

原 ffmpeg 源代码简单分析：avcodec_decode_video2()

2013年10月13日 16:07:23 阅读数：54614

=====

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

=====

ffmpeg中的avcodec_decode_video2()的作用是解码一帧视频数据。输入一个压缩编码的结构体AVPacket，输出一个解码后的结构体AVFrame。该函数的声明位于libavcodec\avcodec.h，如下所示。

```
[cpp]
1.  /**
2.   * Decode the video frame of size avpkt->size from avpkt->data into picture.
3.   * Some decoders may support multiple frames in a single AVPacket, such
4.   * decoders would then just decode the first frame.
5.   *
6.   * @warning The input buffer must be FF_INPUT_BUFFER_PADDING_SIZE larger than
7.   * the actual read bytes because some optimized bitstream readers read 32 or 64
8.   * bits at once and could read over the end.
9.   *
10.  * @warning The end of the input buffer buf should be set to 0 to ensure that
11.  * no overreading happens for damaged MPEG streams.
12.  *
13.  * @note Codecs which have the CODEC_CAP_DELAY capability set have a delay
14.  * between input and output, these need to be fed with avpkt->data=NULL,
15.  * avpkt->size=0 at the end to return the remaining frames.
16.  *
17.  * @param avctx the codec context
18.  * @param[out] picture The AVFrame in which the decoded video frame will be stored.
19.  * Use av_frame_alloc() to get an AVFrame. The codec will
20.  * allocate memory for the actual bitmap by calling the
21.  * AVCodecContext.get_buffer2() callback.
22.  * When AVCodecContext.refcounted_frames is set to 1, the frame is
23.  * reference counted and the returned reference belongs to the
24.  * caller. The caller must release the frame using av_frame_unref()
25.  * when the frame is no longer needed. The caller may safely write
26.  * to the frame if av_frame_is_writable() returns 1.
27.  * When AVCodecContext.refcounted_frames is set to 0, the returned
28.  * reference belongs to the decoder and is valid only until the
29.  * next call to this function or until closing or flushing the
30.  * decoder. The caller may not write to it.
31.  *
32.  * @param[in] avpkt The input AVPacket containing the input buffer.
33.  * You can create such packet with av_init_packet() and by then setting
34.  * data and size, some decoders might in addition need other fields like
35.  * flags&AV_PKT_FLAG_KEY. All decoders are designed to use the least
36.  * fields possible.
37.  * @param[in,out] got_picture_ptr Zero if no frame could be decompressed, otherwise, it is nonzero.
38.  * @return On error a negative value is returned, otherwise the number of bytes
39.  * used or zero if no frame could be decompressed.
40.  */
41. int avcodec_decode_video2(AVCodecContext *avctx, AVFrame *picture,
42.                           int *got_picture_ptr,
43.                           const AVPacket *avpkt);
```

查看源代码之后发现，这个函数竟然十分的简单，源代码位于libavcodec\utils.c，如下所示：

```
[cpp]
1.  int attribute_align_arg avcodec_decode_video2(AVCodecContext *avctx, AVFrame *picture,
2.                                                int *got_picture_ptr,
3.                                                const AVPacket *avpkt)
4.  {
5.      AVCodecInternal *avci = avctx->internal;
6.      int ret;
7.      // copy to ensure we do not change avpkt
8.      AVPacket tmp = *avpkt;
9.
10.     if (!avctx->codec)
11.         return AVERROR(EINVAL);
12.     //检查是不是视频（非音频）
13.     if (avctx->codec->type != AVMEDIA_TYPE_VIDEO) {
14.         av_log(avctx, AV_LOG_ERROR, "Invalid media type for video\n");
15.         return AVERROR(EINVAL);
16.     }
17.
18.     *got_picture_ptr = 0;
19.     //检查宽、高设置是否正确
20.     if ((avctx->coded_width || avctx->coded_height) && av_image_check_size(avctx->coded_width, avctx->coded_height, 0, avctx))
21.         return AVERROR(EINVAL);
22. }
```

```

22.
23.     av_frame_unref(picture);
24.
25.     if ((avctx->codec->capabilities & CODEC_CAP_DELAY) || avpkt->size || (avctx->active_thread_type & FF_THREAD_FRAME)) {
26.         int did_split = av_packet_split_side_data(&tmp);
27.         ret = apply_param_change(avctx, &tmp);
28.         if (ret < 0) {
29.             av_log(avctx, AV_LOG_ERROR, "Error applying parameter changes.\n");
30.             if (avctx->err_recognition & AV_EF_EXPLODE)
31.                 goto fail;
32.         }
33.
34.         avctx->internal->pkt = &tmp;
35.         if (HAVE_THREADS && avctx->active_thread_type & FF_THREAD_FRAME)
36.             ret = ff_thread_decode_frame(avctx, picture, got_picture_ptr,
37.                                         &tmp);
38.         else {
39.             // 最关键的解码函数
40.             ret = avctx->codec->decode(avctx, picture, got_picture_ptr,
41.                                       &tmp);
42.             // 设置pkt_dts字段的值
43.             picture->pkt_dts = avpkt->dts;
44.
45.             if (!avctx->has_b_frames) {
46.                 av_frame_set_pkt_pos(picture, avpkt->pos);
47.             }
48.             //FIXME these should be under if(!avctx->has_b_frames)
49.             /* get_buffer is supposed to set frame parameters */
50.             if (!(avctx->codec->capabilities & CODEC_CAP_DR1)) {
51.                 // 对一些字段进行赋值
52.                 if (!picture->sample_aspect_ratio.num)    picture->sample_aspect_ratio = avctx->sample_aspect_ratio;
53.                 if (!picture->width)                      picture->width = avctx->width;
54.                 if (!picture->height)                    picture->height = avctx->height;
55.                 if (picture->format == AV_PIX_FMT_NONE)   picture->format = avctx->pix_fmt;
56.             }
57.         }
58.         add_metadata_from_side_data(avctx, picture);
59.
60. fail:
61.         emms_c(); //needed to avoid an emms_c() call before every return;
62.
63.         avctx->internal->pkt = NULL;
64.         if (did_split) {
65.             av_packet_free_side_data(&tmp);
66.             if (ret == tmp.size)
67.                 ret = avpkt->size;
68.         }
69.
70.         if (*got_picture_ptr) {
71.             if (!avctx->refcounted_frames) {
72.                 int err = unrefcount_frame(avci, picture);
73.                 if (err < 0)
74.                     return err;
75.             }
76.
77.             avctx->frame_number++;
78.             av_frame_set_best_effort_timestamp(picture,
79.                                               guess_correct_pts(avctx,
80.                                                                picture->pkt_pts,
81.                                                                picture->pkt_dts));
82.         } else
83.             av_frame_unref(picture);
84.     } else
85.         ret = 0;
86.
87.     /* many decoders assign whole AVFrames, thus overwriting extended_data;
88.      * make sure it's set correctly */
89.     av_assert0(!picture->extended_data || picture->extended_data == picture->data);
90.
91. #if FF_API_AVCTX_TIMEBASE
92.     if (avctx->framerate.num > 0 && avctx->framerate.den > 0)
93.         avctx->time_base = av_inv_q(av_mul_q(avctx->framerate, (AVRational){avctx->ticks_per_frame, 1}));
94. #endif
95.
96.     return ret;
97. }

```

从代码中可以看出，avcodec_decode_video2()主要做了以下几个方面的工作：

- (1) 对输入的字进行了一系列的检查工作：例如宽高是否正确，输入是否为视频等等。
- (2) 通过ret = avctx->codec->decode(avctx, picture, got_picture_ptr,&tmp)这句代码，调用了相应AVCodec的decode()函数，完成了解码操作。
- (3) 对得到的AVFrame的一些字段进行了赋值，例如宽高、像素格式等等。

其中第二步是关键的一步，它调用了AVCodec的decode()方法完成了解码。AVCodec的decode()方法是一个函数指针，指向了具体解码器的解码函数。在这里我们以H.264解码器为例，看一下解码的实现过程。H.264解码器对应的AVCodec的定义位于libavcodec/h264.c，如下所示。

```

1. AVCodec ff_h264_decoder = {
2.     .name           = "h264",
3.     .long_name      = NULL_IF_CONFIG_SMALL("H.264 / AVC / MPEG-4 AVC / MPEG-4 part 10"),
4.     .type           = AVMEDIA_TYPE_VIDEO,
5.     .id             = AV_CODEC_ID_H264,
6.     .priv_data_size = sizeof(H264Context),
7.     .init            = ff_h264_decode_init,
8.     .close           = h264_decode_end,
9.     .decode          = h264_decode_frame,
10.    .capabilities     = /*CODEC_CAP_DRAW_HORIZ_BAND |*/ CODEC_CAP_DR1 |
11.                       CODEC_CAP_DELAY | CODEC_CAP_SLICE_THREADS |
12.                       CODEC_CAP_FRAME_THREADS,
13.    .flush            = flush_dpb,
14.    .init_thread_copy = ONLY_IF_THREADS_ENABLED(decode_init_thread_copy),
15.    .update_thread_context = ONLY_IF_THREADS_ENABLED(ff_h264_update_thread_context),
16.    .profiles         = NULL_IF_CONFIG_SMALL(profiles),
17.    .priv_class       = &h264_class,
18. };

```

从ff_h264_decoder的定义可以看出，decode()指向了h264_decode_frame()函数。继续看一下h264_decode_frame()函数的定义，如下所示。

```

1. static int h264_decode_frame(AVCodecContext *avctx, void *data,
2.                             int *got_frame, AVPacket *avpkt)
3. {
4.     const uint8_t *buf = avpkt->data;
5.     int buf_size      = avpkt->size;
6.     H264Context *h    = avctx->priv_data;
7.     AVFrame *pict      = data;
8.     int buf_index      = 0;
9.     H264Picture *out;
10.    int i, out_idx;
11.    int ret;
12.
13.    h->flags = avctx->flags;
14.    /* reset data partitioning here, to ensure GetBitContexts from previous
15.     * packets do not get used. */
16.    h->data_partitioning = 0;
17.
18.    /* end of stream, output what is still in the buffers */
19.    if (buf_size == 0) {
20.        out:
21.
22.        h->cur_pic_ptr = NULL;
23.        h->first_field = 0;
24.
25.        /* FIXME factorize this with the output code below
26.         out  = h->delayed_pic[0];
27.         out_idx = 0;
28.         for (i = 1;
29.             h->delayed_pic[i] &&
30.             !h->delayed_pic[i]->f.key_frame &&
31.             !h->delayed_pic[i]->mmco_reset;
32.             i++)
33.             if (h->delayed_pic[i]->poc < out->poc) {
34.                 out  = h->delayed_pic[i];
35.                 out_idx = i;
36.             }
37.
38.         for (i = out_idx; h->delayed_pic[i]; i++)
39.             h->delayed_pic[i] = h->delayed_pic[i + 1];
40.
41.         if (out) {
42.             out->reference &= ~DELAYED_PIC_REF;
43.             ret = output_frame(h, pict, out);
44.             if (ret < 0)
45.                 return ret;
46.             *got_frame = 1;
47.         }
48.
49.         return buf_index;
50.     }
51.     if (h->is_avc && av_packet_get_side_data(avpkt, AV_PKT_DATA_NEW_EXTRADATA, NULL)) {
52.         int side_size;
53.         uint8_t *side = av_packet_get_side_data(avpkt, AV_PKT_DATA_NEW_EXTRADATA, &side_size);
54.         if (is_extra(side, side_size))
55.             ff_h264_decode_extradata(h, side, side_size);
56.     }
57.     if (h->is_avc && buf_size >= 9 && buf[0]==1 && buf[2]==0 && (buf[4]&0xFC)==0xFC && (buf[5]&0x1F) && buf[8]==0x67){
58.         if (is_extra(buf, buf_size))
59.             return ff_h264_decode_extradata(h, buf, buf_size);
60.     }
61.    /*H.264解码
62.    buf_index = decode_nal_units(h, buf, buf_size, 0);
63.    if (buf_index < 0)
64.        return AVERROR_INVALIDDATA;
65.
66.    if (h->is_avc && buf_size >= 9 && buf[0]==1 && buf[2]==0 && (buf[4]&0xFC)==0xFC && (buf[5]&0x1F) && buf[8]==0x67){

```

```

66.     if (!h->cur_pic_ptr && h->nal_unit_type == NAL_END_SEQUENCE) {
67.         av_assert0(buf_index <= buf_size);
68.         goto out;
69.     }
70.
71.     if (!(avctx->flags2 & CODEC_FLAG2_CHUNKS) && !h->cur_pic_ptr) {
72.         if (avctx->skip_frame >= AVDISCARD_NONREF ||
73.             buf_size >= 4 && !memcmp("Q264", buf, 4))
74.             return buf_size;
75.         av_log(avctx, AV_LOG_ERROR, "no frame!\n");
76.         return AVERROR_INVALIDDATA;
77.     }
78.
79.     if (!(avctx->flags2 & CODEC_FLAG2_CHUNKS) ||
80.         (h->mb_y >= h->mb_height && h->mb_height)) {
81.         if (avctx->flags2 & CODEC_FLAG2_CHUNKS)
82.             decode_postinit(h, 1);
83.
84.         ff_h264_field_end(h, 0);
85.
86.         /* Wait for second field. */
87.         *got_frame = 0;
88.         if (h->next_output_pic && (
89.             h->next_output_pic->recovered)) {
90.             if (!h->next_output_pic->recovered)
91.                 h->next_output_pic->f.flags |= AV_FRAME_FLAG_CORRUPT;
92.
93.             ret = output_frame(h, pict, h->next_output_pic);
94.             if (ret < 0)
95.                 return ret;
96.             *got_frame = 1;
97.             if (CONFIG_MPEGVIDEO) {
98.                 ff_print_debug_info2(h->avctx, pict, h->er.mbskip_table,
99.                                     h->next_output_pic->mb_type,
100.                                    h->next_output_pic->qscale_table,
101.                                    h->next_output_pic->motion_val,
102.                                    &h->low_delay,
103.                                    h->mb_width, h->mb_height, h->mb_stride, 1);
104.             }
105.         }
106.     }
107.
108.     assert(pict->buf[0] || !*got_frame);
109.
110.     return get_consumed_bytes(buf_index, buf_size);
111. }

```

从h264_decode_frame()的定义可以看出，它调用了decode_nal_units()完成了具体的H.264解码工作。有关H.264解码就不在详细分析了。

版权声明：本文为博主原创文章，未经博主允许不得转载。 <https://blog.csdn.net/leixiaohua1020/article/details/12679719>

文章标签：[ffmpeg](#) [源代码简单](#) [avcodec_decode_video](#)

个人分类：[FFMPEG](#)

所属专栏：[开源多媒体项目源代码分析](#) [FFmpeg](#)

此PDF由spygg生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com