最简单的基于FFMPEG的视频编码器（YUV编码为H.264）

2014年05月12日 00:42:25　阅读数：117566

==========

最简单的基于FFmpeg的视频编码器文章列表：

最简单的基于FFMPEG的视频编码器（YUV编码为H.264）

最简单的基于FFmpeg的视频编码器-更新版（YUV编码为HEVC(H.265)）

最简单的基于FFmpeg的编码器-纯净版（不包含libavformat）

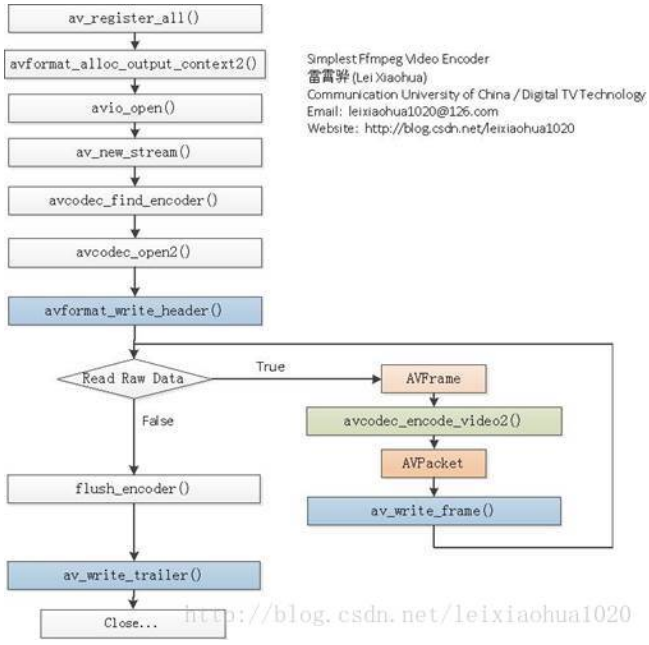
==========

本文介绍一个最简单的基于FFMPEG的视频编码器。该编码器实现了YUV420P的像素数据编码为H.264的压缩编码数据。编码器代码十分简单，但是每一行代码都很重要，适合好好研究一下。弄清楚了本代码也就基本弄清楚了FFMPEG的编码流程。目前我虽然已经调通了程序，但是还是有些地方没有完全搞明白，需要下一步继续探究然后补充内容。

本程序使用最新版的类库（编译时间为2014.5.6），开发平台为VC2010。所有的配置都已经做好，只需要运行就可以了。

流程

下面附一张使用FFmpeg编码视频的流程图。使用该流程，不仅可以编码H.264的视频，而且可以编码MPEG4/MPEG2/VP8等等各种FFmpeg支持的视频。图中蓝色背景的函数是实际输出数据的函数。浅绿色的函数是视频编码的函数。



简单介绍一下流程中各个函数的意义：

av_register_all()：注册FFmpeg所有编解码器。

avformat_alloc_output_context2()：初始化输出码流的AVFormatContext。

avio_open()：打开输出文件。

av_new_stream()：创建输出码流的AVStream。

avcodec_find_encoder()：查找编码器。

avcodec_open2()：打开编码器。

avformat_write_header()：写文件头（对于某些没有文件头的封装格式，不需要此函数。比如说MPEG2TS）。

avcodec_encode_video2()：编码一帧视频。即将AVFrame（存储YUV像素数据）编码为AVPacket（存储H.264等格式的码流数据）。

av_write_frame()：将编码后的视频码流写入文件。

flush_encoder()：输入的像素数据读取完成后调用此函数。用于输出编码器中剩余的AVPacket。

av_write_trailer()：写文件尾（对于某些没有文件头的封装格式，不需要此函数。比如说MPEG2TS）。

代码

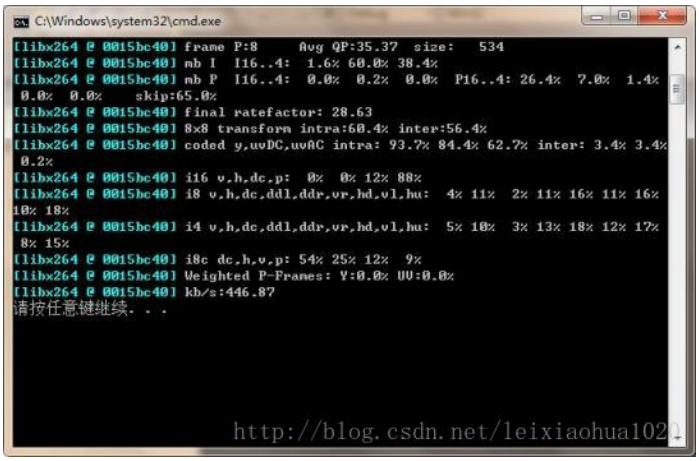
```cpp
/**
 * 最简单的基于FFmpeg的视频编码器
 * Simplest FFmpeg Video Encoder
 *
 * 雷霄骅 Lei Xiaohua
 * leixiaohua1020@126.com
 * 中国传媒大学/数字电视技术
 * Communication University of China / Digital TV Technology
 * http://blog.csdn.net/leixiaohua1020
 *
 * 本程序实现了YUV像素数据编码为视频码流（H264，MPEG2，VP8等等）。
 * 是最简单的FFmpeg视频编码方面的教程。
 * 通过学习本例子可以了解FFmpeg的编码流程。
 * This software encode YUV420P data to H.264 bitstream.
 * It's the simplest video encoding software based on FFmpeg.
 * Suitable for beginner of FFmpeg
 */

#include <stdio.h>

#define __STDC_CONSTANT_MACROS

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


int flush_encoder(AVFormatContext *fmt_ctx,unsigned int stream_index){
    int ret;
    int got_frame;
    AVPacket enc_pkt;
    if (!(fmt_ctx->streams[stream_index]->codec->codec->capabilities &
        CODEC_CAP_DELAY))
        return 0;
    while (1) {
        enc_pkt.data = NULL;
        enc_pkt.size = 0;
        av_init_packet(&enc_pkt);
        ret = avcodec_encode_video2 (fmt_ctx->streams[stream_index]->codec, &enc_pkt,
            NULL, &got_frame);
        av_frame_free(NULL);
        if (ret < 0)
            break;
        if (!got_frame){
            ret=0;
            break;
        }
        printf("Flush Encoder: Succeed to encode 1 frame!\tsize:%5d\n",enc_pkt.size);
        /* mux encoded frame */
        ret = av_write_frame(fmt_ctx, &enc_pkt);
        if (ret < 0)
            break;
    }
    return ret;
}

int main(int argc, char* argv[])
{
    AVFormatContext* pFormatCtx;
    AVOutputFormat* fmt;
    AVStream* video_st;
    AVCodecContext* pCodecCtx;
```

```
    AVCodec* pCodec;
    AVPacket pkt;
    uint8_t* picture_buf;
    AVFrame* pFrame;
    int picture_size;
    int y_size;
    int framecnt=0;
    //FILE *in_file = fopen("src01_480x272.yuv", "rb"); //Input raw YUV data
    FILE *in_file = fopen("../ds_480x272.yuv", "rb");   //Input raw YUV data
    int in_w=480,in_h=272;                              //Input data's width and height
    int framenum=100;                                   //Frames to encode
    //const char* out_file = "src01.h264";              //Output Filepath
    //const char* out_file = "src01.ts";
    //const char* out_file = "src01.hevc";
    const char* out_file = "ds.h264";

    av_register_all();
    //Method1.
    pFormatCtx = avformat_alloc_context();
    //Guess Format
    fmt = av_guess_format(NULL, out_file, NULL);
    pFormatCtx->oformat = fmt;

    //Method 2.
    //avformat_alloc_output_context2(&pFormatCtx, NULL, NULL, out_file);
    //fmt = pFormatCtx->oformat;


    //Open output URL
    if (avio_open(&pFormatCtx->pb,out_file, AVIO_FLAG_READ_WRITE) < 0){
        printf("Failed to open output file! \n");
        return -1;
    }

    video_st = avformat_new_stream(pFormatCtx, 0);
    //video_st->time_base.num = 1;
    //video_st->time_base.den = 25;

    if (video_st==NULL){
        return -1;
    }
    //Param that must set
    pCodecCtx = video_st->codec;
    //pCodecCtx->codec_id =AV_CODEC_ID_HEVC;
    pCodecCtx->codec_id = fmt->video_codec;
    pCodecCtx->codec_type = AVMEDIA_TYPE_VIDEO;
    pCodecCtx->pix_fmt = AV_PIX_FMT_YUV420P;
    pCodecCtx->width = in_w;
    pCodecCtx->height = in_h;
    pCodecCtx->bit_rate = 400000;
    pCodecCtx->gop_size=250;

    pCodecCtx->time_base.num = 1;
    pCodecCtx->time_base.den = 25;

    //H264
    //pCodecCtx->me_range = 16;
    //pCodecCtx->max_qdiff = 4;
    //pCodecCtx->qcompress = 0.6;
    pCodecCtx->qmin = 10;
    pCodecCtx->qmax = 51;

    //Optional Param
    pCodecCtx->max_b_frames=3;

    // Set Option
    AVDictionary *param = 0;
    //H.264
    if(pCodecCtx->codec_id == AV_CODEC_ID_H264) {
        av_dict_set(¶m, "preset", "slow", 0);
        av_dict_set(¶m, "tune", "zerolatency", 0);
        //av_dict_set(¶m, "profile", "main", 0);
    }
    //H.265
    if(pCodecCtx->codec_id == AV_CODEC_ID_H265){
        av_dict_set(¶m, "preset", "ultrafast", 0);
        av_dict_set(¶m, "tune", "zero-latency", 0);
    }

    //Show some Information
    av_dump_format(pFormatCtx, 0, out_file, 1);

    pCodec = avcodec_find_encoder(pCodecCtx->codec_id);
    if (!pCodec){
        printf("Can not find encoder! \n");
        return -1;
    }
    if (avcodec_open2(pCodecCtx, pCodec,¶m) < 0){
        printf("Failed to open encoder! \n");
        return -1;
    }
```

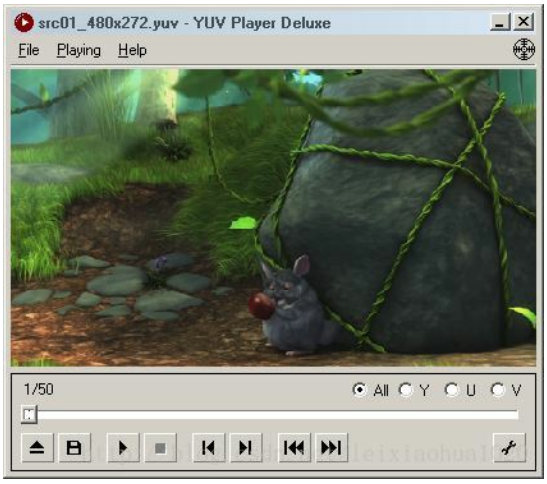
```
    pFrame = av_frame_alloc();
    picture_size = avpicture_get_size(pCodecCtx->pix_fmt, pCodecCtx->width, pCodecCtx->height);
    picture_buf = (uint8_t *)av_malloc(picture_size);
    avpicture_fill((AVPicture *)pFrame, picture_buf, pCodecCtx->pix_fmt, pCodecCtx->width, pCodecCtx->height);

    //Write File Header
    avformat_write_header(pFormatCtx,NULL);

    av_new_packet(&pkt,picture_size);

    y_size = pCodecCtx->width * pCodecCtx->height;

    for (int i=0; i<framenum; i++){
        //Read raw YUV data
        if (fread(picture_buf, 1, y_size*3/2, in_file) <= 0){
            printf("Failed to read raw data! \n");
            return -1;
        }else if(feof(in_file)){
            break;
        }
        pFrame->data[0] = picture_buf;              // Y
        pFrame->data[1] = picture_buf+ y_size;      // U
        pFrame->data[2] = picture_buf+ y_size*5/4;  // V
        //PTS
        //pFrame->pts=i;
        pFrame->pts=i*(video_st->time_base.den)/((video_st->time_base.num)*25);
        int got_picture=0;
        //Encode
        int ret = avcodec_encode_video2(pCodecCtx, &pkt,pFrame, &got_picture);
        if(ret < 0){
            printf("Failed to encode! \n");
            return -1;
        }
        if (got_picture==1){
            printf("Succeed to encode frame: %5d\tsize:%5d\n",framecnt,pkt.size);
            framecnt++;
            pkt.stream_index = video_st->index;
            ret = av_write_frame(pFormatCtx, &pkt);
            av_free_packet(&pkt);
        }
    }
    //Flush Encoder
    int ret = flush_encoder(pFormatCtx,0);
    if (ret < 0) {
        printf("Flushing encoder failed\n");
        return -1;
    }

    //Write file trailer
    av_write_trailer(pFormatCtx);

    //Clean
    if (video_st){
        avcodec_close(video_st->codec);
        av_free(pFrame);
        av_free(picture_buf);
    }
    avio_close(pFormatCtx->pb);
    avformat_free_context(pFormatCtx);

    fclose(in_file);

    return 0;
}
```

结果

软件运行截图（受限于文件体积，原始YUV帧数很少）：



编码前的YUV序列：



编码后的H.264码流：

下载

Simplest FFmpeg Video Encoder

项目主页

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

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

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

下载地址：

http://download.csdn.net/detail/leixiaohua1020/7324115

【修正】之前发现编码后的H.264码流与YUV输入的帧数不同。经过观察对比其他程序后发现需要调用flush_encoder()将编码器中剩余的视频帧输出。已经将该问题修正。

CSDN下载地址（修正后）：

http://download.csdn.net/detail/leixiaohua1020/7466649

PUDN下载地址（修正后）：

http://www.pudn.com/downloads644/sourcecode/multimedia/detail2605258.html

SourceForge上已经更新。

更新-1.1 (2015.1.03)==========

增加了《最简单的基于FFmpeg的编码器-纯净版（不包含libavformat）》中的simplest_ffmpeg_video_encoder_pure工程。

CSDN下载地址：http://download.csdn.net/detail/leixiaohua1020/8322003

更新-1.2 (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_video_encoder.cpp /link avcodec.lib avformat.lib avutil.lib ^
9. avdevice.lib avfilter.lib postproc.lib swresample.lib swscale.lib /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_video_encoder.cpp -g -o simplest_ffmpeg_video_encoder.exe \
2. -I /usr/local/include -L /usr/local/lib \
3. -lavformat -lavcodec -lavutil
```

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_video_encoder.cpp -g -o simplest_ffmpeg_video_encoder.out \
2. -I /usr/local/include -L /usr/local/lib -lavformat -lavcodec -lavutil
```

PS：相关的编译命令已经保存到了工程文件夹中

CSDN下载地址：http://download.csdn.net/detail/leixiaohua1020/8444967

SourceForge上已经更新。

文章标签： ffmpeg 视频 编码 yuv h264

个人分类： FFMPEG 我的开源项目

所属专栏： FFmpeg