ffmpeg

源码地址：https://github.com/FFmpeg/FFmpeg

库相关文档：https://ffmpeg.org/doxygen/trunk/index.html

库：

- `libavcodec` provides implementation of a wider range of codecs.
- `libavformat` implements streaming protocols, container formats and basic I/O access.
- `libavutil` includes hashers, decompressors and miscellaneous utility functions.
- `libavfilter` provides a mean to alter decoded Audio and Video through chain of filters.
- `libavdevice` provides an abstraction to access capture and playback devices.
- `libswresample` implements audio mixing and resampling routines.
- `libswscale` implements color conversion and scaling routines.

工具（可执行）

- ffmpeg is a command line toolbox to manipulate, convert and stream multimedia content.
- ffplay is a minimalistic multimedia player.
- ffprobe is a simple analysis tool to inspect multimedia content.

Part I

ffmpeg-libav-tutorial-github

- What is the **codec**: 编解码器
- What is the **container**: 视频、音频格式包，除了音视频数据外，还有**metadata**；

FFmpeg命令行参数

Official doc: https://www.ffmpeg.org/ffmpeg.html

Reference：http://slhck.info/ffmpeg-encoding-course/#/

https://github.com/leandromoreira/digital_video_introduction/blob/master/encoding_pratical_examples.md#split-and-merge-smoothly

- 基本格式：`$ ffmpeg -i input.mp4 output.avi` 实现简单的转码（容器类型转换）
- 概述：

```
ffmpeg {1} {2} -i {3} {4} {5}
1. global options
2. input file options
3. input url
4. output file options
5. output url
```

举例：

```
$ ffmpeg \
-y \ # global options
-c:a libfdk_aac -c:v libx264 \ # input options
-i bunny_1080p_60fps.mp4 \ # input url
-c:v libvpx-vp9 -c:a libvorbis \ # output options
bunny_1080p_60fps_vp9.webm # output url
```

视频相关操作

- 转码-transcoding：将audio **或** video中的格式转为另外的格式

```
$ ffmpeg \
-i bunny_1080p_60fps.mp4 \
-c:v libx265 \
bunny_1080p_60fps_h265.mp4
```

- 转码-transmuxing：container格式转换

```
$ ffmpeg \
-i bunny_1080p_60fps.mp4 \
-c copy \ # just saying to ffmpeg to skip encoding
bunny_1080p_60fps.webm
```

- 码率转换-transrating：转变比特率或渲染方式

Reference：https://slhck.info/posts/

Plain Text

```
$ ffmpeg \
-i bunny_1080p_60fps.mp4 \
-minrate 964K -maxrate 3856K -bufsize 2000K \
bunny_1080p_60fps_transrating_964_3856.mp4
```

- Transsizing：分辨率转换

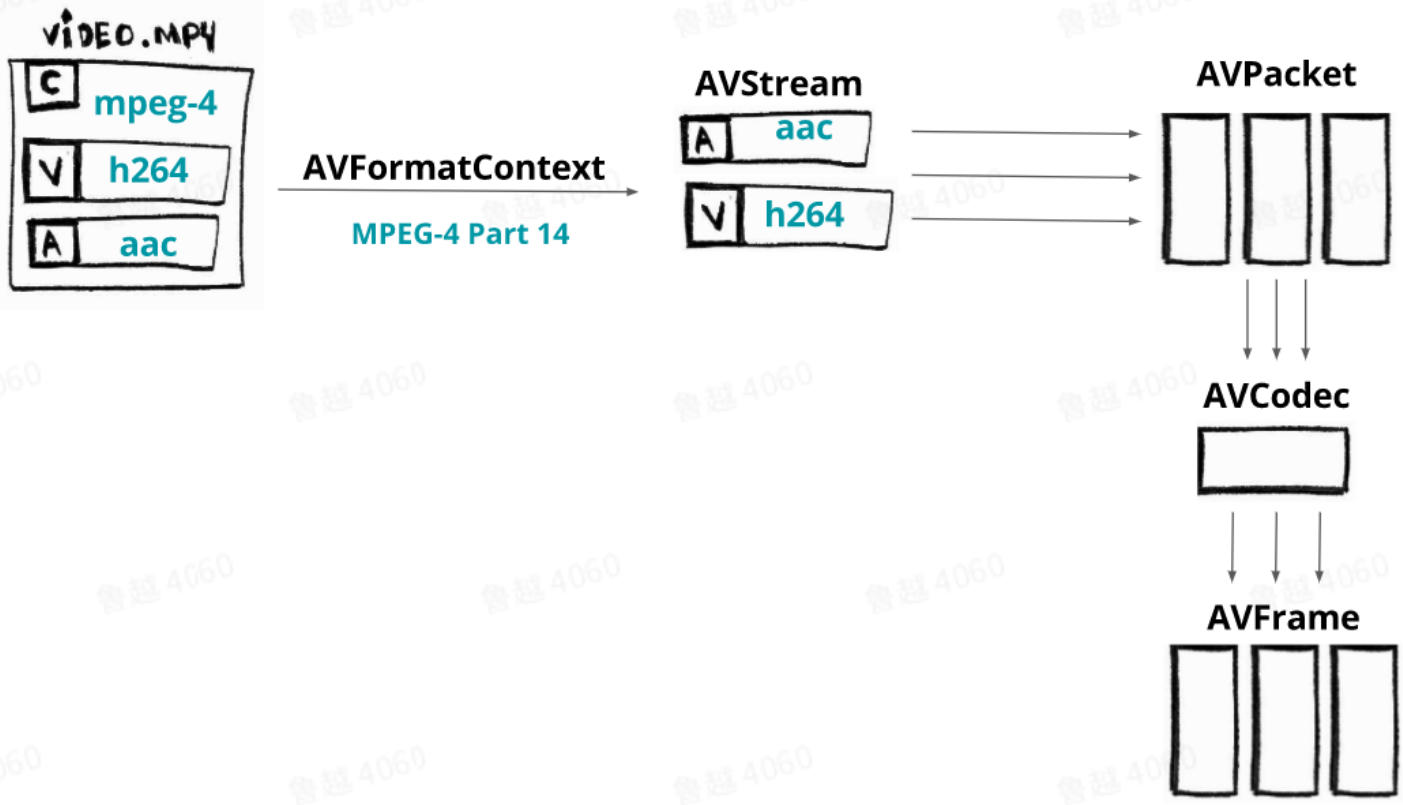
Plain Text

```
$ ffmpeg \
-i bunny_1080p_60fps.mp4 \
-vf scale=480:-1 \
bunny_1080p_60fps_transsizing_480.mp4
```

- 自适应流：the act of producing many resolutions (bit rates) and split the media into chunks and serve them via http.（参考：webm-wiki）

Hello-FFmpeg

Get head-info of container (AVFormatContext) -> av stream data -> extract slices of data into packages (AVPacket) -> decode data (AVCodec) -> obtain uncompressed frames （decode into AVFrame)



如何将ffmpeg相关的库链接到visual studio中?

1. 下载ffmpeg shared版本，其中包含动态链接库（链接可能会变化，目前网上许多以前的文章引用的链接已经打不开）；
2. 下载解压后的文件结构如下

```
Plain Text
```

```
bin: 包含运行所需dll以及三个可执行文件
    avcodec-58.dll
    avdevice-58.dll
    ...*.dll
    ffmpeg.exe
    ffplay.exe
    ffprobe.exe
include: 动态链接库对应的头文件
lib: import libraries (需要被声明的依赖项)
```

> use pre-built .lib/.dll files and your binary produced with Visual Studio will be dependent on av*.dll files (from Use FFmpeg in Visual Studio)

3. 配置visual studio

```
Plain Text
```

```
1. 添加包含路径:
project->properties->c/c++->general->additional include dir-> +<path of include>
2. 添加import libraries路径
...->linker->general->additional library dir-> +<path of lib>
3. 添加依赖项
...->linker->input->additional dependencies-> +<*.lib>
4. copy dll文件到当前project的路径下。（可以用脚本实现？待研究）
```

关于dll配置到工程中的四步，参考Walkthrough: Create and use your own Dynamic Link Library (C++). 关于vs属性设置中依赖性和附加依赖项，包含路径与附加包含路径的区别可以参考：Visual Studio中C++的包含目录、附加包含目录和库目录和附加库目录的区别_MIss-Y的博客-CSDN博客

4. 一般c++工程生成的dll，按照以上配置即可成功运行程序，但这里编译会不通过，提示“无法解析的外部符号”，根源在于ffmpeg相关头文件以及dll本身是以c语言生成的，同时也没有像一般标准库那样考虑到自身可能被包含到c++工程中（与可执行文件同路径），因此在包含都文件时需要添加（举例）

C++

```cpp
// 不要以c++风格rename函数名
extern "C" {
    #include <libavcodec/avcodec.h>
    #include <libavformat/avformat.h>
}
// 路径中寻找 avcodec.lib
#pragma comment(lib, "avcodec.lib")
#pragma comment(lib, "avformat.lib")
```

以上参考Why do we need extern "C"{ #include <foo.h> } in C++? , C++: What does #pragma comment(lib, "XXX") actually do with "XXX"? and Use FFmpeg in Visual Studio.

5. 注意x64/x32

第一个ffmpeg程序

- 基本功能：读取视频文件，按照基本流程解码，对于video数据，保存八帧，并转为灰度图像（pgm格式）
- 源码分析：https://github.com/leandromoreira/ffmpeg-libav-tutorial/blob/master/0_hello_world.c
- ffmpeg结构体解析：FFMPEG中最关键的结构体之间的关系_雷霄骅(leixiaohua1020)的专栏-CSDN博客_ffmpeg 结构体

1. 读取视频文件的头部及metadata（libavformat相关，AVFormatContext结构贯穿始终）：

> AVFormatContext是包含码流参数较多的结构体。

C++

```cpp
// 1. 预先分配空间
AVFormatContext* pFormatContext = avformat_alloc_context();

// 2. 打开文件，读取header
// metadate保存在了pFormatContext结构体内
avformat_open_input(&pFormatContext, argv[1], NULL, NULL);

// 3. 读取流数据
// 相关信息存到AVFormatContext结构中的nb_streams以及streams
avformat_find_stream_info(pFormatContext, NULL)
```

What are muxing and demuxing:

> Multiplexing means combining different types of data in a single stream or file. On this board, they're talking about combining the video and audio data into a single file. Demultiplexing means splitting the video and audio out into separate files.

2. 读取流数据，获得解码信息（AVCodec, AVCodecParameters）

> AVCodec是存储编解码器信息的结构体

C++

```
// 循环，处理每一流
AVCodec* pCodec = NULL;
for (int i = 0; i < pFormatContext->nb_streams; i++){...}

// 循环内
// 1. 从AVFormatContext中取出codec参数信息
AVCodecParameters* pLocalCodecParameters = pFormatContext->streams[i]->codecpar;

// 2. 根据codec id 获取解码器
//    codec id表示编码方式，map到对应解码器
AVCodec* pLocalCodec = NULL;
pLocalCodec = avcodec_find_decoder(pLocalCodecParameters->codec_id);

// 3. 保存video解码相关信息，以及对应流的index
```

3. 解码准备（建立AVCodecContext）

C++

```
// 1. Allocate an AVCodecContext and set its fields to default values.
AVCodecContext* pCodecContext = avcodec_alloc_context3(pCodec);

// 2. 根据codec-parameters填充
avcodec_parameters_to_context(pCodecContext, pCodecParameters);

// 3. Initialize the AVCodecContext to use the given AVCodec.
avcodec_open2(pCodecContext, pCodec, NULL);
```

4. 解码（AVPacket and AVFrame）

> AVPacket是存储压缩编码数据相关信息的结构体

Stream -> packet -> frames

C++

```cpp
// 1. alloc
AVFrame* pFrame = av_frame_alloc();
AVPacket* pPacket = av_packet_alloc();

// 2. 循环，读取8帧数据
while (av_read_frame(pFormatContext, pPacket) >= 0){/*...*/};

// 3. 循环内对一帧数据解码(packet->codec--->frame)
// 3.1 Supply raw packet data as input to a decoder
int response = avcodec_send_packet(pCodecContext, pPacket);
while (response >= 0)  // *****
{
    // 3.2 Return decoded output data (into a frame) from a decoder
    response = avcodec_receive_frame(pCodecContext, pFrame);
    // 3.3 Other operations, e.g,
    save_gray_frame(/*...*/);
}
```

视频同步与时间

参考：

http://dranger.com/ffmpeg/tutorial05.html

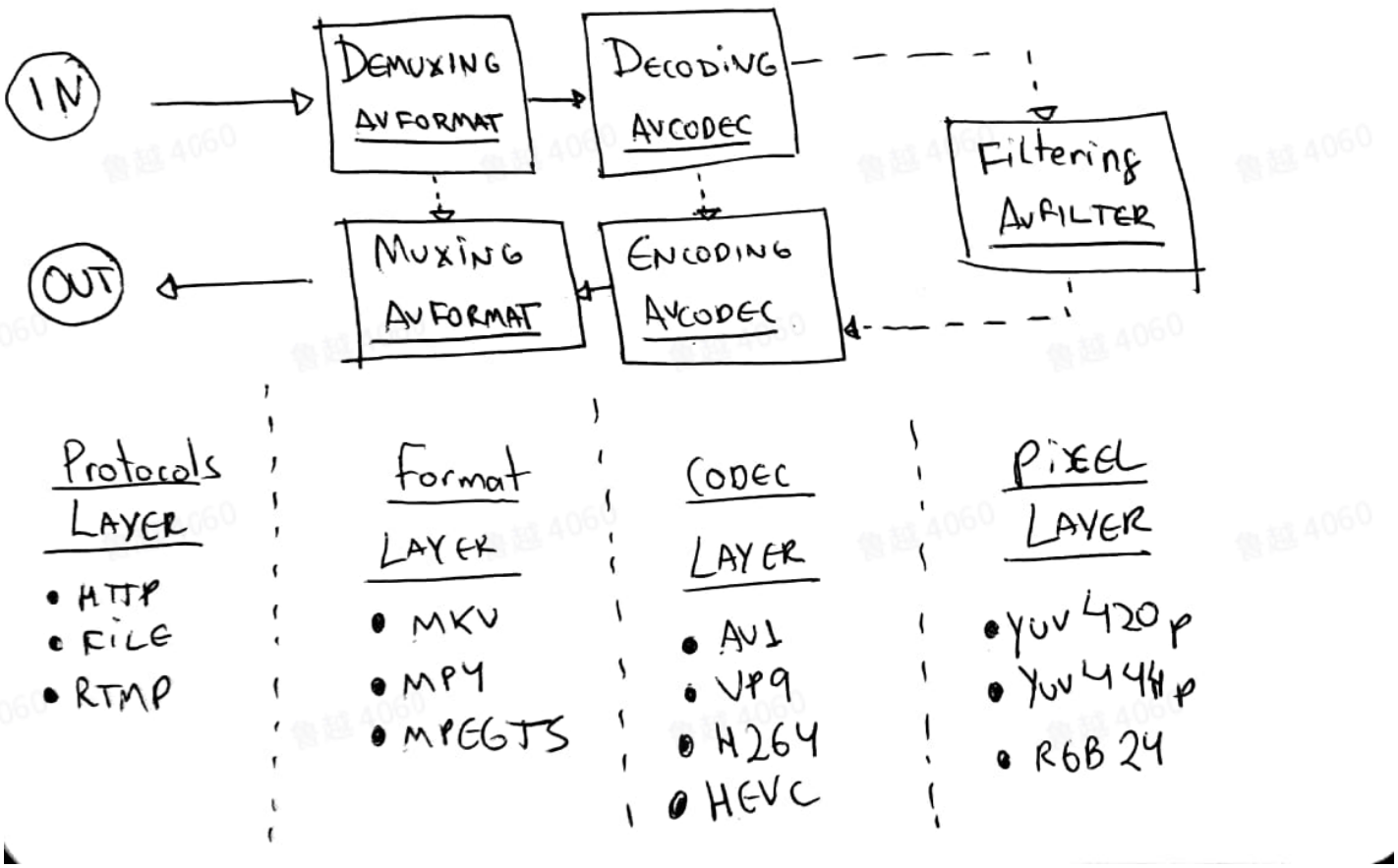
https://en.wikipedia.org/wiki/Presentation_timestamp

- fps：一秒内出几帧；
- timescale：？？？
- PTS: presentation timestamp (increment of PTS = timescale / fps)
- DTS：decode timestamp
- 接收到视频数据时，是以DTS的顺序，同步为PTS

> Decoding of N elementary streams is synchronized by adjusting the decoding of streams to a common master time base rather than by adjusting the decoding of one stream to match that of another. The master time base may be one of the N decoders' clocks, the data source’s clock, or it may be some external clock

Remuxing

- 无需编码-解码
- 整个基于ffmpeg处理音视频的流程



Remuxing代码

1. 构造输出流

C++

```
AVStream* out_stream;
AVFormatContext* output_format_context = NULL;

// prepare output_format_context
avformat_alloc_output_context2(&output_format_context, NULL, NULL,
out_filename);
out_stream = avformat_new_stream(output_format_context, NULL);

avcodec_parameters_copy(out_stream->codecpar, in_codecpar);
// 输出detailed info
av_dump_format(output_format_context, 0, out_filename, 1);
// AVIOContext-create output file
avio_open(&output_format_context->pb, out_filename, AVIO_FLAG_WRITE);
```

2. 实现转码，写header

> 命令行：

Plain Text

```
ffmpeg -i non_fragmented.mp4 -movflags
frag_keyframe+empty_moov+default_base_moof fragmented.mp4
```

C++

```
AVDictionary* opts = NULL;
av_dict_set(&opts, "movflags", "frag_keyframe+empty_moov+default_base_moof", 0);
// realize remuxing
avformat_write_header(output_format_context, &opts);
```

3. 拷贝**packet**

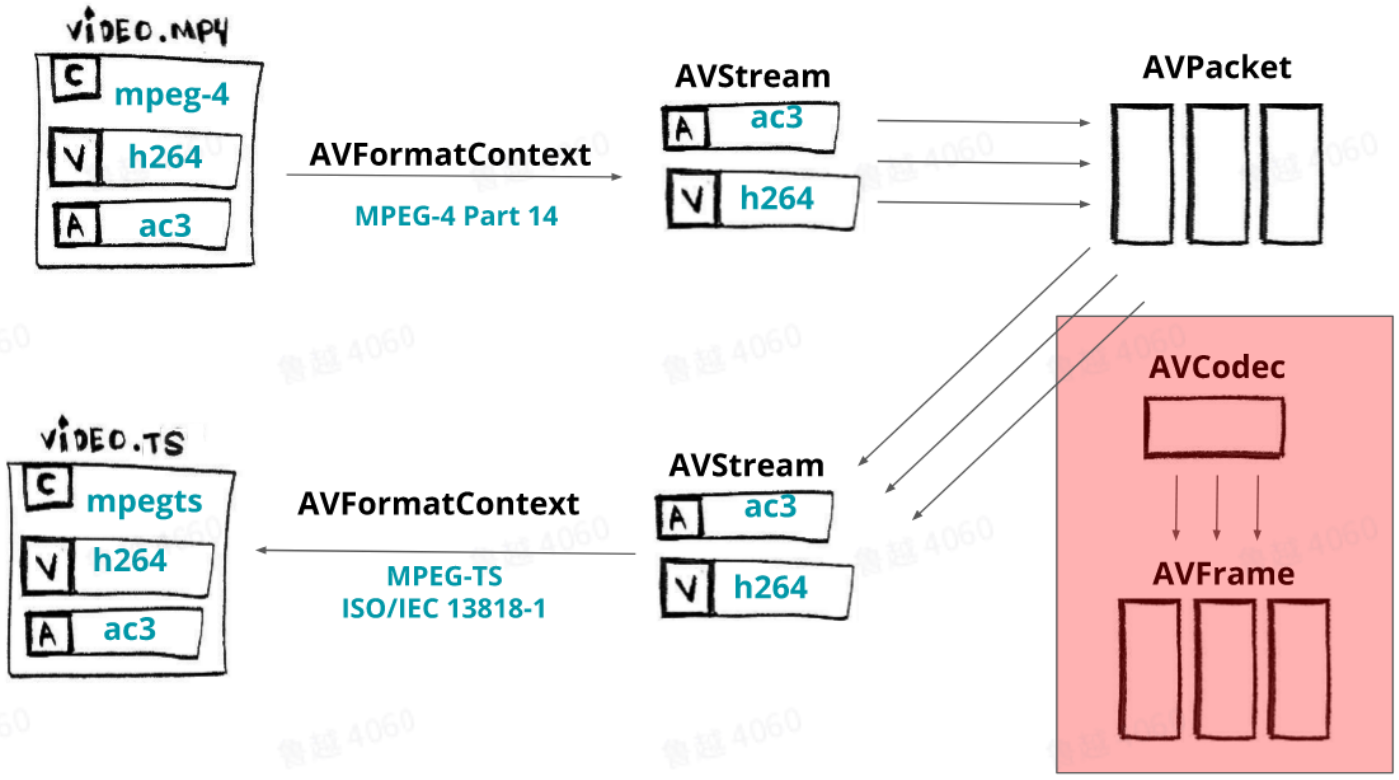
> we can **copy the streams, packet by packet**, from our input to our output streams. We'll loop while it has packets (`av_read_frame`), for each packet we need to **re-calculate the PTS and DTS** to finally write it (`av_interleaved_write_frame`) to our output format context.

C++

```
while (1) {
    AVStream* in_stream, * out_stream;
    ret = av_read_frame(input_format_context, &packet);
    if (ret < 0)
        break;
    in_stream = input_format_context->streams[packet.stream_index];
    if (packet.stream_index >= number_of_streams || streams_list[packet.stream_index] < 0) {
        av_packet_unref(&packet); // wipe the packet
        continue;
    }
    packet.stream_index = streams_list[packet.stream_index];
    out_stream = output_format_context->streams[packet.stream_index];
    /* copy packet */
    packet.pts = av_rescale_q_rnd(packet.pts, in_stream->time_base, out_stream->time_base, static_cast<AVRounding>(AV_ROUND_NEAR_INF | AV_ROUND_PASS_MINMAX));
    packet.dts = av_rescale_q_rnd(packet.dts, in_stream->time_base, out_stream->time_base, static_cast<AVRounding>(AV_ROUND_NEAR_INF | AV_ROUND_PASS_MINMAX));
    packet.duration = av_rescale_q(packet.duration, in_stream->time_base, out_stream->time_base);
    packet.pos = -1;

    // Write a packet to an output media file ensuring correct interleaving.
    ret = av_interleaved_write_frame(output_format_context, &packet);
    if (ret < 0) {
        fprintf(stderr, "Error muxing packet\n");
        break;
    }
    av_packet_unref(&packet);
}
// Write the stream trailer to an output media file
av_write_trailer(output_format_context);
```

流程

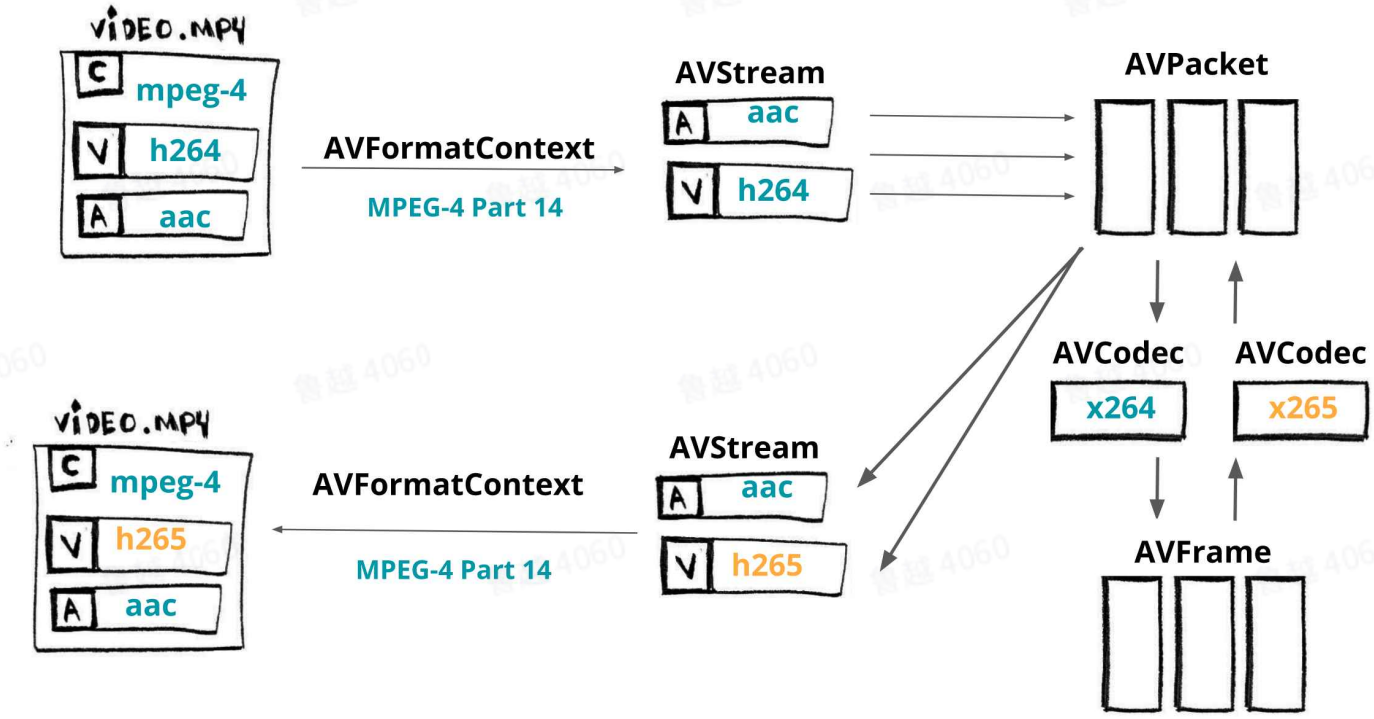


转码-transcoding

- 各个结构体间的关系

> The **AVFormatContext** is the abstraction for the format of the media file, aka container (ex: MKV, MP4, Webm, TS). The **AVStream** represents each type of data for a given format (ex: audio, video, subtitle, metadata). The **AVPacket** is a slice of compressed data obtained from the `AVStream` that can be decoded by an **AVCodec** (ex: av1, h264, vp9, hevc) generating a raw data called **AVFrame**.
>
> The `AVFormatContext` will **give us access to all the** `AVStream` components and for each one of them, we can get their `AVCodec` and create the particular `AVCodecContext` and finally we can open the given codec so we can proceed to the decoding process.
>
> The **AVCodecContext** holds data about **media configuration** such as bit rate, frame rate, sample rate, channels, height, and many others.

- 流程



- Transcoding代码

1. Set up the encoder

   - Create the video `AVStream` in the encoder, `avformat_new_stream`
   - Use the `AVCodec` called `libx265`, `avcodec_find_encoder_by_name`
   - Create the `AVCodecContext` based in the created codec, `avcodec_alloc_context3`
   - Set up basic attributes for the transcoding session, and
   - Open the codec and copy parameters from the context to the stream. `avcodec_open2` and `avcodec_parameters_from_context`

2. expand our decoding loop for the video stream transcodingexpand our decoding loop for the video stream transcoding

   - Send the empty `AVPacket` to the decoder, `avcodec_send_packet`
   - Receive the uncompressed `AVFrame`, `avcodec_receive_frame`
   - Start to transcode this raw frame,
   - Send the raw frame, `avcodec_send_frame`
   - Receive the compressed, based on our codec, `AVPacket`, `avcodec_receive_packet`
   - Set up the timestamp, and `av_packet_rescale_ts`
   - Write it to the output file. `av_interleaved_write_frame`

配置好的源码：https://github.com/YueLu0116/ffmpeg-example

Part II

http://slhck.info/ffmpeg-encoding-course/#/

Reference:

Lei Xiaohua's learning resource about video/audio technics

FFmpeg

主流编解码器

MOST IMPORTANT (LOSSY) CODECS

Currently mostly used, standardized by ITU/ISO:

- 🎥 H.262 / MPEG-2 Part H: Broadcasting, TV, used for backwards compatibility
- 🎥 H.264 / MPEG-4 Part 10: The de-facto standard for video encoding today
- 🎥 H.265 / HEVC / MPEG-H: Successor of H.264, up to 50% better quality
- 🔊 MP3 / MPEG-2 Audio Layer III: Used to be the de-facto audio coding standard
- 🔊 AAC / ISO/IEC 14496-3:2009: Advanced Audio Coding standard

Competitors that are royalty-free:

- 🎥 VP8: Free, open-source codec from Google (not so much in use anymore)
- 🎥 VP9: Successor to VP8, almost as good as H.265
- 🎥 AV1: A successor to VP9, claims to be better than H.265

uncompressed frames filter

裁剪视频

Plain Text

```
ffmpeg -ss <start-time> -i <input> -t <duration> -c copy <output>
ffmpeg -ss <start-time> -i <input> -to <end> -c copy <output>
```

视频质量设置

- Do not just encode without setting any quality level!

```
// set bitrate
-b:v / -b:a <bitrates: 1000k | 8M, etc.>
// set fixed-quality
-q:v / -q:a <2>
```

- two-pass encoding??

```
ffmpeg -y -i <input> -c:v libx264 -b:v 8M -pass 1 -c:a aac -b:a 128k -f mp4 /dev/null
ffmpeg -i <input> -c:v libx264 -b:v 8M -pass 2 -c:a aac -b:a 128k output.mp4
```

- encoding with the present option

```
ffmpeg -i <input> -c:v libx264 -crf 23 -preset ultrafast -an output.mkv
ffmpeg -i <input> -c:v libx264 -crf 23 -preset medium -an output.mkv
ffmpeg -i <input> -c:v libx264 -crf 23 -preset veryslow -an output.mkv
```

改变帧率

Streaming Mapping

STREAM MAPPING

Each file and its streams have a unique ID, starting with 0.

Examples:

- `0:0` is the first stream of the first input file
- `0:1` is the second stream of the first input file
- `2:a:0` is the first audio stream of the third input file
- …

You can map input streams to output, e.g. to add audio to a video:

```
ffmpeg -i input.mp4 -i input.m4a -c copy -map 0:v:0 -map 1:a:0 output.mp4
```

Scaling

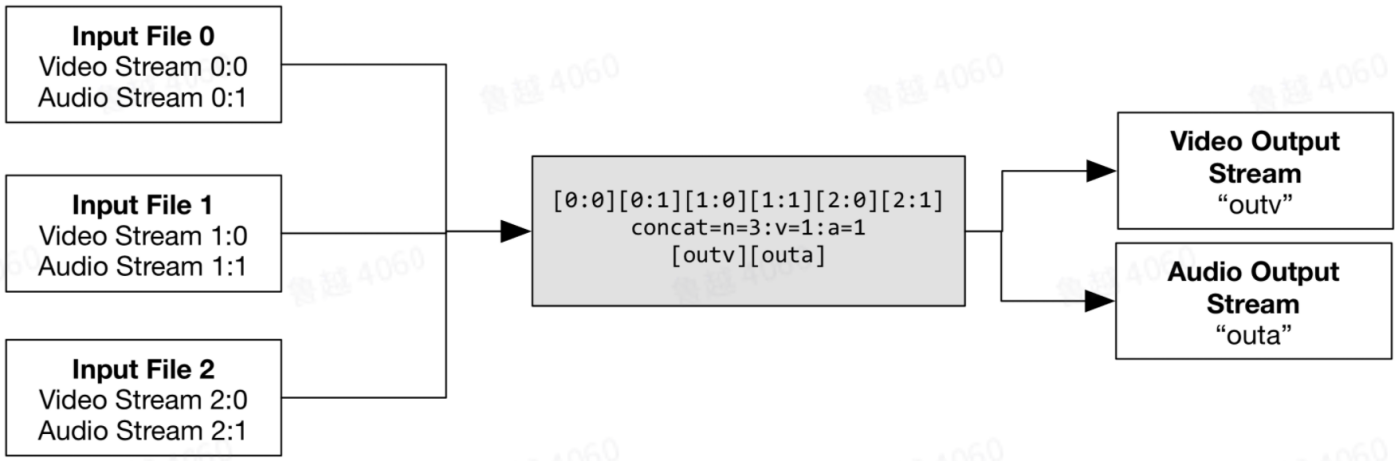
Padding

FADING

视频上绘制文本

流合并

Decode three video/audio streams and append to one another:




```
ffmpeg -i <input1> -i <input2> -i <input3> -filter_complex \
    "[0:0][0:1][1:0][1:1][2:0][2:1]concat=n=3:v=1:a=1[outv][outa]" \
    -map "[outv]" -map "[outa]" <output>
```

时间线编辑

计算信噪比等

fprobe

Plain Text

```
ffprobe <input>
    [-select_streams <selection>]
    [-show_streams|-show_format|-show_frames|-show_packets]
    [-show_entries <entries>]
    [-of <output-format>]
```