFmpeg 源代码简单分析： libavdevice 的 avdevice_register_all()

FFmpeg 源代码简单分析： libavdevice 的 gdigrab

【脚本】

FFmpeg 源代码简单分析： makefile

FFmpeg 源代码简单分析： configure

【H.264】

FFmpeg 的 H.264 解码器源代码简单分析：概述

==========

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

该类库常用的函数数量很少，一般情况下就3个：

sws_getContext()：初始化一个SwsContext。

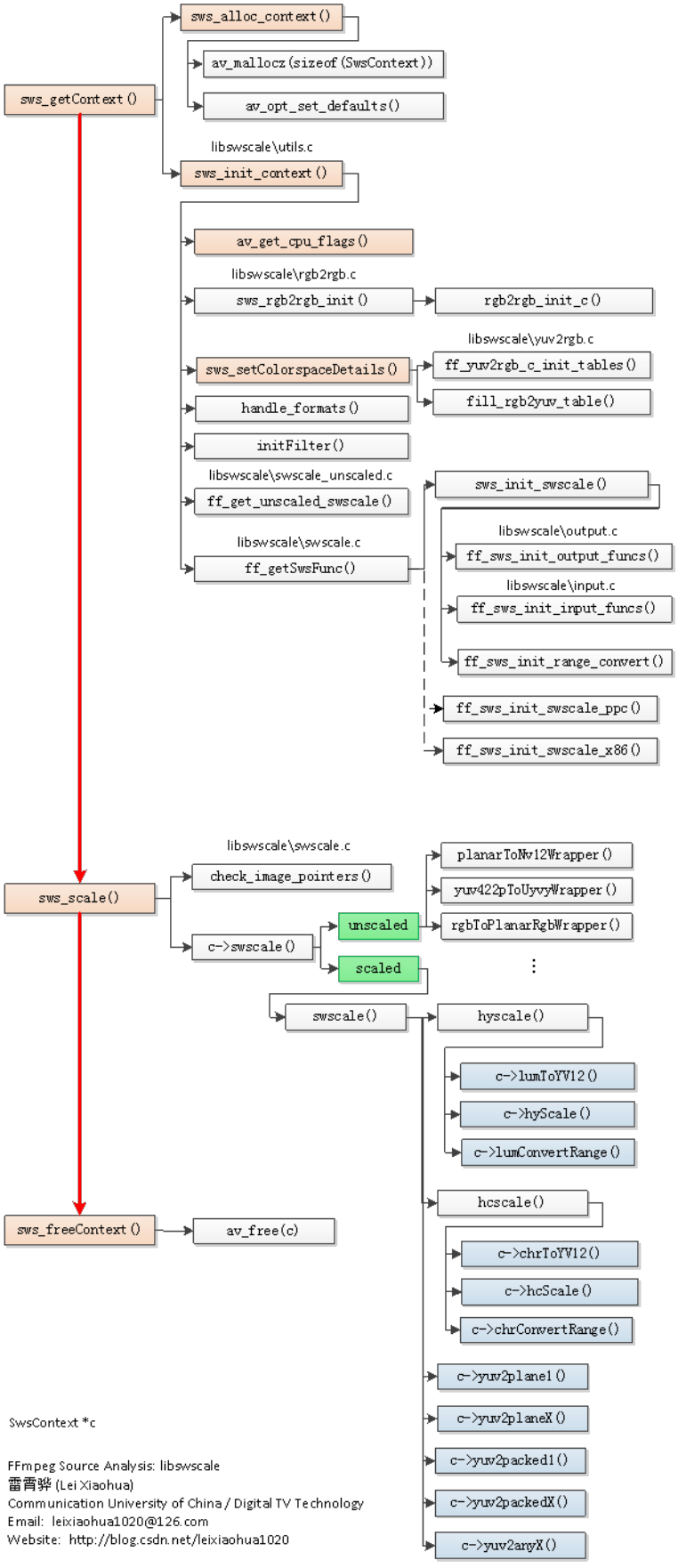
sws_scale()：处理图像数据。

sws_freeContext()：释放一个SwsContext。

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

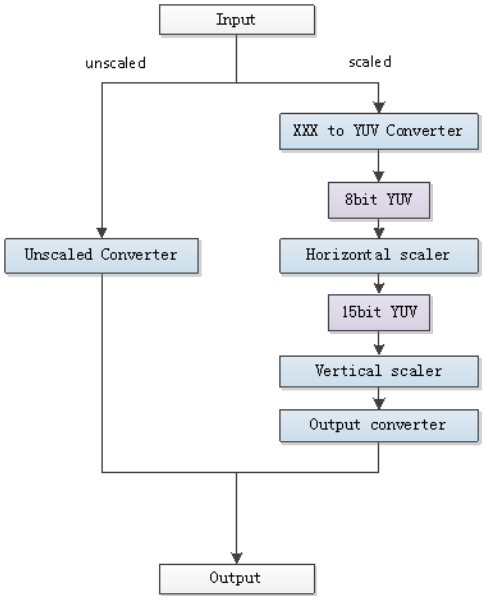
函数调用结构图

分析得到的libswscale的函数调用关系如下图所示。



Libswscale处理数据流程

Libswscale处理像素数据的流程可以概括为下图。



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

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

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

Vertical scaler：垂直拉伸图像；

Output converter：转换为输出像素格式。

sws_scale()

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

```cpp
/**
 * Scale the image slice in srcSlice and put the resulting scaled
 * slice in the image in dst. A slice is a sequence of consecutive
 * rows in an image.
 *
 * Slices have to be provided in sequential order, either in
 * top-bottom or bottom-top order. If slices are provided in
 * non-sequential order the behavior of the function is undefined.
 *
 * @param c         the scaling context previously created with
 *                  sws_getContext()
 * @param srcSlice  the array containing the pointers to the planes of
 *                  the source slice
 * @param srcStride the array containing the strides for each plane of
 *                  the source image
 * @param srcSliceY the position in the source image of the slice to
 *                  process, that is the number (counted starting from
 *                  zero) in the image of the first row of the slice
 * @param srcSliceH the height of the source slice, that is the number
 *                  of rows in the slice
 * @param dst       the array containing the pointers to the planes of
 *                  the destination image
 * @param dstStride the array containing the strides for each plane of
 *                  the destination image
 * @return          the height of the output slice
 */
int sws_scale(struct SwsContext *c, const uint8_t *const srcSlice[],
              const int srcStride[], int srcSliceY, int srcSliceH,
              uint8_t *const dst[], const int dstStride[]);
```

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

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

```
 * swscale wrapper, so we don't need to export the SwsContext.
 * Assumes planar YUV to be in YUV order instead of YVU.
 */
int sws_scale(struct SwsContext *c,
                                  const uint8_t * const srcSlice[],
                                  const int srcStride[], int srcSliceY,
                                  int srcSliceH, uint8_t *const dst[],
                                  const int dstStride[])
{
    int i, ret;
    const uint8_t *src2[4];
    uint8_t *dst2[4];
    uint8_t *rgb0_tmp = NULL;
    //检查输入参数
    if (!srcStride || !dstStride || !dst || !srcSlice) {
        av_log(c, AV_LOG_ERROR, "One of the input parameters to sws_scale() is NULL, please check the calling code\n");
        return 0;
    }
    if (c->cascaded_context[0] && srcSliceY == 0 && srcSliceH == c->cascaded_context[0]->srcH) {
        ret = sws_scale(c->cascaded_context[0],
                        srcSlice, srcStride, srcSliceY, srcSliceH,
                        c->cascaded_tmp, c->cascaded_tmpStride);
        if (ret < 0)
            return ret;
        ret = sws_scale(c->cascaded_context[1],
                        (const uint8_t * const * )c->cascaded_tmp, c->cascaded_tmpStride, 0, c->cascaded_context[0]->dstH,
                        dst, dstStride);
        return ret;
    }

    memcpy(src2, srcSlice, sizeof(src2));
    memcpy(dst2, dst, sizeof(dst2));

    // do not mess up sliceDir if we have a "trailing" 0-size slice
    if (srcSliceH == 0)
        return 0;
    //检查
    if (!check_image_pointers(srcSlice, c->srcFormat, srcStride)) {
        av_log(c, AV_LOG_ERROR, "bad src image pointers\n");
        return 0;
    }
    if (!check_image_pointers((const uint8_t* const*)dst, c->dstFormat, dstStride)) {
        av_log(c, AV_LOG_ERROR, "bad dst image pointers\n");
        return 0;
    }

    if (c->sliceDir == 0 && srcSliceY != 0 && srcSliceY + srcSliceH != c->srcH) {
        av_log(c, AV_LOG_ERROR, "Slices start in the middle!\n");
        return 0;
    }
    if (c->sliceDir == 0) {
        if (srcSliceY == 0) c->sliceDir = 1; else c->sliceDir = -1;
    }
    //使用调色板palette的特殊处理？应该不常见
    if (usePal(c->srcFormat)) {
        for (i = 0; i < 256; i++) {
            int r, g, b, y, u, v, a = 0xff;
            if (c->srcFormat == AV_PIX_FMT_PAL8) {
                uint32_t p = ((const uint32_t *)(srcSlice[1]))[i];
                a = (p >> 24) & 0xFF;
                r = (p >> 16) & 0xFF;
                g = (p >>  8) & 0xFF;
                b =  p        & 0xFF;
            } else if (c->srcFormat == AV_PIX_FMT_RGB8) {
                r = ( i >> 5     ) * 36;
                g = ((i >> 2) & 7) * 36;
                b = ( i       & 3) * 85;
            } else if (c->srcFormat == AV_PIX_FMT_BGR8) {
                b = ( i >> 6     ) * 85;
                g = ((i >> 3) & 7) * 36;
                r = ( i       & 7) * 36;
            } else if (c->srcFormat == AV_PIX_FMT_RGB4_BYTE) {
                r = ( i >> 3     ) * 255;
                g = ((i >> 1) & 3) * 85;
                b = ( i       & 1) * 255;
            } else if (c->srcFormat == AV_PIX_FMT_GRAY8 || c->srcFormat == AV_PIX_FMT_GRAY8A) {
                r = g = b = i;
            } else {
                av_assert1(c->srcFormat == AV_PIX_FMT_BGR4_BYTE);
                b = ( i >> 3     ) * 255;
                g = ((i >> 1) & 3) * 85;
                r = ( i       & 1) * 255;
            }
#define RGB2YUV_SHIFT 15
#define BY ( (int) (0.114 * 219 / 255 * (1 << RGB2YUV_SHIFT) + 0.5))
#define BV (-(int) (0.081 * 224 / 255 * (1 << RGB2YUV_SHIFT) + 0.5))
#define BU ( (int) (0.500 * 224 / 255 * (1 << RGB2YUV_SHIFT) + 0.5))
#define GY ( (int) (0.587 * 219 / 255 * (1 << RGB2YUV_SHIFT) + 0.5))
#define GV (-(int) (0.419 * 224 / 255 * (1 << RGB2YUV_SHIFT) + 0.5))
#define GU (-(int) (0.331 * 224 / 255 * (1 << RGB2YUV_SHIFT) + 0.5))
#define RY ( (int) (0.299 * 219 / 255 * (1 << RGB2YUV_SHIFT) + 0.5))
#define RV ( (int) (0.500 * 224 / 255 * (1 << RGB2YUV_SHIFT) + 0.5))
```

```
#define RU (-(int) (0.169 * 224 / 255 * (1 << RGB2YUV_SHIFT) + 0.5))

            y = av_clip_uint8((RY * r + GY * g + BY * b + ( 33 << (RGB2YUV_SHIFT - 1))) >> RGB2YUV_SHIFT);
            u = av_clip_uint8((RU * r + GU * g + BU * b + (257 << (RGB2YUV_SHIFT - 1))) >> RGB2YUV_SHIFT);
            v = av_clip_uint8((RV * r + GV * g + BV * b + (257 << (RGB2YUV_SHIFT - 1))) >> RGB2YUV_SHIFT);
            c->pal_yuv[i]= y + (u<<8) + (v<<16) + ((unsigned)a<<24);

            switch (c->dstFormat) {
            case AV_PIX_FMT_BGR32:
#if !HAVE_BIGENDIAN
            case AV_PIX_FMT_RGB24:
#endif
                c->pal_rgb[i]=  r + (g<<8) + (b<<16) + ((unsigned)a<<24);
                break;
            case AV_PIX_FMT_BGR32_1:
#if HAVE_BIGENDIAN
            case AV_PIX_FMT_BGR24:
#endif
                c->pal_rgb[i]= a + (r<<8) + (g<<16) + ((unsigned)b<<24);
                break;
            case AV_PIX_FMT_RGB32_1:
#if HAVE_BIGENDIAN
            case AV_PIX_FMT_RGB24:
#endif
                c->pal_rgb[i]= a + (b<<8) + (g<<16) + ((unsigned)r<<24);
                break;
            case AV_PIX_FMT_RGB32:
#if !HAVE_BIGENDIAN
            case AV_PIX_FMT_BGR24:
#endif
            default:
                c->pal_rgb[i]=  b + (g<<8) + (r<<16) + ((unsigned)a<<24);
            }
        }
    }
    //Alpha的特殊处理?
    if (c->src0Alpha && !c->dst0Alpha && isALPHA(c->dstFormat)) {
        uint8_t *base;
        int x,y;
        rgb0_tmp = av_malloc(FFABS(srcStride[0]) * srcSliceH + 32);
        if (!rgb0_tmp)
            return AVERROR(ENOMEM);

        base = srcStride[0] < 0 ? rgb0_tmp - srcStride[0] * (srcSliceH-1) : rgb0_tmp;
        for (y=0; y<srcSliceH; y++){
            memcpy(base + srcStride[0]*y, src2[0] + srcStride[0]*y, 4*c->srcW);
            for (x=c->src0Alpha-1; x<4*c->srcW; x+=4) {
                base[ srcStride[0]*y + x] = 0xFF;
            }
        }
        src2[0] = base;
    }
    //XYZ的特殊处理?
    if (c->srcXYZ && !(c->dstXYZ && c->srcW==c->dstW && c->srcH==c->dstH)) {
        uint8_t *base;
        rgb0_tmp = av_malloc(FFABS(srcStride[0]) * srcSliceH + 32);
        if (!rgb0_tmp)
            return AVERROR(ENOMEM);

        base = srcStride[0] < 0 ? rgb0_tmp - srcStride[0] * (srcSliceH-1) : rgb0_tmp;

        xyz12Torgb48(c, (uint16_t*)base, (const uint16_t*)src2[0], srcStride[0]/2, srcSliceH);
        src2[0] = base;
    }

    if (!srcSliceY && (c->flags & SWS_BITEXACT) && c->dither == SWS_DITHER_ED && c->dither_error[0])
        for (i = 0; i < 4; i++)
            memset(c->dither_error[i], 0, sizeof(c->dither_error[0][0]) * (c->dstW+2));


    // copy strides, so they can safely be modified
    // sliceDir: 1 = top-to-bottom; -1 = bottom-to-top;
    if (c->sliceDir == 1) {
        // slices go from top to bottom
        int srcStride2[4] = { srcStride[0], srcStride[1], srcStride[2],
                              srcStride[3] };
        int dstStride2[4] = { dstStride[0], dstStride[1], dstStride[2],
                              dstStride[3] };

        reset_ptr(src2, c->srcFormat);
        reset_ptr((void*)dst2, c->dstFormat);

        /* reset slice direction at end of frame */
        if (srcSliceY + srcSliceH == c->srcH)
            c->sliceDir = 0;
        //关键：调用
        ret = c->swscale(c, src2, srcStride2, srcSliceY, srcSliceH, dst2,
                         dstStride2);
    } else {
        // slices go from bottom to top => we flip the image internally
        int srcStride2[4] = { -srcStride[0], -srcStride[1], -srcStride[2],
```

```
                               -srcStride[3] };
        int dstStride2[4] = { -dstStride[0], -dstStride[1], -dstStride[2],
                               -dstStride[3] };

        src2[0] += (srcSliceH - 1) * srcStride[0];
        if (!usePal(c->srcFormat))
            src2[1] += ((srcSliceH >> c->chrSrcVSubSample) - 1) * srcStride[1];
        src2[2] += ((srcSliceH >> c->chrSrcVSubSample) - 1) * srcStride[2];
        src2[3] += (srcSliceH - 1) * srcStride[3];
        dst2[0] += ( c->dstH                         - 1) * dstStride[0];
        dst2[1] += ((c->dstH >> c->chrDstVSubSample) - 1) * dstStride[1];
        dst2[2] += ((c->dstH >> c->chrDstVSubSample) - 1) * dstStride[2];
        dst2[3] += ( c->dstH                         - 1) * dstStride[3];

        reset_ptr(src2, c->srcFormat);
        reset_ptr((void*)dst2, c->dstFormat);

        /* reset slice direction at end of frame */
        if (!srcSliceY)
            c->sliceDir = 0;
        //关键：调用
        ret = c->swscale(c, src2, srcStride2, c->srcH-srcSliceY-srcSliceH,
                         srcSliceH, dst2, dstStride2);
    }


    if (c->dstXYZ && !(c->srcXYZ && c->srcW==c->dstW && c->srcH==c->dstH)) {
        /* replace on the same data */
        rgb48Toxyz12(c, (uint16_t*)dst2[0], (const uint16_t*)dst2[0], dstStride[0]/2, ret);
    }

    av_free(rgb0_tmp);
    return ret;
}
```

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

1.检查输入的图像参数的合理性。

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

```cpp
static int check_image_pointers(const uint8_t * const data[4], enum AVPixelFormat pix_fmt,
                                const int linesizes[4])
{
    const AVPixFmtDescriptor *desc = av_pix_fmt_desc_get(pix_fmt);
    int i;

    for (i = 0; i < 4; i++) {
        int plane = desc->comp[i].plane;
        if (!data[plane] || !linesizes[plane])
            return 0;
    }

    return 1;
}
```

从check_image_pointers()的定义可以看出，在特定像素格式前提下，如果该像素格式应该包含像素的分量为空，就返回0，否则返回1。
2.如果输入像素数据中使用了“调色板”（palette），则进行一些相应的处理。这一步通过函数usePal()来判定。 usePal()的定义如下。

```cpp
static av_always_inline int usePal(enum AVPixelFormat pix_fmt)
{
    const AVPixFmtDescriptor *desc = av_pix_fmt_desc_get(pix_fmt);
    av_assert0(desc);
    return (desc->flags & AV_PIX_FMT_FLAG_PAL) || (desc->flags & AV_PIX_FMT_FLAG_PSEUDOPAL);
}
```

从定义可以看出该函数通过判定AVPixFmtDescriptor中的flag是否包含AV_PIX_FMT_FLAG_PAL来断定像素格式是否使用了“调色板”。
3.其它一些特殊格式的处理，比如说Alpha，XYZ等的处理（这方面没有研究过）。
4.如果输入的图像的扫描方式是从底部到顶部的（一般情况下是从顶部到底部），则将图像进行反转。
5.调用SwsContext中的swscale()。

SwsContext中的swscale()

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

可以看一下SwsFunc这个类型的定义：

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

可以看出SwsFunc的定义的参数类型和libswscale类库外部接口函数swscale()的参数类型一模一样。
在libswscale中，该指针的指向可以分成2种情况：

1.图像没有伸缩的时候，指向专有的像素转换函数

2.图像有伸缩的时候，指向swscale()函数。

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

没有拉伸--专有的像素转换函数

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

ff_get_unscaled_swscale()

ff_get_unscaled_swscale()的定义如下。

```cpp
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);
}
```

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

有拉伸--swscale()

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

```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;
}
```

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

```cpp
c->swscale = ff_getSwsFunc(c);
```

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

```cpp
static int swscale(SwsContext *c, const uint8_t *src[],
                   int srcStride[], int srcSliceY,
                   int srcSliceH, uint8_t *dst[], int dstStride[])
```

```
{
    /* load a few things into local vars to make the code more readable?
     * and faster */
    //注意一下这些参数
    //以亮度为准
    const int srcW                  = c->srcW;
    const int dstW                  = c->dstW;
    const int dstH                  = c->dstH;
    //以色度为准
    const int chrDstW               = c->chrDstW;
    const int chrSrcW               = c->chrSrcW;
    const int lumXInc               = c->lumXInc;
    const int chrXInc               = c->chrXInc;
    const enum AVPixelFormat dstFormat = c->dstFormat;
    const int flags                 = c->flags;
    int32_t *vLumFilterPos          = c->vLumFilterPos;
    int32_t *vChrFilterPos          = c->vChrFilterPos;
    int32_t *hLumFilterPos          = c->hLumFilterPos;
    int32_t *hChrFilterPos          = c->hChrFilterPos;
    int16_t *hLumFilter             = c->hLumFilter;
    int16_t *hChrFilter             = c->hChrFilter;
    int32_t *lumMmxFilter           = c->lumMmxFilter;
    int32_t *chrMmxFilter           = c->chrMmxFilter;
    const int vLumFilterSize        = c->vLumFilterSize;
    const int vChrFilterSize        = c->vChrFilterSize;
    const int hLumFilterSize        = c->hLumFilterSize;
    const int hChrFilterSize        = c->hChrFilterSize;
    int16_t **lumPixBuf             = c->lumPixBuf;
    int16_t **chrUPixBuf            = c->chrUPixBuf;
    int16_t **chrVPixBuf            = c->chrVPixBuf;
    int16_t **alpPixBuf             = c->alpPixBuf;
    const int vLumBufSize           = c->vLumBufSize;
    const int vChrBufSize           = c->vChrBufSize;
    uint8_t *formatConvBuffer       = c->formatConvBuffer;
    uint32_t *pal                   = c->pal_yuv;
    yuv2planar1_fn yuv2plane1       = c->yuv2plane1;
    yuv2planarX_fn yuv2planeX       = c->yuv2planeX;
    yuv2interleavedX_fn yuv2nv12cX  = c->yuv2nv12cX;
    yuv2packed1_fn yuv2packed1      = c->yuv2packed1;
    yuv2packed2_fn yuv2packed2      = c->yuv2packed2;
    yuv2packedX_fn yuv2packedX      = c->yuv2packedX;
    yuv2anyX_fn yuv2anyX            = c->yuv2anyX;
    const int chrSrcSliceY          =               srcSliceY >> c->chrSrcVSubSample;
    const int chrSrcSliceH          = FF_CEIL_RSHIFT(srcSliceH,   c->chrSrcVSubSample);
    int should_dither               = is9_OR_10BPS(c->srcFormat) ||
                                      is16BPS(c->srcFormat);
    int lastDstY;

    /* vars which will change and which we need to store back in the context */
    int dstY         = c->dstY;
    int lumBufIndex  = c->lumBufIndex;
    int chrBufIndex  = c->chrBufIndex;
    int lastInLumBuf = c->lastInLumBuf;
    int lastInChrBuf = c->lastInChrBuf;

    if (!usePal(c->srcFormat)) {
        pal = c->input_rgb2yuv_table;
    }

    if (isPacked(c->srcFormat)) {
        src[0] =
        src[1] =
        src[2] =
        src[3] = src[0];
        srcStride[0] =
        srcStride[1] =
        srcStride[2] =
        srcStride[3] = srcStride[0];
    }
    srcStride[1] <<= c->vChrDrop;
    srcStride[2] <<= c->vChrDrop;

    DEBUG_BUFFERS("swscale() %p[%d] %p[%d] %p[%d] %p[%d] -> %p[%d] %p[%d] %p[%d] %p[%d]\n",
                  src[0], srcStride[0], src[1], srcStride[1],
                  src[2], srcStride[2], src[3], srcStride[3],
                  dst[0], dstStride[0], dst[1], dstStride[1],
                  dst[2], dstStride[2], dst[3], dstStride[3]);
    DEBUG_BUFFERS("srcSliceY: %d srcSliceH: %d dstY: %d dstH: %d\n",
                  srcSliceY, srcSliceH, dstY, dstH);
    DEBUG_BUFFERS("vLumFilterSize: %d vLumBufSize: %d vChrFilterSize: %d vChrBufSize: %d\n",
                  vLumFilterSize, vLumBufSize, vChrFilterSize, vChrBufSize);

    if (dstStride[0]&15 || dstStride[1]&15 ||
        dstStride[2]&15 || dstStride[3]&15) {
        static int warnedAlready = 0; // FIXME maybe move this into the context
        if (flags & SWS_PRINT_INFO && !warnedAlready) {
            av_log(c, AV_LOG_WARNING,
                   "Warning: dstStride is not aligned!\n"
                   "         ->cannot do aligned memory accesses anymore\n");
            warnedAlready = 1;
        }
    }
```

```
    }

    if (   (uintptr_t)dst[0]&15 || (uintptr_t)dst[1]&15 || (uintptr_t)dst[2]&15
        || (uintptr_t)src[0]&15 || (uintptr_t)src[1]&15 || (uintptr_t)src[2]&15
        || dstStride[0]&15 || dstStride[1]&15 || dstStride[2]&15 || dstStride[3]&15
        || srcStride[0]&15 || srcStride[1]&15 || srcStride[2]&15 || srcStride[3]&15
    ) {
        static int warnedAlready=0;
        int cpu_flags = av_get_cpu_flags();
        if (HAVE_MMXEXT && (cpu_flags & AV_CPU_FLAG_SSE2) && !warnedAlready){
            av_log(c, AV_LOG_WARNING, "Warning: data is not aligned! This can lead to a speedloss\n");
            warnedAlready=1;
        }
    }

    /* Note the user might start scaling the picture in the middle so this
     * will not get executed. This is not really intended but works
     * currently, so people might do it. */
    if (srcSliceY == 0) {
        lumBufIndex  = -1;
        chrBufIndex  = -1;
        dstY         = 0;
        lastInLumBuf = -1;
        lastInChrBuf = -1;
    }

    if (!should_dither) {
        c->chrDither8 = c->lumDither8 = sws_pb_64;
    }
    lastDstY = dstY;
    //逐行循环，一次循环代表处理一行
    //注意dstY和dstH两个变量
    for (; dstY < dstH; dstY++) {
        //色度的和亮度之间的关系
        const int chrDstY = dstY >> c->chrDstVSubSample;
        uint8_t *dest[4]  = {
            dst[0] + dstStride[0] * dstY,
            dst[1] + dstStride[1] * chrDstY,
            dst[2] + dstStride[2] * chrDstY,
            (CONFIG_SWSCALE_ALPHA && alpPixBuf) ? dst[3] + dstStride[3] * dstY : NULL,
        };
        int use_mmx_vfilter= c->use_mmx_vfilter;

        // First line needed as input
        const int firstLumSrcY  = FFMAX(1 - vLumFilterSize, vLumFilterPos[dstY]);
        const int firstLumSrcY2 = FFMAX(1 - vLumFilterSize, vLumFilterPos[FFMIN(dstY | ((1 << c-
>chrDstVSubSample) - 1), dstH - 1)]);
        // First line needed as input
        const int firstChrSrcY  = FFMAX(1 - vChrFilterSize, vChrFilterPos[chrDstY]);

        // Last line needed as input
        int lastLumSrcY  = FFMIN(c->srcH,    firstLumSrcY  + vLumFilterSize) - 1;
        int lastLumSrcY2 = FFMIN(c->srcH,    firstLumSrcY2 + vLumFilterSize) - 1;
        int lastChrSrcY  = FFMIN(c->chrSrcH, firstChrSrcY  + vChrFilterSize) - 1;
        int enough_lines;

        // handle holes (FAST_BILINEAR & weird filters)
        if (firstLumSrcY > lastInLumBuf)
            lastInLumBuf = firstLumSrcY - 1;
        if (firstChrSrcY > lastInChrBuf)
            lastInChrBuf = firstChrSrcY - 1;
        av_assert0(firstLumSrcY >= lastInLumBuf - vLumBufSize + 1);
        av_assert0(firstChrSrcY >= lastInChrBuf - vChrBufSize + 1);

        DEBUG_BUFFERS("dstY: %d\n", dstY);
        DEBUG_BUFFERS("\tfirstLumSrcY: %d lastLumSrcY: %d lastInLumBuf: %d\n",
                      firstLumSrcY, lastLumSrcY, lastInLumBuf);
        DEBUG_BUFFERS("\tfirstChrSrcY: %d lastChrSrcY: %d lastInChrBuf: %d\n",
                      firstChrSrcY, lastChrSrcY, lastInChrBuf);

        // Do we have enough lines in this slice to output the dstY line
        enough_lines = lastLumSrcY2 < srcSliceY + srcSliceH &&
                       lastChrSrcY < FF_CEIL_RSHIFT(srcSliceY + srcSliceH, c->chrSrcVSubSample);

        if (!enough_lines) {
            lastLumSrcY = srcSliceY + srcSliceH - 1;
            lastChrSrcY = chrSrcSliceY + chrSrcSliceH - 1;
            DEBUG_BUFFERS("buffering slice: lastLumSrcY %d lastChrSrcY %d\n",
                          lastLumSrcY, lastChrSrcY);
        }

        // Do horizontal scaling
        //水平拉伸
        //亮度
        while (lastInLumBuf < lastLumSrcY) {
            const uint8_t *src1[4] = {
                src[0] + (lastInLumBuf + 1 - srcSliceY) * srcStride[0],
                src[1] + (lastInLumBuf + 1 - srcSliceY) * srcStride[1],
                src[2] + (lastInLumBuf + 1 - srcSliceY) * srcStride[2],
                src[3] + (lastInLumBuf + 1 - srcSliceY) * srcStride[3],
            };
            lumBufIndex++;
```

```
            av_assert0(lumBufIndex < 2 * vLumBufSize);
            av_assert0(lastInLumBuf + 1 - srcSliceY < srcSliceH);
            av_assert0(lastInLumBuf + 1 - srcSliceY >= 0);
            //关键：拉伸
            hyscale(c, lumPixBuf[lumBufIndex], dstW, src1, srcW, lumXInc,
                    hLumFilter, hLumFilterPos, hLumFilterSize,
                    formatConvBuffer, pal, 0);
            if (CONFIG_SWSCALE_ALPHA && alpPixBuf)
                hyscale(c, alpPixBuf[lumBufIndex], dstW, src1, srcW,
                        lumXInc, hLumFilter, hLumFilterPos, hLumFilterSize,
                        formatConvBuffer, pal, 1);
            lastInLumBuf++;
            DEBUG_BUFFERS("\t\tlumBufIndex %d: lastInLumBuf: %d\n",
                          lumBufIndex, lastInLumBuf);
        }
        //水平拉伸
        //色度
        while (lastInChrBuf < lastChrSrcY) {
            const uint8_t *src1[4] = {
                src[0] + (lastInChrBuf + 1 - chrSrcSliceY) * srcStride[0],
                src[1] + (lastInChrBuf + 1 - chrSrcSliceY) * srcStride[1],
                src[2] + (lastInChrBuf + 1 - chrSrcSliceY) * srcStride[2],
                src[3] + (lastInChrBuf + 1 - chrSrcSliceY) * srcStride[3],
            };
            chrBufIndex++;
            av_assert0(chrBufIndex < 2 * vChrBufSize);
            av_assert0(lastInChrBuf + 1 - chrSrcSliceY < (chrSrcSliceH));
            av_assert0(lastInChrBuf + 1 - chrSrcSliceY >= 0);
            // FIXME replace parameters through context struct (some at least)
            //关键：拉伸
            if (c->needs_hcscale)
                hcscale(c, chrUPixBuf[chrBufIndex], chrVPixBuf[chrBufIndex],
                        chrDstW, src1, chrSrcW, chrXInc,
                        hChrFilter, hChrFilterPos, hChrFilterSize,
                        formatConvBuffer, pal);
            lastInChrBuf++;
            DEBUG_BUFFERS("\t\tchrBufIndex %d: lastInChrBuf: %d\n",
                          chrBufIndex, lastInChrBuf);
        }
        // wrap buf index around to stay inside the ring buffer
        if (lumBufIndex >= vLumBufSize)
            lumBufIndex -= vLumBufSize;
        if (chrBufIndex >= vChrBufSize)
            chrBufIndex -= vChrBufSize;
        if (!enough_lines)
            break;  // we can't output a dstY line so let's try with the next slice

#if HAVE_MMX_INLINE
        updateMMXDitherTables(c, dstY, lumBufIndex, chrBufIndex,
                              lastInLumBuf, lastInChrBuf);
#endif
        if (should_dither) {
            c->chrDither8 = ff_dither_8x8_128[chrDstY & 7];
            c->lumDither8 = ff_dither_8x8_128[dstY    & 7];
        }
        if (dstY >= dstH - 2) {
            /* hmm looks like we can't use MMX here without overwriting
             * this array's tail */
            ff_sws_init_output_funcs(c, &yuv2plane1, &yuv2planeX, &yuv2nv12cX,
                                     &yuv2packed1, &yuv2packed2, &yuv2packedX, &yuv2anyX);
            use_mmx_vfilter= 0;
        }

        {
            const int16_t **lumSrcPtr  = (const int16_t **)
(void*) lumPixBuf  + lumBufIndex + firstLumSrcY - lastInLumBuf + vLumBufSize;
            const int16_t **chrUSrcPtr = (const int16_t **)
(void*) chrUPixBuf + chrBufIndex + firstChrSrcY - lastInChrBuf + vChrBufSize;
            const int16_t **chrVSrcPtr = (const int16_t **)
(void*) chrVPixBuf + chrBufIndex + firstChrSrcY - lastInChrBuf + vChrBufSize;
            const int16_t **alpSrcPtr  = (CONFIG_SWSCALE_ALPHA && alpPixBuf) ?
                                         (const int16_t **)
(void*) alpPixBuf + lumBufIndex + firstLumSrcY - lastInLumBuf + vLumBufSize : NULL;
            int16_t *vLumFilter = c->vLumFilter;
            int16_t *vChrFilter = c->vChrFilter;

            if (isPlanarYUV(dstFormat) ||
                (isGray(dstFormat) && !isALPHA(dstFormat))) { // YV12 like
                const int chrSkipMask = (1 << c->chrDstVSubSample) - 1;

                vLumFilter +=    dstY * vLumFilterSize;
                vChrFilter += chrDstY * vChrFilterSize;

//                av_assert0(use_mmx_vfilter != (
//                                   yuv2planeX == yuv2planeX_10BE_c
//                               || yuv2planeX == yuv2planeX_10LE_c
//                               || yuv2planeX == yuv2planeX_9BE_c
//                               || yuv2planeX == yuv2planeX_9LE_c
//                               || yuv2planeX == yuv2planeX_16BE_c
//                               || yuv2planeX == yuv2planeX_16LE_c
//                               || yuv2planeX == yuv2planeX_8_c) || !ARCH_X86);
```

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

```
        }
    }
    if (isPlanar(dstFormat) && isALPHA(dstFormat) && !alpPixBuf) {
        int length = dstW;
        int height = dstY - lastDstY;

        if (is16BPS(dstFormat) || isNBPS(dstFormat)) {
            const AVPixFmtDescriptor *desc = av_pix_fmt_desc_get(dstFormat);
            fillPlane16(dst[3], dstStride[3], length, height, lastDstY,
                    1, desc->comp[3].depth_minus1,
                    isBE(dstFormat));
        } else
            fillPlane(dst[3], dstStride[3], length, height, lastDstY, 255);
    }

#if HAVE_MMXEXT_INLINE
    if (av_get_cpu_flags() & AV_CPU_FLAG_MMXEXT)
        __asm__ volatile ("sfence" ::: "memory");
#endif
    emms_c();

    /* store changed local vars back in the context */
    c->dstY         = dstY;
    c->lumBufIndex  = lumBufIndex;
    c->chrBufIndex  = chrBufIndex;
    c->lastInLumBuf = lastInLumBuf;
    c->lastInChrBuf = lastInChrBuf;

    return dstY - lastDstY;
}
```

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

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

下面具体看看这几个函数的定义。

hyscale()

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

```cpp
// *** horizontal scale Y line to temp buffer
static av_always_inline void hyscale(SwsContext *c, int16_t *dst, int dstWidth,
                                     const uint8_t *src_in[4],
                                     int srcW, int xInc,
                                     const int16_t *hLumFilter,
                                     const int32_t *hLumFilterPos,
                                     int hLumFilterSize,
                                     uint8_t *formatConvBuffer,
                                     uint32_t *pal, int isAlpha)
{
    void (*toYV12)(uint8_t *, const uint8_t *, const uint8_t *, const uint8_t *, int, uint32_t *) =
        isAlpha ? c->alpToYV12 : c->lumToYV12;
    void (*convertRange)(int16_t *, int) = isAlpha ? NULL : c->lumConvertRange;
    const uint8_t *src = src_in[isAlpha ? 3 : 0];

    if (toYV12) {
        toYV12(formatConvBuffer, src, src_in[1], src_in[2], srcW, pal);
        src = formatConvBuffer;
    } else if (c->readLumPlanar && !isAlpha) {
        //读取
        c->readLumPlanar(formatConvBuffer, src_in, srcW, c->input_rgb2yuv_table);
        //赋值
        src = formatConvBuffer;
    } else if (c->readAlpPlanar && isAlpha) {
        c->readAlpPlanar(formatConvBuffer, src_in, srcW, NULL);
        src = formatConvBuffer;
    }

    if (!c->hyscale_fast) {
        //亮度-水平拉伸
        c->hyScale(c, dst, dstWidth, src, hLumFilter,
                   hLumFilterPos, hLumFilterSize);
    } else { // fast bilinear upscale / crap downscale
        c->hyscale_fast(c, dst, dstWidth, src, srcW, xInc);
    }
    //如果需要取值范围的转换（0-255和16-235之间）
    if (convertRange)
        convertRange(dst, dstWidth);
}
```

从hyscale()的源代码可以看出，它的流程如下所示。
1.转换成Y（亮度）
如果SwsContext的toYV12()函数存在，调用用该函数将数据转换为Y。如果该函数不存在，则调用SwsContext的readLumPlanar()读取Y。
2.拉伸
拉伸通过SwsContext的hyScale ()函数完成。如果存在hyscale_fast()方法的话，系统会优先调用hyscale_fast()。
3.转换范围（如果需要的话）
如果需要转换亮度的取值范围（例如需要进行16-235的MPEG标准与0-255的JPEG标准之间的转换），则会调用SwsContext的lumConvertRange ()函数。
上述几个步骤的涉及到的函数在上一篇文章中几经介绍过了，在这里重复一下。

toYV12() [SwsContext ->lumToYV12()]

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

输入格式为YUYV422/ YVYU422

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

```cpp
case AV_PIX_FMT_YUYV422:
case AV_PIX_FMT_YVYU422:
case AV_PIX_FMT_YA8:
    c->lumToYV12 = yuy2ToY_c;
    break;
```

yuy2ToY_c()的定义如下所示。

```cpp
static void yuy2ToY_c(uint8_t *dst, const uint8_t *src, const uint8_t *unused1, const uint8_t *unused2,  int width,
                      uint32_t *unused)
{
    int i;
    for (i = 0; i < width; i++)
        dst[i] = src[2 * i];
}
```

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

输入格式为RGB24

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

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

rgb24ToY_c()的定义如下所示。

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

SwsContext -> hyScale ()

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

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

hScale8To15_c()

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

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

lumConvertRange () [SwsContext -> lumConvertRange()]

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

```cpp
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;
            }
        }
    }
}
```

SwsContext 的lumConvertRange()函数主要用于JPEG标准像素取值范围（0-255）和MPEG标准像素取值范围（16-235）之间的转换。有关这方面的分析在上一篇文章中一斤详细叙述过，在这里不再重复。简单看一下其中的一个函数。

lumRangeFromJpeg_c()

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

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

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

hcscale()

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

```cpp
static av_always_inline void hcscale(SwsContext *c, int16_t *dst1,
                                     int16_t *dst2, int dstWidth,
                                     const uint8_t *src_in[4],
                                     int srcW, int xInc,
                                     const int16_t *hChrFilter,
                                     const int32_t *hChrFilterPos,
                                     int hChrFilterSize,
                                     uint8_t *formatConvBuffer, uint32_t *pal)
{
    const uint8_t *src1 = src_in[1], *src2 = src_in[2];
    if (c->chrToYV12) {
        uint8_t *buf2 = formatConvBuffer +
                        FFALIGN(srcW*2+78, 16);
        //转换
        c->chrToYV12(formatConvBuffer, buf2, src_in[0], src1, src2, srcW, pal);
        src1= formatConvBuffer;
        src2= buf2;
    } else if (c->readChrPlanar) {
        uint8_t *buf2 = formatConvBuffer +
                        FFALIGN(srcW*2+78, 16);
        //读取
        c->readChrPlanar(formatConvBuffer, buf2, src_in, srcW, c->input_rgb2yuv_table);
        //赋值
        src1 = formatConvBuffer;
        src2 = buf2;
    }

    if (!c->hcscale_fast) {
        //色度-水平拉伸
        c->hcScale(c, dst1, dstWidth, src1, hChrFilter, hChrFilterPos, hChrFilterSize);
        c->hcScale(c, dst2, dstWidth, src2, hChrFilter, hChrFilterPos, hChrFilterSize);
    } else { // fast bilinear upscale / crap downscale
        c->hcscale_fast(c, dst1, dst2, dstWidth, src1, src2, srcW, xInc);
    }
    //如果需要取值范围的转换（0-255和16-235之间）
    if (c->chrConvertRange)
        c->chrConvertRange(dst1, dst2, dstWidth);
}
```

从hcscale()的源代码可以看出，它的流程如下所示。

1.转换成UV

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

2.拉伸

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

3.转换范围（如果需要的话）

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

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

还有几个函数没有分析，但是时间有限，以后有机会再进行补充。

雷霄骅

leixiaohua1020@126.com

http://blog.csdn.net/leixiaohua1020

文章标签： FFmpeg swscale YUV RGB 源代码

个人分类： FFMPEG

所属专栏： FFmpeg