原创 芥末的无奈 于 2020-10-11 16:19:41 发布 941 收藏 2 版权
分类专栏： 音频处理 文章标签： ffmpeg 音频编码解码

音频编码

- 基于 FFMPEG 的音频编解码（一）：Hello FFMPEG，安装与编译
- 基于 FFMPEG 的音频编解码（二）：音频解码
- 基于 FFMPEG 的音频编解码（三）：音频编码

在前面文章中，我们了解如何用 FFMPEG 进行音频解码，今天我们来讨论音频编码

一个很重要的概念是，**编码与解码是一对逆过程。** 回想一下，解码的时候我们是如何做的？下面代码是解码的关键部分：

```
av_read_frame(ctx, packet);
avcodec_send_packet(ctx, packet);
avcodec_receive_frame(ctx, frame);
```

1. `av_read_frame` 从文件中读取一个 `packet`
2. `avcodec_send_packet` 将 `packet` 送去解码
3. `avcodec_receive_frame` 取回解码好的数据

编码的过程刚好是解码的逆过程，下面代码是编码的关键部分：

```
fill_data_to_frame(frame);
avcodec_send_frame(ctx, frame);
avcodec_receive_packet(ctx, packet);
```

1. `fill_data_to_frame` 将数据填充至 `frame` 中
2. `avcodec_send_frame` 将 `frame` 送去编码
3. `avcodec_receive_packet` 取回编码好的数据

Show me the code

FFMPEG 官方 examples 中有一份关于编码的示例 encode_audio.c，这份代码中，向我们展示了如何编码音频数据为 MP2 格式。

在我折腾这份代码时，发现它并不能编码例如 aac、flac、wav 等格式，输出的文件播放器不认识，而 mp1/mp2/mp3 却是能够打开并正常播放的。搜索一番后才找到原因，aac 等格式缺少音频文件头，这些必要信息的缺失导致无法正常播放，而 mp3 等格式有一种叫 **frame header** 来记录每一帧数据的信息，因此 encode_audio.c 编码后的文件，只有 mp3 等格式才能正常播放。

非常遗憾 encode_audio.c 并不是我们想要的，但这份代码中，有很多细节值得我们学习，例如判断是否支持当前采样率、判断是否支持当前采样格式、如何编码一个 frame 等。

事情变得稍微复杂了一些，我们不仅要知道如何编码数据，还需要知道如何写文件头。又是一番苦苦搜索，最终锁定了一份雷霄骅大神的代码（愿天堂没有代码 ）最简单的基于FFMPEG的音频编码器（PCM编码为AAC，这份就是我们想要的，可以将音频数据编码为任意格式（只要 ffmpeg 支持）。这份代码年代有些久远了，里面使用到的 API 有些已经过时了，在其基础上，使用目前较新的 API 得到了下面这份代码：

```
//
// Created by William.Hua on 2020/9/30.
//
#if defined(__cplusplus)
extern "C"
{
#endif

#include <libavcodec/avcodec.h>

#include <libavutil/channel_layout.h>
#include <libavutil/common.h>
#include <libavutil/frame.h>
#include <libavutil/samplefmt.h>
#include <libavformat/avformat.h>

#if defined(__cplusplus)
}
#endif

#include <iostream>
#include <string>
using namespace std;

static void encode(AVCodecContext *ctx, AVFrame *frame, AVPacket *pkt,
                   AVFormatContext* pFormatCtx, AVStream* audio_st)
{
    int ret;

    /* send the frame for encoding */
    ret = avcodec_send_frame(ctx, frame);
    if (ret < 0) {
        fprintf(stderr, "Error sending the frame to the encoder\n");
        exit(1);
    }

    /* read all the available output packets (in general there may be any
     * number of them */
    while (ret >= 0) {
        ret = avcodec_receive_packet(ctx, pkt);
        if (ret == AVERROR(EAGAIN) || ret == AVERROR_EOF)
            return;
        else if (ret < 0) {
            fprintf(stderr, "Error encoding audio frame\n");
            exit(1);
        }

        pkt->stream_index = audio_st->index;
        if(frame){
            pkt->pts = frame->pts;
            frame->pts += 100;
        }
        av_write_frame(pFormatCtx, pkt);
        av_packet_unref(pkt);
    }
}

int main(int argc, char* argv[])
{
```

目录
音频编码
Show me the code
总结
参考资料

分类专栏
音视频 11篇
MLT 2篇
opengl 7篇
SDL2 5篇
SIMD 4篇
DSP 6篇
音效处理 8篇
优化 1篇
cmake
qt 1篇
portaudio 1篇
rtaudio
stl 6篇
ffmpeg 4篇
c++ 20篇
嵌入式 2篇
python 5篇
算法 6篇
图像处理 6篇
数学 5篇
深度学习 14篇
神经网络 5篇
keras 15篇
deep-learning 12篇
tensorflow 8篇
QRCode 4篇
zint 2篇
音频处理 28篇
OC 2篇
其他 3篇
git 1篇
安卓 5篇

```cpp
    if(argc < 2){
        cerr << "Usage: encode_audio /full/path/to/output_file\n";
        return -1;
    }

    const string output_file = argv[1];

    // open context
    AVFormatContext* format_ctx = NULL;
    avformat_alloc_output_context2(&format_ctx, NULL, NULL, output_file.c_str());
    if(!format_ctx){
        cerr << "Cannot alloc output context\n";
        return -1;
    }

    // open output file
    if(avio_open(&format_ctx->pb, output_file.c_str(), AVIO_FLAG_WRITE) < 0){
        cerr << "Cannot open output file\n";
        return -1;
    }

    // create new audio stream
    AVStream* audio_st = avformat_new_stream(format_ctx, 0);
    if(!audio_st){
        cerr << "Cannot create audio stream\n";
        return -1;
    }

    // set codec context parameters
    const int sample_rate = 44100;
    const int num_channels = 2;
    const int bit_rate = 64000;
    AVCodecContext* codec_ctx = audio_st->codec;
    codec_ctx->codec_id = format_ctx->oformat->audio_codec;
    codec_ctx->codec_type = AVMEDIA_TYPE_AUDIO;
    codec_ctx->sample_rate = sample_rate;
    codec_ctx->channels = num_channels;
    codec_ctx->sample_fmt = AV_SAMPLE_FMT_FLTP; // planar float
    codec_ctx->channel_layout = av_get_default_channel_layout(num_channels);
    codec_ctx->bit_rate = bit_rate;

    // print detailed information about output format
    av_dump_format(format_ctx, 0, output_file.c_str(), 1);

    // find encode codec
    AVCodec* codec = avcodec_find_encoder(codec_ctx->codec_id);
    if(!codec){
        cerr << "Cannot find encode codec\n";
        return -1;
    }

    // open encode codec
    if(avcodec_open2(codec_ctx, codec, NULL) < 0){
        cerr << "Cannot open codec\n";
        return -1;
    }

    // alloc frame
    AVFrame* frame = av_frame_alloc();
    if(!frame){
        cerr << "Cannot alloc frame\n";
        return -1;
    }
    frame->nb_samples = codec_ctx->frame_size != 0 ? codec_ctx->frame_size : 1024;
    frame->format = codec_ctx->sample_fmt;
    frame->channel_layout = codec_ctx->channel_layout;
    // allocate buffer for frame
    if(av_frame_get_buffer(frame, 0) < 0){
        cerr << "Cannot allocate buffer for frame\n";
        return -1;
    }
    // make sure frame is writeable
    if(av_frame_make_writable(frame) < 0){
        cerr << "Cannot make frame writeable\n";
        return -1;
    }

    // alloc packet
    AVPacket* pkt = av_packet_alloc();
    if(!pkt){
        cerr << "Cannot allocate packet\n";
        return -1;
    }

    // write header
    if(avformat_write_header(format_ctx, NULL) < 0){
        cerr << "Cannot write header\n";
        return -1;
    }

    // write some
    const int num_output_frame = 500;
    float t = 0;
    float tincr = 2 * M_PI * 440.0f / sample_rate; // 440Hz sine wave
    auto* left_channel = reinterpret_cast<float*>(frame->data[0]);
    auto* right_channel = reinterpret_cast<float*>(frame->data[1]);
    frame->pts = 0;
    for(int i = 0; i < num_output_frame; ++i){
        // generate sine wave
        for(int j = 0; j < frame->nb_samples; ++j){
            left_channel[j] = sin(t);
            right_channel[j] = left_channel[j];

            t += tincr;
        }

        // encode sine wave, and send them to output file
        encode(codec_ctx, frame, pkt, format_ctx, audio_st);
    }

    // flush
    encode(codec_ctx, NULL, pkt, format_ctx, audio_st);

    av_packet_free(&pkt);
    av_frame_free(&frame);
    avcodec_close(audio_st->codec);
    avio_close(format_ctx->pb);
    avformat_free_context(format_ctx);
}
```

接下来对上述代码进行一些解释与说明

首先，申请并打开容器。`avformat_alloc_output_context2` 通过文件名后缀来猜测容器类型。

```
AVFormatContext* format_ctx = NULL;
avformat_alloc_output_context2(&format_ctx, NULL, NULL, output_file.c_str());
```

接着打开输出文件。

```
avio_open(&format_ctx->pb, output_file.c_str(), AVIO_FLAG_WRITE)
```

在容器中创建一条音频流，用于输出编码后的数据。

```
AVStream* audio_st = avformat_new_stream(format_ctx, 0);
```

设置编码器的各类参数，包括采样率、通道数、比特率等。需要注意的是，为了简化代码，这里采用 `AV_SAMPLE_FMT_FLTP` 作为采样格式，但是有些格式（例如 .flac ）是不支持 `AV_SAMPLE_FMT_FLTP` 的，在实际项目中，应该判断当前容器类型是否支持采样格式，判断的代码大家可以在 encode_audio.c 找到灵感，这里不再展开说了。

```
const int sample_rate = 44100;
const int num_channels = 2;
const int bit_rate = 64000;
AVCodecContext* codec_ctx = audio_st->codec;
codec_ctx->codec_id = format_ctx->oformat->audio_codec;
codec_ctx->codec_type = AVMEDIA_TYPE_AUDIO;
codec_ctx->sample_rate = sample_rate;
codec_ctx->channels = num_channels;
codec_ctx->sample_fmt = AV_SAMPLE_FMT_FLTP; // planar float
codec_ctx->channel_layout = av_get_default_channel_layout(num_channels);
codec_ctx->bit_rate = bit_rate;
```

找到并打开编码器。

```
AVCodec* codec = avcodec_find_encoder(codec_ctx->codec_id);
avcodec_open2(codec_ctx, codec, NULL);
```

申请 `AVFrame` 用于存放数据。`av_frame_get_buffer` 通过 `nb_samples, sample_fmt, channel_layout` 信息计算需要的内存大小，并进行申请。`av_frame_make_writable` 保证 `AVFrame` 是可写的。

```
AVFrame* frame = av_frame_alloc();
frame->nb_samples = codec_ctx->frame_size != 0 ? codec_ctx->frame_size : 1024;
frame->format = codec_ctx->sample_fmt;
frame->channel_layout = codec_ctx->channel_layout;
av_frame_get_buffer(frame, 0);
av_frame_make_writable(frame);
```

写文件头。

```
avformat_write_header(format_ctx, NULL);
```

生成正弦波，并写入文件。由于采样格式为 `AV_SAMPLE_FMT_FLTP`，且声道数为 2，因此 `frame->data[0]` 为左声道，`frame->data[1]` 为右声道。

```
auto* left_channel = reinterpret_cast<float*>(frame->data[0]);
auto* right_channel = reinterpret_cast<float*>(frame->data[1]);

for(int j = 0; j < frame->nb_samples; ++j){
    left_channel[j] = sin(t);
    right_channel[j] = left_channel[j];

    t += tincr;
}
encode(codec_ctx, frame, pkt, format_ctx, audio_st);
```

在 `encode` 函数中，将 `frame` 送去编码，接着通过 `avcodec_receive_packet` 取出编码后的 `packet`，最后将 `packet` 写入文件。

```
avcodec_send_frame(ctx, frame);
while (ret >= 0) {
    ret = avcodec_receive_packet(ctx, pkt);
    if (ret == AVERROR(EAGAIN) || ret == AVERROR_EOF)
        return;
    else if (ret < 0) {
        fprintf(stderr, "Error encoding audio frame\n");
        exit(1);
    }

    pkt->stream_index = audio_st->index;
    if(frame){
        pkt->pts = frame->pts;
        frame->pts += 100;
    }
```

进行 flush，把剩余数据一网打尽。

```
encode(codec_ctx, NULL, pkt, format_ctx, audio_st);
```

最后别忘记释放内存。

```
av_packet_free(&pkt);
av_frame_free(&frame);
avcodec_close(audio_st->codec);
avio_close(format_ctx->pb);
avformat_free_context(format_ctx);
```

总结

我们介绍了如何利用 FFMPEG 进行音频编码。编码与解码是一对逆过程，理解这一概念后，我们给出了编码的具体代码，并对其进行了说明与解释，学会了如何写文件头，如何申请音频帧内存，以及如何编码数据等。

完整代码已上传至 github，欢迎 star：
https://github.com/jiemojiemo/ffmepg_audio_tutorial

参考资料

- ffmpeg examples - encode_audio
- 最简单的基于FFMPEG的音频编码器（PCM编码为AAC）

RBF神经网络简单介绍与MATLAB实现 106087
频域特征提取的Python实现（频谱、功率谱、倒频谱） 33745
基于Keras的卷积神经网络（CNN）可视化 29987
RNN循环神经网络的直观理解：基于TensorFlow的简单RNN例子 27110
窗函数作用和性质 24374

最新评论

频域特征提取的Python实现（频谱、功…
bnblzq: 请教一下，在cepstrum图里面，可以看到低频信号的频率，那么高频信号怎…
C++ 中 async、packaged_task、promis…
我就是这样的自己: 是不是说一个是连着阻塞5s，一个是有些并行的阻塞5s呢?
内窥镜去反光的论文整理
芥末的无奈: 抱歉啊，这是研究生时的课题。现在毕业好久了，没再搞了。
内窥镜去反光的论文整理
zzzlnb: 老哥能交流下吗，内窥镜去反光现在还有人搞吗，现在的sota是啥
【音效处理】Compressor 压缩器算法简介
芥末的无奈: 按 c 结构的话，其实是 input signal(db) - gain_computer_output_db

您愿意向朋友推荐“博客详情页”吗?

强烈不推荐 不推荐 一般般 推荐 强烈推荐

最新文章

基于 FFmpeg 的跨平台视频播放器简明教程（七）：使用多线程解码视频和音频
基于 FFmpeg 的跨平台视频播放器简明教程（六）：使用 SDL 播放音频和视频
基于 FFmpeg 的跨平台视频播放器简明教程（五）：使用 SDL 播放视频

2023年 18篇 2022年 21篇
2021年 7篇 2020年 16篇
2019年 10篇 2018年 32篇
2017年 22篇 2016年 3篇
2015年 16篇 2014年 3篇

深入浅出：FFmpeg 音频解码与处理全解析

FFMPEG_音频编码WAV_MP3_本地文件 04-13
FFMPEG_音频编码WAV_MP3_本地文件

ffmpeg音视频解码 急支糖浆的博客 816
#include <iostream> extern "C" { #include "libavformat\avformat.h" #include "libavcodec\avcodec.h" #include "libavutil\avutil.h" #include "libswscale\sws…

FFmpeg音频编解码处理 phymat.nico的专栏 1006
新版的ffmpeg对音频编码处理已经有了很大的变化，记录在此，做个备忘。 早期ffmpeg编码音频，输入数据一般都是S16格式，解码输出一般也是S16，…

ffmpeg4.2.2 音频编码，pcm编码成AAC 想挣五斗米的博客 725
新版的ffmpeg 编码AAC只支持的AV_SAMPLE_FMT_FLTP，老版本的是AV_SAMPLE_FMT_S16,如果输入的PCM数据是AV_SAMPLE_FMT_S16的，avc…

ffmpeg报错：avcodec_send_frame() 返回 -22 a1367666195的博客 3466
可以看到，函数一上来就会检查编码器是否打开，以及传入的AVCodecContext是不是一个编码器，两个条件有一个不成立，就返回。这个错误，avcodec…

Android音视频开发 - > FFmpeg音频编码本地PCM文件转AAC aagjkgluhgyy的博客 3061
编码流程 注册所有组件 av_register_all() 创建封装格式上下文 avformat_alloc_context() 返回一个AVFormatContext 初始化输入输出上下文 avio_open(),…

第三章 使用ffmpeg把wav转aac … 581
本文章介绍基本使用。

七、FFmpeg使用---AAC音频编译 irainsa的博客 303
上一篇文章讲到FFmpeg默认的编译静态库中是没有音视频的编码器的，需要我们手动编译进FFmpeg，这篇文章就讲一样如何编译AAC音频编码器，主要…

基于ALSA-FFMPEG实现音频采集与推流功能 02-25
通过alsa将麦克风设备采集回来的音频裸流数据，通过FFmpeg将裸流数据编码，再将封装好的音频流数据推送到nginx-rtmp服务器

Qt基于FFmpeg读取摄像头并进行H264编码 01-20
Qt5基于FFmpeg读取摄像头生成yuv和rgb数据两份数据，利用解码线程类实现边解码生成rgb数据边在窗口类播放，利用生成的yuv数据进行编码生成h26…

FFmpeg音频编码实例 09-20
使用FFmpeg SDK3.2开发的音频编码、解码例子（vs2008）。【注】网上整理。

基于FFmpeg，读取RTSP, 编解码音视频流，视频流添加文字，保存mp4 12-27
1.基于FFmpeg实现rtsp访问 2.编解码音视频 3.视频流添加文字 4.音视频同步mp4 5.兼容Windows和Linux

ffmpeg mp4 mp3 wav flac webm aac ac3 ogg格式转换 as57147的博客 1460
转载自：ffmpeg mp4 mp3 wav flac webm aac ac3 ogg格式转换 - liuyihua1992 - 博客园 fmpeg是Linux中转换音频视频文件的常用工具。 mp4 to mp3: ffm…

FFmpeg 音频编码 希望能帮助大家，希望大家多多支持，你们的支持… 1279
编码音频数据，把pcm原始数据编码为MP3或者AAC。

使用ffmpeg将无损APE或FLAC转换为苹果的无损格式 weixin_30237281的博客 536
ffmpeg -i "待转换的APE或FLAC" -acodec alac "输出的Apple无损.m4a" 转载于:https://www.cnblogs.com/MaxWoods/archive/2012/11/02/2751863.html

FFmpeg基础:获取音视频的各种编码参数 yinshipin007的博客 1455
获取视频编码参数视频编码参数主要包括:帧率、分辨率、编码格式、码率等，对应的概念如下。帧率(Frame Rate)每秒显示帧数(Frames Per Second)。…

ffmpeg 音频解码 05-29
FFmpeg是一个开源的音视频处理库，可以用来解码、编码、转码、播放等等。它支持多种格式的音频文件解码，包括但不限于mp3、aac、wav等。 使用…

“相关推荐”对你有帮助么?

非常没帮助 没帮助 一般 有帮助 非常有帮助

芥末的无奈 关注

0 2 0 专栏目录

Beta Beta 举报