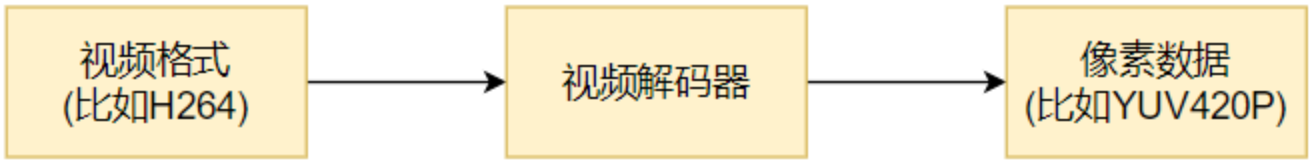
视频解码实战

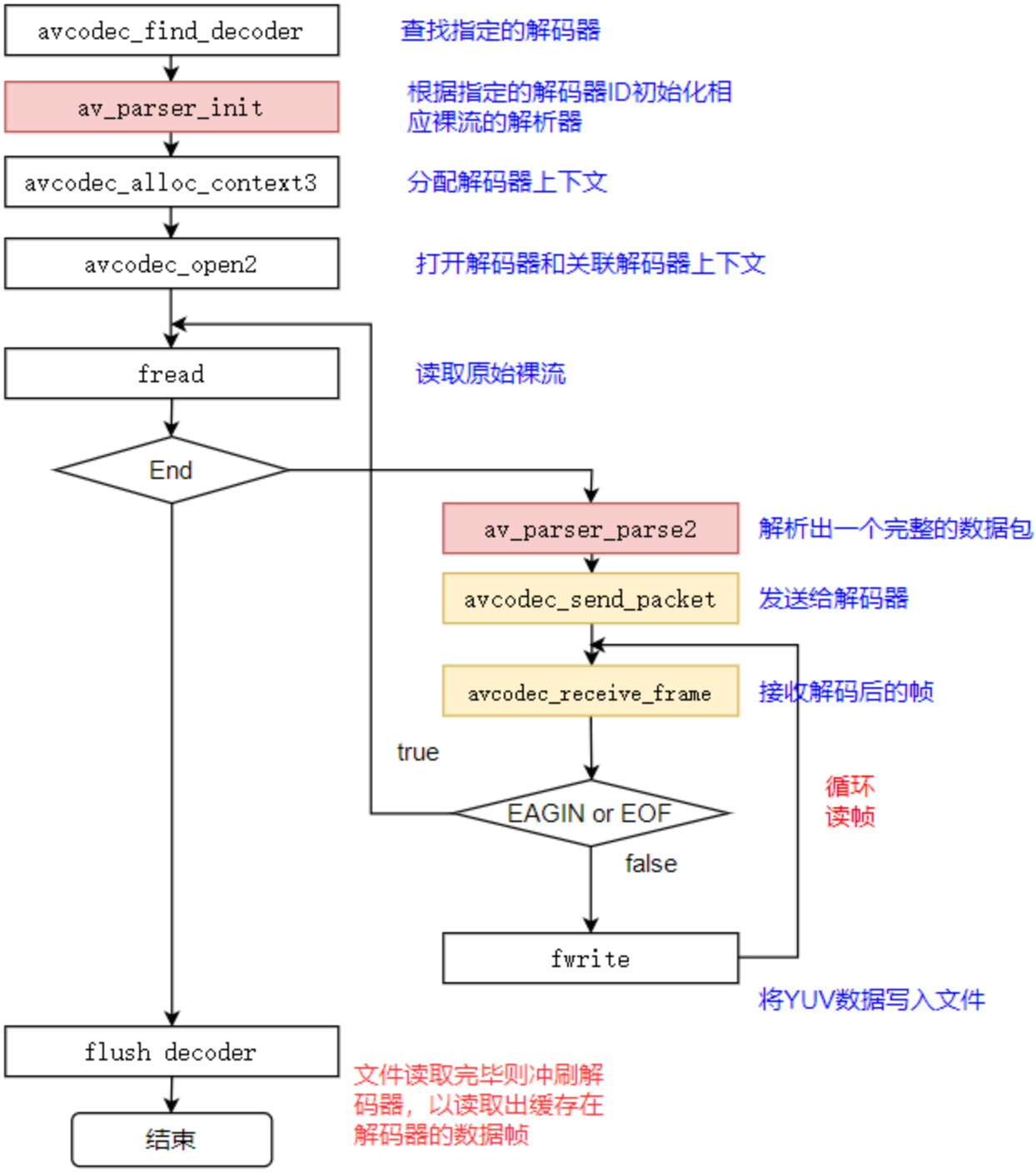
视频解码对于FFmpeg而言，流程基本上和音频解码一致。
对于音频的resample和视频的rescale在音视频合成输出再做讲解。

视频解码过程

视频解码过程如下图所示：
一般解出来的是420p



FFmpeg流程



关键函数

关键函数说明：

- avcodec_find_decoder：根据指定的AVCodecID查找注册的解码器。
- av_parser_init：初始化AVCodecParserContext。
- avcodec_alloc_context3：为AVCodecContext分配内存。
- avcodec_open2：打开解码器。
- av_parser_parse2：解析获得一个Packet。

- avcodec_send_packet：将AVPacket压缩数据给解码器。
- avcodec_receive_frame：获取到解码后的AVFrame数据。
- av_get_bytes_per_sample: 获取每个sample中的字节数。

关键数据结构

关键数据结构说明：

- AVCodecParser：用于解析输入的数据流并把它分成一帧一帧的压缩编码数据。比较形象的说法就是把长长的一段连续的数据“切割”成一段段的数据。

比如H264 aac_parser

ffmpeg-4.2.1\libavcodec\h264_parser.c

```
AVCodecParser ff_h264_parser = {
    .codec_ids      = { AV_CODEC_ID_H264 },
    .priv_data_size = sizeof(H264ParseContext),
    .parser_init    = init,
    .parser_parse   = h264_parse,
    .parser_close   = h264_close,
    .split          = h264_split,
};
```

从AVCodecParser结构的实例化我们可以看出来，不同编码类型的parser是和CODE_ID进行绑定的。所以也就可以解释

```
parser = av_parser_init(AV_CODEC_ID_H264);
```

可以通过CODE_ID查找到对应的码流 parser。

avcodec编解码API介绍

avcodec_send_packet、avcodec_receive_frame的API是FFmpeg3版本加入的。为了正确的使用它们，有必要阅读FFmpeg的文档说明(请点击链接)。

以下内容摘译自文档说明

FFmpeg提供了两组函数，分别用于编码和解码：

- 解码：avcodec_send_packet()、avcodec_receive_frame()。
- 解码：avcodec_send_frame()、avcodec_receive_packet()。

API的设计与编解码的流程非常贴切。

建议的使用流程如下：

1. 像以前一样设置并打开AVCodecContext。
2. 输入有效的数据：
   - 解码：调用avcodec_send_packet()给解码器传入包含原始的压缩数据的AVPacket对象。
   - 编码：调用 avcodec_send_frame()给编码器传入包含解压数据的AVFrame对象。

   两种情况下推荐AVPacket和AVFrame都使用refcounted（引用计数）的模式，否则libavcodec可能不得不对输入的数据进行拷贝。
3. 在一个循环体内去接收codec的输出，即周期性地调用avcodec_receive_*()来接收codec输出的数据：
   - 解码：调用avcodec_receive_frame()，如果成功会返回一个包含未压缩数据的AVFrame。
   - 编码：调用avcodec_receive_packet()，如果成功会返回一个包含压缩数据的AVPacket。

   反复地调用avcodec_receive_packet()**直到返回 AVERROR(EAGAIN)或其他错误**。返回AVERROR(EAGAIN)错误表示codec需要新的输入来输出更多的数据。对于每个输入的packet或frame，codec一般会输出一个frame或packet，但是也有可能输出0个或者多于1个。
4. 流处理结束的时候需要flush（冲刷） codec。因为codec可能在内部缓冲多个frame或packet，出于性能或其他必要的情况（如考虑B帧的情况）。 处理流程如下：
   - 调用avcodec_send_*()传入的AVFrame或AVPacket指针设置为NULL。 这将进入draining mode（排水模式）。
   - 反复地调用avcodec_receive_*()直到返回AVERROR_EOF，该方法在draining mode时不会返回AVERROR(EAGAIN)的错误，除非你没有进入draining mode。
   - 当重新开启codec时，需要先调用 avcodec_flush_buffers()来重置codec。

说明：

1. 编码或者解码刚开始的时候，codec可能接收了多个输入的frame或packet后还没有输出数据，直到内部的buffer被填充满。上面的使用流程可以处理这种情况。
2. 理论上，只有在输出数据没有被完全接收的情况调用avcodec_send_*()的时候才可能会发生AVERROR(EAGAIN)的错误。你可以依赖这个机制来实现区别于上面建议流程的处理方式，比如每次循环都调用avcodec_send_*()，在出现AVERROR(EAGAIN)错误的时候再去调用avcodec_receive_*()。
3. 并不是所有的codec都遵循一个严格、可预测的数据处理流程，唯一可以保证的是**“调用avcodec_send_*()/avcodec_receive_*()返回AVERROR(EAGAIN)的时候去avcodec_receive_*()/avcodec_send_*()会成功，否则不应该返回AVERROR(EAGAIN)的错误。”**一般来说，任何codec都不允许无限制地缓存输入或者输出。
4. 在同一个AVCodecContext上混合使用新旧API是不允许的，这将导致未定义的行为。

avcodec_send_packet

函数： int avcodec_send_packet(AVCodecContext *avctx, const AVPacket *avpkt);

作用：支持将裸流数据包送给解码器

警告：

- 输入的avpkt->data缓冲区必须大于AV_INPUT_PADDING_SIZE，因为优化的字节流读取器必须一次读取32或者64比特的数据
- 不能跟之前的API(例如avcodec_decode_video2)混用，否则会返回不可预知的错误

备注：

- 在将包发送给解码器的时候，AVCodecContext必须已经通过avcodec_open2打开

参数：

- avctx：解码上下文
- avpkt：输入AVPakcet.通常情况下，输入数据是一个单一的视频帧或者几个完整的音频帧。调用者保留包的原有属性，解码器不会修改包的内容。解码器可能创建对包的引用。如果包没有引用计数将拷贝一份。跟以往的API不一样，输入的包的数据将被完全地消耗，如果包含有多个帧，要求多次调用avcodec_recvive_frame，直到avcodec_recvive_frame返回VERROR(EAGAIN)或AVERROR_EOF。输入参数可以为NULL，或者AVPacket的data域设置为NULL或者size域设置为0，表示将刷新所有的包，意味着数据流已经结束了。第一次发送刷新会总会成功，第二次发送刷新包是没有必要的，并且返回AVERROR_EOF,如果×××缓存了一些帧，返回一个刷新包，将会返回所有的解码包

返回值：

- 0: 表示成功
- AVERROR(EAGAIN)：当前状态不接受输入，用户必须先使用avcodec_receive_frame() 读取数据帧；
- AVERROR_EOF：解码器已刷新，不能再向其发送新包；
- AVERROR(EINVAL)：没有打开解码器，或者这是一个编码器，或者要求刷新；
- AVERRO(ENOMEN)：无法将数据包添加到内部队列。

avcodec_receive_frame

函数： int avcodec_receive_frame (AVCodecContext * avctx, AVFrame * frame)

作用： 从解码器返回已解码的输出数据。

参数：

- avctx: 编解码器上下文
- frame: 获取使用reference-counted机制的audio或者video帧（取决于解码器类型）。请注意，在执行其他操作之前，函数内部将始终先调用av_frame_unref(frame)。

返回值：

- 0: 成功，返回一个帧
- AVERROR(EAGAIN): 该状态下没有帧输出，需要使用avcodec_send_packet发送新的packet到解码器
- AVERROR_EOF: 解码器已经被完全刷新，不再有输出帧
- AVERROR(EINVAL): 编解码器没打开
- 其他<0的值: 具体查看对应的错误码

附录

分离H264或mpeg2video视频格式数据

提取H264:

ffmpeg –i source.200kbps.768x320_10s.flv –vcodec libx264 –an –f h264 source.200kbps.768x320_10s.h264

提取MPEG2:

ffmpeg –i source.200kbps.768x320_10s.flv –vcodec mpeg2video –an –f mpeg2video source.200kbps.768x320_10s.mpeg2v

播放YUV

播放：

ffplay –pixel_format yuv420p –video_size 768x320 –framerate 25 source.200kbps.768x320_10s.yuv

FFmpeg命令查找重定向

比如我们在–f fmt打算指定格式时，怎么知道什么样的格式才是适合的format?

可以通过ffmpeg –formats | findstr xx的方式去查找。

对于findstr，/i是忽略大小写

比如：

查找Audio的裸流解复用器：ffmpeg –formats | findstr /i audio

查找Video的裸流解复用器：ffmpeg –formats | findstr /i video