FFmpeg的学习和在Qt上的应用

# 视频介绍

所谓视频，其实就是将一张一张的图片连续的放出来，就像放幻灯片一样，由于人眼的惰性，因此只要图片的数量足够多，就会觉得是连续的动作。 所以，只需要将一张一张的图片保存下来，这样就可以构成一个视频了。

但是，由于目前网络和存储空间的限制，直接存储图片显然不可行。举个例子：在视频聊天中，假定从摄像头采集的图片每张大小200KB，每秒钟发送大概15帧图片，这样每秒钟需要的流量就需要3M（意味着需要24M的宽带），显然这个要求太高了。

于是，就可以考虑将这些图片压缩来减少大小。并且视频基本上都是连续的两帧图像差别不是太大。因此，在记录下第一张完整的图像之后以后的每张图像都是只记录下和上一帧图像不一样的地方，直到出现了差别很大的图像，才重新记录一帧完整的图像（这个完整的图像就叫做关键帧）。这样就可以很大程度上减小空间了。

上面提到的压缩专业术语叫做视频编码，目前主流的编码格式有国际电联的H.261、H.263、H.264、H.265，运动静止图像专家组的M-JPEG和国际标准化组织运动图像专家组的MPEG系列标准，此外在互联网上被广泛应用的还有Real-Networks的RealVideo、微软公司的WMV以及Apple公司的QuickTime等。

编码格式这么多，我们应该选择哪种呢？ 经过查阅资料目前比较好的选择是H.264，因此，只需要了解H.264就行。这里就介绍下H.264有哪些优势：

1. 低码率（Low Bit Rate）：和MPEG2和MPEG4 ASP等压缩技术相比，在同等图像质量下，采用H.264技术压缩后的数据量只有MPEG2的1/8，MPEG4的1/3。
2. 高质量的图像：H.264能提供连续、流畅的高质量图像（DVD质量）。
3. 容错能力强：H.264提供了解决在不稳定网络环境下容易发生的丢包等错误的必要工具。
4. 网络适应性强：H.264提供了网络抽象层（Network Abstraction Layer），使得H.264的文件能容易地在不同网络上传输（例如互联网，CDMA，GPRS，WCDMA，CDMA2000等）。
5. 高压缩率，H.264的压缩比达到惊人的102∶1。

H.264最大的优势是具有很高的数据压缩比率，在同等图像质量的条件下，H.264的压缩比是MPEG-2的2倍以上，是MPEG-4的1.5～2倍。举个例子，原始文件的大小如果为88GB，采用MPEG-2压缩标准压缩后变成3.5GB，压缩比为25∶1，而采用H.264压缩标准压缩后变为879MB，从88GB到879MB，H.264的压缩比达到惊人的102∶1。低码率（Low Bit Rate）对H.264的高的压缩比起到了重要的作用，和MPEG-2和MPEG-4 ASP等压缩技术相比，H.264压缩技术将大大节省用户的下载时间和数据流量收费。尤其值得一提的是，H.264在具有高压缩比的同时还拥有高质量流畅的图像，正因为如此，经过H.264压缩的视频数据，在网络传输过程中所需要的带宽更少，也更加经济。

# 音频介绍

前面我们说过视频有一个每秒钟采集多少张的概念，这就叫做视频的帧率。和视频的帧率一样的道理，声音也有一个频率，叫做**采样率**。

人对频率的识别范围是 20HZ - 20000HZ, 如果每秒钟能对声音做 20000 个采样, 回放时就足可以满足人耳的需求. 所以 **22050 的采样频率是常用的, 44100已是CD音质, 超过48000的采样对人耳已经没有意义。**这和电影的每秒 24 帧图片的道理差不多。

关于音频经常见到这样的描述:44100HZ 16bit stereo 或者 22050HZ 8bit mono 等等。

**44100HZ 16bit stereo: 表示每秒钟有 44100 次采样, 采样数据用 16 位(2字节)记录, 双声道(立体声);**

**22050HZ 8bit mono: 表示每秒钟有 22050 次采样, 采样数据用 8 位(1字节)记录, 单声道;**

当然也可以有 16bit 的单声道或 8bit 的立体声, 等等。

上面的44100HZ代表的就是采样率：

**所谓采样率是指：声音信号在“模→数”转换过程中单位时间内采样的次数。采样值是指每一次采样周期内声音模拟信号的积分值。**

对于单声道声音文件，采样数据为八位的短整数（short int 00H-FFH）；

而对于双声道立体声声音文件，每次采样数据为一个16位的整数（int），高八位(左声道)和低八位(右声道)分别代表两个声道。

每个采样数据记录的是振幅, 采样精度取决于储存空间的大小:

1 字节(也就是8bit) 只能记录 256 个数, 也就是只能将振幅划分成 256 个等级;

2 字节(也就是16bit) 可以细到 65536 个数, 这已是 CD 标准了;

4 字节(也就是32bit) 能把振幅细分到 4294967296 个等级, 实在是没必要了。

如果是双声道(stereo), 采样就是双份的, 文件也差不多要大一倍.

这样我们就可以根据一个 wav 文件的大小、采样频率和采样大小估算出一个 wav 文件的播放长度。

譬如 "Windows XP 启动.wav" 的文件长度是 424,644 字节, 它是 "22050HZ / 16bit / 立体声" 格式(这可以从其 "属性->摘要" 里看到),那么它的每秒的传输速率(位速, 也叫比特率、取样率)是 22050\*16\*2 = 705600(bit/s), 换算成字节单位就是 705600/8 = 88200(字节/秒),

播放时间：424644(总字节数) / 88200(每秒字节数) ≈ 4.8145578(秒)。

但是这还不够精确, 包装标准的 PCM 格式的 WAVE 文件(\*.wav)中至少带有 42 个字节的头信息, 在计算播放时间时应该将其去掉, 所以就有：(424644-42) / (22050\*16\*2/8) ≈ 4.8140816(秒). 这样就比较精确了。

采样率（Sample Rate）：22050 Hz（每秒采样次数）

位深度（Bit Depth）：16 bit（每个采样的位数）

声道数（Channels）：2（立体声，即双声道）

比特率（Bit Rate，也叫位速或取样率）计算：

比特率 = 采样率 × 位深度 × 声道数

= 22050 Hz × 16 bit × 2

= 705,600 bit/s（比特每秒）

字节率（Byte Rate）计算：

字节率 = 比特率 ÷ 8

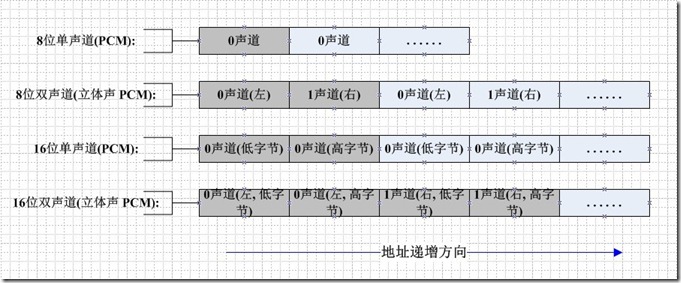
关于声音文件还有一个概念: "位速", 也有叫做比特率、取样率, 譬如上面文件的位速是 705.6kbps 或 705600bps, 其中的 b 是 bit, ps 是每秒的意思;

压缩的音频文件常常用位速来表示, 譬如达到 CD 音质的 MP3 是: 128kbps / 44100HZ.

2. PCM数据格式

PCM(Pulse Code Modulation)也被称为脉码编码调制。PCM中的声音数据没有被压缩，如果是单声道的文件，采样数据按时间的先后顺序依次存入。(它的基本组织单位是BYTE(8bit)或WORD(16bit))。

一般情况下，一帧PCM是由2048次采样组成的，如果是双声道的文件，采样数据按时间先后顺序交叉地存入。如图所示:



1. 单声道（Mono）

定义：

所有声音信号通过 一个声道 录制和播放，无论听者位置如何，听到的声音都是相同的。

技术原理：

音频信号被混合成 单一路径，左右扬声器输出完全一致的声音。

特点：

✅ 设备兼容性强：适合老式收音机、电话、对讲机等单喇叭设备。

✅ 文件体积小：数据量只有双声道的一半（相同码率下）。

❌ 缺乏空间感：无法还原声音的方向、距离或立体空间信息。

常见场景：

电话通话、广播播报、播客录音、监控音频等对方向感要求不高的场景。

2. 双声道（Stereo）

定义：

声音通过两个独立声道（左声道和右声道）录制和播放，模拟人耳听到声音的方位差异。

技术原理：

录制时使用两个麦克风（或立体声麦克风），分别捕捉左右方向的声源；播放时左右扬声器输出不同信号。

特点：

✅ 立体空间感：能还原声音的方位、深度和空间感（如乐器在左、人声在右）。

✅ 沉浸式体验：适合音乐、电影、游戏等需要临场感的场景。

❌ 文件体积更大：数据量是单声道的两倍（相同码率下）。

❌ 设备要求：需两个扬声器或耳机支持。

常见场景：

音乐播放、影视作品、游戏音效、VR音频等。

# Windows安装FFmpeg并在Qt上测试使用

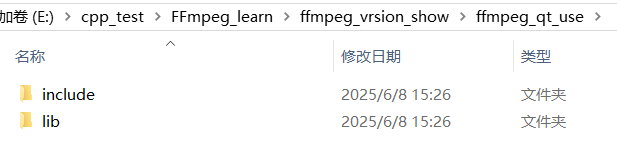
安装过程略

音视频开发中最常做的就是编解码的操作了，以H.264为例：如果想要自己实现编码h.264，需要对H.264非常的了解，首先需要查看H.264的文档，这个文档好像说是三百多页（本人并没有看过）。 想到这，你还写的下去么。 算了吧！

FFMPEG是一个集成了各种编解码器的库，可以说是一个全能型的工具，从视频采集、视频编码到视频传输（包括RTP、RTCP、RTMP、RTSP等等协议）都可以直接使用FFMPEG来完成，更重要的一点FFMPEG是跨平台的，Windows、Linux、Aandroid、IOS这些主流系统通吃。因此初期强烈建议直接使用FFMPEG。

## 在Qt中配置FFmpeg库：

下载好的ffmpeg解压，把其中的include和lib这2个文件夹放入自己新建的文件夹内，直接复制到工程目录下：



Qt Creator中添加工程配置文件 .pro：

INCLUDEPATH += $$PWD/ffmpeg\_qt\_use/include

LIBS += $$PWD/ffmpeg\_qt\_use/lib/avcodec.lib

$$PWD/ffmpeg\_qt\_use/lib/avdevice.lib

$$PWD/ffmpeg\_qt\_use/lib/avfilter.lib

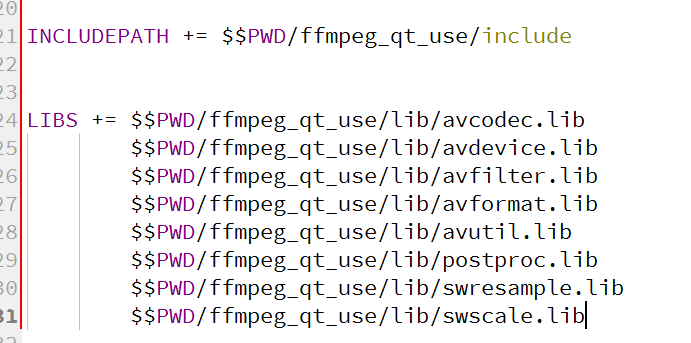
$$PWD/ffmpeg\_qt\_use/lib/avformat.lib

$$PWD/ffmpeg\_qt\_use/lib/avutil.lib

$$PWD/ffmpeg\_qt\_use/lib/postproc.lib

$$PWD/ffmpeg\_qt\_use/lib/swresample.lib

$$PWD/ffmpeg\_qt\_use/lib/swscale.lib



在头文件加入如下代码：

extern "C"

{

#include "libavcodec/avcodec.h"

#include "libavformat/avformat.h"

#include "libswscale/swscale.h"

#include "libavdevice/avdevice.h"

}

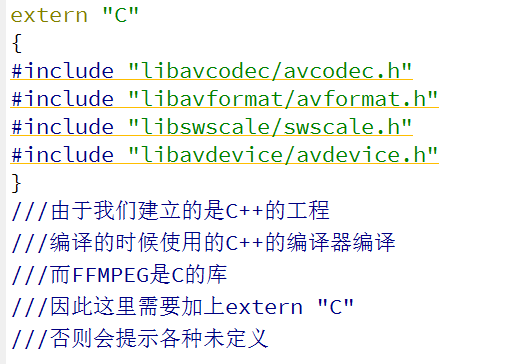
///由于我们建立的是C++的工程

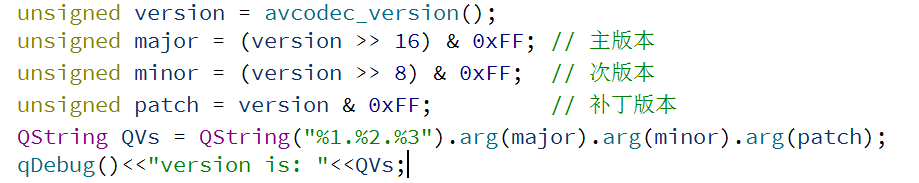
///编译的时候使用的C++的编译器编译

///而FFMPEG是C的库

///因此这里需要加上extern "C"

///否则会提示各种未定义



在main函数中输入：avcodec\_version() 返回的版本号是十六进制编码的（格式：0xMMmmpp，M=主版本，m=次版本，p=补丁），所以我们转为10进制：即可看到ffmpeg的版本号：

还有教程说，要把bin文件夹下的.dll文件放在Qt的exe的文件夹中：



# 1.使用FFMPEG解码视频之保存成图片

使用FFMPEG打开视频文件，并解码保存成一张张的图片。

具体的步骤如下所示：

1.首先需要先初始化一下，使用如下函数：

av\_register\_all(); //初始化FFMPEG

新版本已经不需要

调用了这个才能正常适用编码器和解码器。使用这个函数完成编码器和解码器的初始化，只有初始化了编码器和解码器才能正常使用，否则会在打开编解码器的时候失败。

2.接着需要分配一个AVFormatContext，FFMPEG所有的操作都要通过这个AVFormatContext来进行

AVFormatContext \*pFormatCtx = avformat\_alloc\_context();

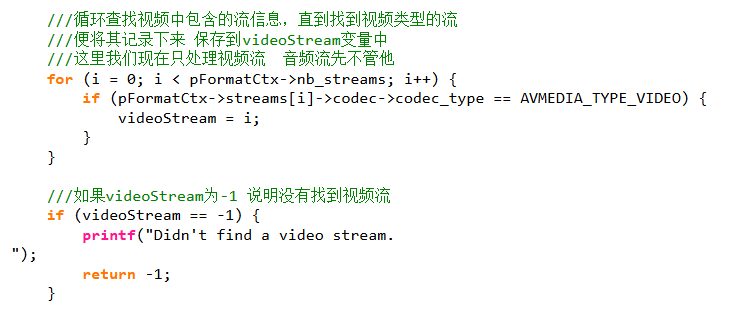
3.接着调用打开视频文件

**这里文件名先不要使用中文，否则会打开失败**

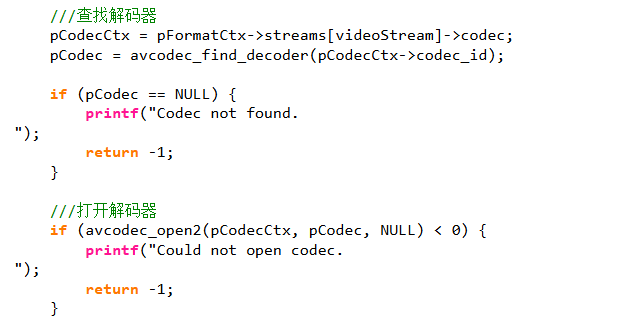
char \*file\_path = "E:in.mp4";

avformat\_open\_input(&pFormatCtx, file\_path, NULL, NULL);

4.文件打开成功后就是查找文件中的视频流了：

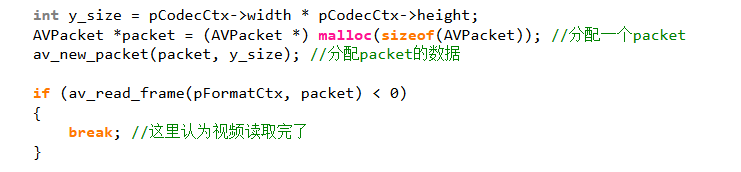


5.现在根据视频流  打开一个解码器来解码：



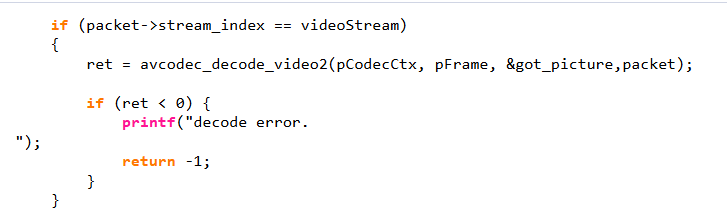
可以看出：我们可以直接根据查找到的视频流信息获取到解码器。而且我们并不知道他实际用的是什么编码器。这就是为什么一开始我们使用FFMPEG来操作，因为很多东西我们可以不关心。

6.现在开始读取视频了：

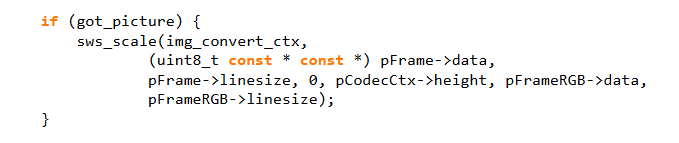


可以看出：av\_read\_frame读取的是一帧视频，并存入一个AVPacket的结构中。

7.前面我们说过 视频里面的数据是经过编码压缩的，因此这里我们需要将其解码：



8.基本上所有解码器解码之后得到的图像数据都是YUV420的格式，而这里我们需要将其保存成图片文件，因此需要将得到的YUV420数据转换成RGB格式，转换格式也是直接使用FFMPEG来完成：



**至于YUV420和RGB图像格式的具体内容，这里不用去了解。这里只需要知道有这么个东西就行了，对我们使用FFMPEG转换没有影响。**

9.得到RGB数据之后就是直接写入文件了：



至此读取视频解码保存成图片就写好了。

# 2.把解码操作放在Qt的子线程去做，主线程UI显示解码后的图片