最简单的基于FFmpeg的解码器-纯净版（不包含libavformat）

2015年01月03日 14:31:33　阅读数：35913

==========

最简单的基于FFmpeg的视频播放器系列文章列表：

100行代码实现最简单的基于FFMPEG+SDL的视频播放器（SDL1.x）

最简单的基于FFMPEG+SDL的视频播放器 ver2 （采用SDL2.0）

最简单的基于FFmpeg的解码器-纯净版（不包含libavformat）

最简单的基于FFMPEG+SDL的视频播放器：拆分-解码器和播放器

最简单的基于FFMPEG的Helloworld程序

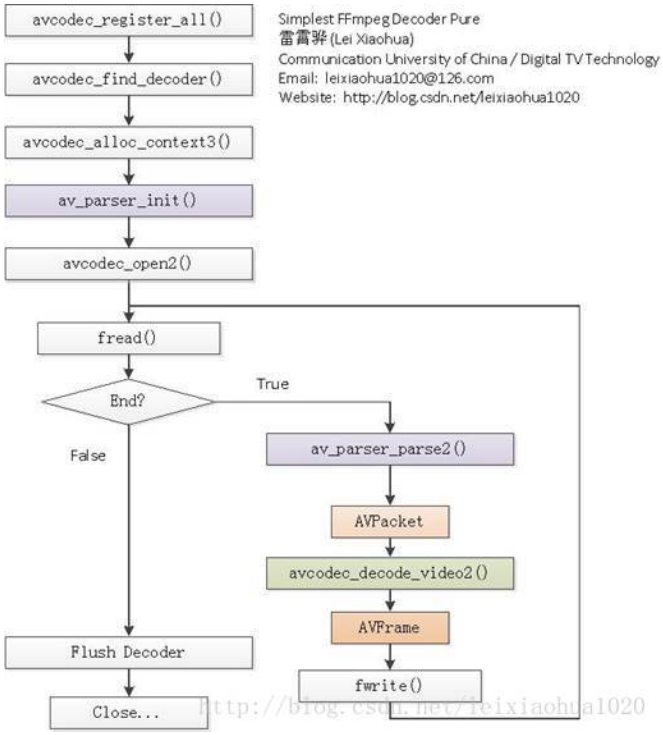
==========

本文记录一个更加“纯净”的基于FFmpeg的视频解码器。此前记录过基于FFmpeg的视频播放器实际上就是一个解码器：

《最简单的基于FFMPEG+SDL的视频播放器 ver2 （采用SDL2.0）》
这个播放器调用了FFmpeg中的libavformat和libavcodec两个库完成了视频解码工作。但是这不是一个“纯净”的解码器。该解码器中libavformat完成封装格式的解析，而libavcodec完成解码工作。一个“纯净”的解码器，理论上说只需要使用libavcodec就足够了，并不需要使用libavformat。本文记录的解码器就是这样的一个“纯净”的解码器，它仅仅通过调用libavcodec将H.264/HEVC等格式的压缩视频码流解码成为YUV数据。

流程图

本文记录的纯净版本的基于FFmpeg的解码器的函数调用流程图如下图所示。需要注意的是，此解码器的输入必须是只包含视频编码数据“裸流”（例如H.264、HEVC码流文件），而不能是包含封装格式的媒体数据（例如AVI、MKV、MP4）。



流程图中关键函数的作用如下所列：

avcodec_register_all()：注册所有的编解码器。
avcodec_find_decoder()：查找解码器。
avcodec_alloc_context3()：为AVCodecContext分配内存。
avcodec_open2()：打开解码器。
avcodec_decode_video2()：解码一帧数据。

有两个平时“不太常见”的函数：

av_parser_init()：初始化AVCodecParserContext。
av_parser_parse2()：解析获得一个Packet。

两个存储数据的结构体如下所列：

AVFrame：存储一帧解码后的像素数据
AVPacket：存储一帧（一般情况下）压缩编码数据

AVCodecParser

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。他的核心函数是av_parser_parse2()。它的定义如下所示。

```cpp
/**
 * Parse a packet.
 *
 * @param s             parser context.
 * @param avctx         codec context.
 * @param poutbuf       set to pointer to parsed buffer or NULL if not yet finished.
 * @param poutbuf_size  set to size of parsed buffer or zero if not yet finished.
 * @param buf           input buffer.
 * @param buf_size      input length, to signal EOF, this should be 0 (so that the last frame can be output).
 * @param pts           input presentation timestamp.
 * @param dts           input decoding timestamp.
 * @param pos           input byte position in stream.
 * @return the number of bytes of the input bitstream used.
 *
 * Example:
 * @code
 *   while(in_len){
 *       len = av_parser_parse2(myparser, AVCodecContext, &data, &size,
 *                                        in_data, in_len,
 *                                        pts, dts, pos);
 *       in_data += len;
 *       in_len  -= len;
 *
 *       if(size)
 *          decode_frame(data, size);
 *   }
 * @endcode
 */
int av_parser_parse2(AVCodecParserContext *s,
                     AVCodecContext *avctx,
                     uint8_t **poutbuf, int *poutbuf_size,
                     const uint8_t *buf, int buf_size,
                     int64_t pts, int64_t dts,
                     int64_t pos);
```

其中poutbuf指向解析后输出的压缩编码数据帧，buf指向输入的压缩编码数据。如果函数执行完后输出数据为空（poutbuf_size为0），则代表解析还没有完成，还需要再次调用av_parser_parse2()解析一部分数据才可以得到解析后的数据帧。当函数执行完后输出数据不为空的时候，代表解析完成，可以将poutbuf中的这帧数据取出来做后续处理。

对比

简单记录一下这个只使用libavcodec的“纯净版”视频解码器和使用libavcodec+libavformat的视频解码器的不同。

PS：使用libavcodec+libavformat的解码器参考文章《 最简单的基于FFMPEG+SDL的视频播放器 ver2 （采用SDL2.0） 》

（1）

下列与libavformat相关的函数在“纯净版”视频解码器中都不存在。

av_register_all()：注册所有的编解码器，复用/解复用器等等组件。其中调用了avcodec_register_all()注册所有编解码器相关的组件。
avformat_alloc_context()：创建AVFormatContext结构体。
avformat_open_input()：打开一个输入流（文件或者网络地址）。其中会调用avformat_new_stream()创建AVStream结构体。avformat_new_stream()中会调用avcodec_alloc_context3()创建AVCodecContext结构体。
avformat_find_stream_info()：获取媒体的信息。
av_read_frame()：获取媒体的一帧压缩编码数据。其中调用了av_parser_parse2()。

（2）

新增了如下几个函数。

avcodec_register_all()：只注册编解码器有关的组件。比如说编码器、解码器、比特流滤镜等，但是不注册复用/解复用器这些和编解码器无关的组件。
avcodec_alloc_context3()：创建AVCodecContext结构体。
av_parser_init()：初始化AVCodecParserContext结构体。
av_parser_parse2()：使用AVCodecParser从输入的数据流中分离出一帧一帧的压缩编码数据。

（3）

程序的流程发生了变化。

在“libavcodec+libavformat”的视频解码器中，使用avformat_open_input()和avformat_find_stream_info()就可以解析出输入视频的信息（例如视频的宽、高）并且赋值给相关的结构体。因此我们在初始化的时候就可以通过读取相应的字段获取到这些信息。
在“纯净”的解码器则不能这样，由于没有上述的函数，所以不能在初始化的时候获得视频的参数。“纯净”的解码器中，可以通过avcodec_decode_video2()获得这些信息。因此我们只有在成功解码第一帧之后，才能通过读取相应的字段获取到这些信息。

源代码

```cpp
/**
 * 最简单的基于FFmpeg的视频解码器（纯净版）
 * Simplest FFmpeg Decoder Pure
 *
 * 雷霄骅 Lei Xiaohua
 * leixiaohua1020@126.com
 * 中国传媒大学/数字电视技术
 * Communication University of China / Digital TV Technology
 * http://blog.csdn.net/leixiaohua1020
 *
 *
 * 本程序实现了视频码流(支持HEVC，H.264，MPEG2等)解码为YUV数据。
 * 它仅仅使用了libavcodec（而没有使用libavformat）。
 * 是最简单的FFmpeg视频解码方面的教程。
 * 通过学习本例子可以了解FFmpeg的解码流程。
 * This software is a simplest decoder based on FFmpeg.
 * It decode bitstreams to YUV pixel data.
 * It just use libavcodec (do not contains libavformat).
 * Suitable for beginner of FFmpeg.
 */

#include <stdio.h>

#define __STDC_CONSTANT_MACROS

#ifdef _WIN32
//Windows
extern "C"
{
#include "libavcodec/avcodec.h"
};
#else
//Linux...
#ifdef __cplusplus
extern "C"
{
#endif
#include <libavcodec/avcodec.h>
#ifdef __cplusplus
};
#endif
#endif


//test different codec
#define TEST_H264  1
#define TEST_HEVC  0

int main(int argc, char* argv[])
{
    AVCodec *pCodec;
    AVCodecContext *pCodecCtx= NULL;
    AVCodecParserContext *pCodecParserCtx=NULL;

    FILE *fp_in;
    FILE *fp_out;
    AVFrame *pFrame;

    const int in_buffer_size=4096;
    uint8_t in_buffer[in_buffer_size + FF_INPUT_BUFFER_PADDING_SIZE]={0};
    uint8_t *cur_ptr;
    int cur_size;
    AVPacket packet;
    int ret, got_picture;
    int y_size;


#if TEST_HEVC
    enum AVCodecID codec_id=AV_CODEC_ID_HEVC;
    char filepath_in[]="bigbuckbunny_480x272.hevc";
#elif TEST_H264
    AVCodecID codec_id=AV_CODEC_ID_H264;
    char filepath_in[]="bigbuckbunny_480x272.h264";
#else
    AVCodecID codec_id=AV_CODEC_ID_MPEG2VIDEO;
    char filepath_in[]="bigbuckbunny_480x272.m2v";
#endif

    char filepath_out[]="bigbuckbunny_480x272.yuv";
    int first_time=1;


    //av_log_set_level(AV_LOG_DEBUG);

    avcodec_register_all();
```

```
    pCodec = avcodec_find_decoder(codec_id);
    if (!pCodec) {
        printf("Codec not found\n");
        return -1;
    }
    pCodecCtx = avcodec_alloc_context3(pCodec);
    if (!pCodecCtx){
        printf("Could not allocate video codec context\n");
        return -1;
    }

    pCodecParserCtx=av_parser_init(codec_id);
    if (!pCodecParserCtx){
        printf("Could not allocate video parser context\n");
        return -1;
    }

    //if(pCodec->capabilities&CODEC_CAP_TRUNCATED)
    //    pCodecCtx->flags|= CODEC_FLAG_TRUNCATED;

    if (avcodec_open2(pCodecCtx, pCodec, NULL) < 0) {
        printf("Could not open codec\n");
        return -1;
    }
    //Input File
    fp_in = fopen(filepath_in, "rb");
    if (!fp_in) {
        printf("Could not open input stream\n");
        return -1;
    }
    //Output File
    fp_out = fopen(filepath_out, "wb");
    if (!fp_out) {
        printf("Could not open output YUV file\n");
        return -1;
    }

    pFrame = av_frame_alloc();
    av_init_packet(&packet);

    while (1) {

        cur_size = fread(in_buffer, 1, in_buffer_size, fp_in);
        if (cur_size == 0)
            break;
        cur_ptr=in_buffer;

        while (cur_size>0){

            int len = av_parser_parse2(
                pCodecParserCtx, pCodecCtx,
                &packet.data, &packet.size,
                cur_ptr , cur_size ,
                AV_NOPTS_VALUE, AV_NOPTS_VALUE, AV_NOPTS_VALUE);

            cur_ptr += len;
            cur_size -= len;

            if(packet.size==0)
                continue;

            //Some Info from AVCodecParserContext
            printf("[Packet]Size:%6d\t",packet.size);
            switch(pCodecParserCtx->pict_type){
                case AV_PICTURE_TYPE_I: printf("Type:I\t");break;
                case AV_PICTURE_TYPE_P: printf("Type:P\t");break;
                case AV_PICTURE_TYPE_B: printf("Type:B\t");break;
                default: printf("Type:Other\t");break;
            }
            printf("Number:%4d\n",pCodecParserCtx->output_picture_number);

            ret = avcodec_decode_video2(pCodecCtx, pFrame, &got_picture, &packet);
            if (ret < 0) {
                printf("Decode Error.\n");
                return ret;
            }
            if (got_picture) {
                if(first_time){
                    printf("\nCodec Full Name:%s\n",pCodecCtx->codec->long_name);
                    printf("width:%d\nheight:%d\n\n",pCodecCtx->width,pCodecCtx->height);
                    first_time=0;
                }
                //Y, U, V
                for(int i=0;i<pFrame->height;i++){
                    fwrite(pFrame->data[0]+pFrame->linesize[0]*i,1,pFrame->width,fp_out);
                }
                for(int i=0;i<pFrame->height/2;i++){
                    fwrite(pFrame->data[1]+pFrame->linesize[1]*i,1,pFrame->width/2,fp_out);
                }
                for(int i=0;i<pFrame->height/2;i++){
```

```
                    fwrite(pFrame->data[2]+pFrame->linesize[2]*i,1,pFrame->width/2,fp_out);
                }

                printf("Succeed to decode 1 frame!\n");
            }
        }

    }

    //Flush Decoder
    packet.data = NULL;
    packet.size = 0;
    while(1){
        ret = avcodec_decode_video2(pCodecCtx, pFrame, &got_picture, &packet);
        if (ret < 0) {
            printf("Decode Error.\n");
            return ret;
        }
        if (!got_picture){
            break;
        }else {
            //Y, U, V
            for(int i=0;i<pFrame->height;i++){
                fwrite(pFrame->data[0]+pFrame->linesize[0]*i,1,pFrame->width,fp_out);
            }
            for(int i=0;i<pFrame->height/2;i++){
                fwrite(pFrame->data[1]+pFrame->linesize[1]*i,1,pFrame->width/2,fp_out);
            }
            for(int i=0;i<pFrame->height/2;i++){
                fwrite(pFrame->data[2]+pFrame->linesize[2]*i,1,pFrame->width/2,fp_out);
            }

            printf("Flush Decoder: Succeed to decode 1 frame!\n");
        }
    }

    fclose(fp_in);
    fclose(fp_out);


    av_parser_close(pCodecParserCtx);

    av_frame_free(&pFrame);
    avcodec_close(pCodecCtx);
    av_free(pCodecCtx);

    return 0;
}
```

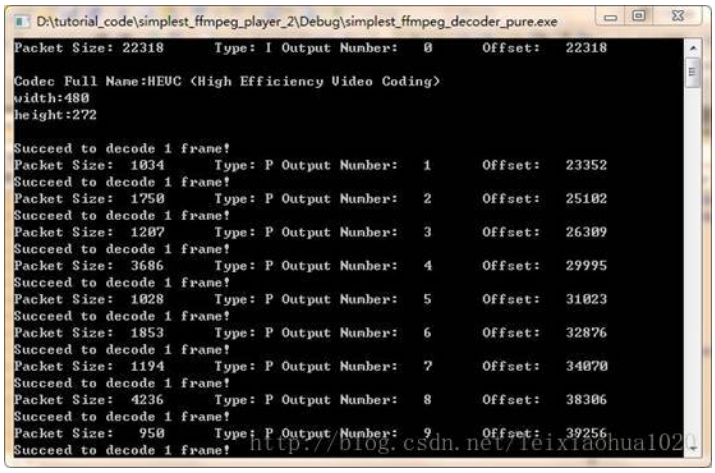
运行结果

通过设定定义在程序开始的宏，确定需要使用的解码器。

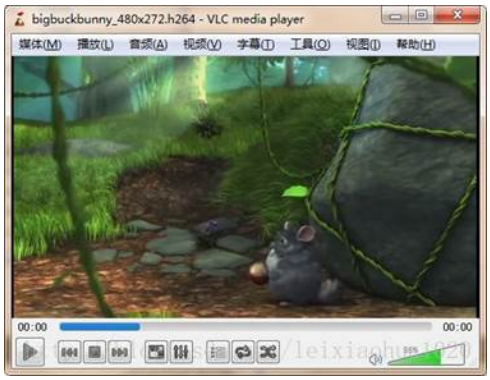
```cpp
//test different codec
#define TEST_H264  0
#define TEST_HEVC  1
```

当TEST_H264设置为1的时候，解码H.264文件“bigbuckbunny_480x272.h264”。
当TEST_HEVC设置为1的时候，解码HEVC文件“bigbuckbunny_480x272.hevc”。
解码后的数据保存成YUV420P格式的文件“bigbuckbunny_480x272.yuv”。

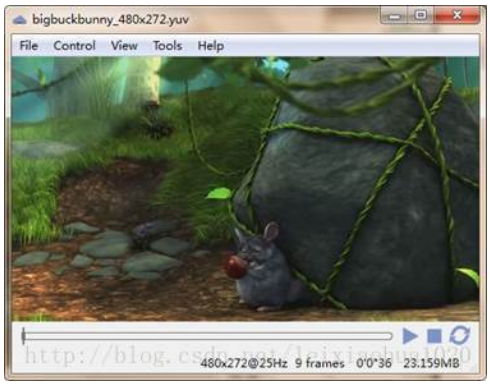
此外，程序在运行的过程中，会打印出AVCodecParserContext中的一些信息，比如说帧类型等等，如下图所示。



输入H.264码流如下所示。



输出YUV420P像素数据如下图所示。



下载

Simplest ffmpeg decoder pure工程被作为子工程添加到了simplest ffmpeg player 2工程中。新版的simplest ffmpeg player 2工程的信息如下。

Simplest ffmpeg player 2

项目主页

SourceForge： https://sourceforge.net/projects/simplestffmpegplayer/

Github： https://github.com/leixiaohua1020/simplest_ffmpeg_player

开源中国： http://git.oschina.net/leixiaohua1020/simplest_ffmpeg_player

本程序实现了视频文件的解码和显示（支持HEVC，H.264，MPEG2等）。
是最简单的FFmpeg视频解码方面的教程。
通过学习本例子可以了解FFmpeg的解码流程。

项目包含3个工程：
simplest_ffmpeg_player：标准版，FFmpeg学习的开始。
simplest_ffmpeg_player_su：SU（SDL Update）版，加入了简单的SDL的Event。
simplest_ffmpeg_decoder_pure：一个纯净的解码器。

version 2.3==========

CSDN下载地址： http://download.csdn.net/detail/leixiaohua1020/8322307

更新-2.4（2015.2.13）==========

这次考虑到了跨平台的要求，调整了源代码。经过这次调整之后，源代码可以在以下平台编译通过：

VC++：打开sln文件即可编译，无需配置。

cl.exe：打开compile_cl.bat即可命令行下使用cl.exe进行编译，注意可能需要按照VC的安装路径调整脚本里面的参数。编译命令如下。

```
[plain]
1. ::VS2010 Environment
2. call "D:\Program Files\Microsoft Visual Studio 10.0\VC\vcvarsall.bat"
3. ::include
4. @set INCLUDE=include;%INCLUDE%
5. ::lib
6. @set LIB=lib;%LIB%
7. ::compile and link
8. cl simplest_ffmpeg_decoder_pure.cpp /link avcodec.lib avutil.lib swscale.lib ^
9. /OPT:NOREF
```

MinGW：MinGW命令行下运行compile_mingw.sh即可使用MinGW的g++进行编译。编译命令如下。

```
[plain]
1. g++ simplest_ffmpeg_decoder_pure.cpp -g -o simplest_ffmpeg_decoder_pure.exe \
2. -I /usr/local/include -L /usr/local/lib -lavcodec -lavutil -lswscale
```

GCC：Linux或者MacOS命令行下运行compile_gcc.sh即可使用GCC进行编译。编译命令如下。

```
[plain]
1. gcc simplest_ffmpeg_decoder_pure.cpp -g -o simplest_ffmpeg_decoder_pure.out -I /usr/local/include -L /usr/local/lib \
2. -lavcodec -lavutil -lswscale
```

PS：相关的编译命令已经保存到了工程文件夹中

CSDN项目下载地址： http://download.csdn.net/detail/leixiaohua1020/8443943

SourceForge、Github等上面已经更新。

更新-2.5（2015.7.17）==========
增加了下列工程：

simplest_ffmpeg_decoder：一个包含了封装格式处理功能的解码器。使用了libavcodec和libavformat。
simplest_video_play_sdl2：使用SDL2播放YUV的例子。
simplest_ffmpeg_helloworld：输出FFmpeg类库的信息。

CSDN项目下载地址： http://download.csdn.net/detail/leixiaohua1020/8924321
SourceForge、Github等上面已经更新。

文章标签： ffmpeg 解码 AVCodecContext 视频
个人分类： FFMPEG 我的开源项目
所属专栏： FFmpeg