

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\(H.265\)）](#)

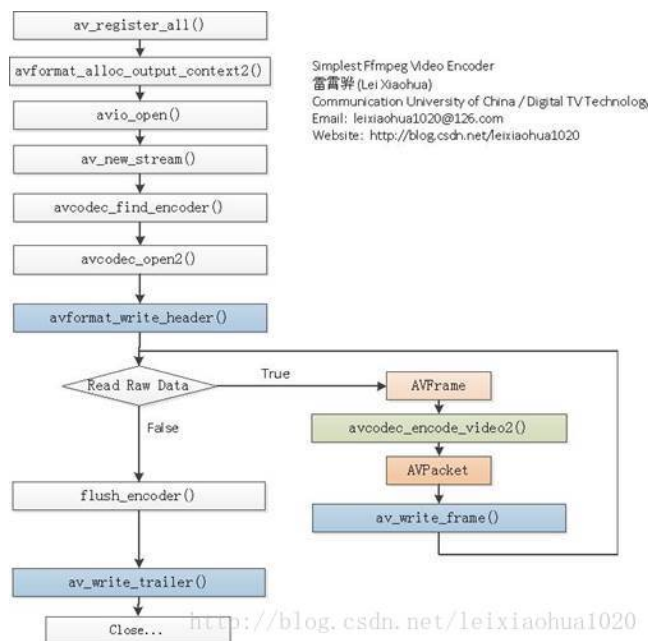
[最简单的基于FFmpeg的编码器-纯净版（不包含libavformat）](#)

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]    
1.  /**  
2.  * 最简单的基于FFmpeg的视频编码器  
3.  * Simplest FFmpeg Video Encoder  
4.  *  
5.  * 雷霄骅 Lei Xiaohua  
6.  * leixiaohua1020@126.com  
7.  * 中国传媒大学/数字电视技术  
8.  * Communication University of China / Digital TV Technology  
9.  * http://blog.csdn.net/leixiaohua1020  
10. *  
11. * 本程序实现了YUV像素数据编码为视频码流 (H264, MPEG2, VP8等等) 。  
12. * 是最简单的FFmpeg视频编码方面的教程。  
13. * 通过学习本例子可以了解FFmpeg的编码流程。  
14. * This software encode YUV420P data to H.264 bitstream.  
15. * It's the simplest video encoding software based on FFmpeg.  
16. * Suitable for beginner of FFmpeg  
17. */  
18.  
19. #include <stdio.h>  
20.  
21. #define __STDC_CONSTANT_MACROS  
22.  
23. #ifdef _WIN32  
24. //Windows  
25. extern "C"  
26. {  
27. #include "libavutil/opt.h"  
28. #include "libavcodec/avcodec.h"  
29. #include "libavformat/avformat.h"  
30. };  
31. #else  
32. //Linux...  
33. #ifdef __cplusplus  
34. extern "C"  
35. {  
36. #endif  
37. #include <libavutil/opt.h>  
38. #include <libavcodec/avcodec.h>  
39. #include <libavformat/avformat.h>  
40. #ifdef __cplusplus  
41. };  
42. #endif  
43. #endif  
44.  
45.  
46. int flush_encoder(AVFormatContext *fmt_ctx,unsigned int stream_index){  
47.     int ret;  
48.     int got_frame;  
49.     AVPacket enc_pkt;  
50.     if (!(fmt_ctx->streams[stream_index]->codec->capabilities &  
51.         CODEC_CAP_DELAY))  
52.         return 0;  
53.     while (1) {  
54.         enc_pkt.data = NULL;  
55.         enc_pkt.size = 0;  
56.         av_init_packet(&enc_pkt);  
57.         ret = avcodec_encode_video2 (fmt_ctx->streams[stream_index]->codec, &enc_pkt,  
58.             NULL, &got_frame);  
59.         av_frame_free(NULL);  
60.         if (ret < 0)  
61.             break;  
62.         if (!got_frame){  
63.             ret=0;  
64.             break;  
65.         }  
66.         printf("Flush Encoder: Succeed to encode 1 frame!\tsize:%5d\n",enc_pkt.size);  
67.         /* mux encoded frame */  
68.         ret = av_write_frame(fmt_ctx, &enc_pkt);  
69.         if (ret < 0)  
70.             break;  
71.     }  
72.     return ret;  
73. }  
74.  
75. int main(int argc, char* argv[])  
76. {  
77.     AVFormatContext* pFormatCtx;  
78.     AVOutputFormat* fmt;  
79.     AVStream* video_st;  
80.     AVCodecContext* pCodecCtx;
```

```

81.     AVCodec* pCodec;
82.     AVPacket pkt;
83.     uint8_t* picture_buf;
84.     AVFrame* pFrame;
85.     int picture_size;
86.     int y_size;
87.     int framecnt=0;
88.     //FILE *in_file = fopen("src01_480x272.yuv", "rb"); //Input raw YUV data
89.     FILE *in_file = fopen("../ds_480x272.yuv", "rb"); //Input raw YUV data
90.     int in_w=480,in_h=272; //Input data's width and height
91.     int framenum=100; //Frames to encode
92.     //const char* out_file = "src01.h264"; //Output Filepath
93.     //const char* out_file = "src01.ts";
94.     //const char* out_file = "src01.hevc";
95.     const char* out_file = "ds.h264";
96.
97.     av_register_all();
98.     //Method1.
99.     pFormatCtx = avformat_alloc_context();
100.    //Guess Format
101.    fmt = av_guess_format(NULL, out_file, NULL);
102.    pFormatCtx->oformat = fmt;
103.
104.    //Method 2.
105.    //avformat_alloc_output_context2(&pFormatCtx, NULL, NULL, out_file);
106.    //fmt = pFormatCtx->oformat;
107.
108.
109.    //Open output URL
110.    if (avio_open(&pFormatCtx->pb,out_file, AVIO_FLAG_READ_WRITE) < 0){
111.        printf("Failed to open output file! \n");
112.        return -1;
113.    }
114.
115.    video_st = avformat_new_stream(pFormatCtx, 0);
116.    //video_st->time_base.num = 1;
117.    //video_st->time_base.den = 25;
118.
119.    if (video_st==NULL){
120.        return -1;
121.    }
122.    //Param that must set
123.    pCodecCtx = video_st->codec;
124.    //pCodecCtx->codec_id =AV_CODEC_ID_HEVC;
125.    pCodecCtx->codec_id = fmt->video_codec;
126.    pCodecCtx->codec_type = AVMEDIA_TYPE_VIDEO;
127.    pCodecCtx->pix_fmt = AV_PIX_FMT_YUV420P;
128.    pCodecCtx->width = in_w;
129.    pCodecCtx->height = in_h;
130.    pCodecCtx->bit_rate = 400000;
131.    pCodecCtx->gop_size=250;
132.
133.    pCodecCtx->time_base.num = 1;
134.    pCodecCtx->time_base.den = 25;
135.
136.    //H264
137.    //pCodecCtx->me_range = 16;
138.    //pCodecCtx->max_qdiff = 4;
139.    //pCodecCtx->qcompress = 0.6;
140.    pCodecCtx->qmin = 10;
141.    pCodecCtx->qmax = 51;
142.
143.    //Optional Param
144.    pCodecCtx->max_b_frames=3;
145.
146.    // Set Option
147.    AVDictionary *param = 0;
148.    //H.264
149.    if(pCodecCtx->codec_id == AV_CODEC_ID_H264) {
150.        av_dict_set(&param, "preset", "slow", 0);
151.        av_dict_set(&param, "tune", "zerolatency", 0);
152.        //av_dict_set(&param, "profile", "main", 0);
153.    }
154.    //H.265
155.    if(pCodecCtx->codec_id == AV_CODEC_ID_H265){
156.        av_dict_set(&param, "preset", "ultrafast", 0);
157.        av_dict_set(&param, "tune", "zero-latency", 0);
158.    }
159.
160.    //Show some Information
161.    av_dump_format(pFormatCtx, 0, out_file, 1);
162.
163.    pCodec = avcodec_find_encoder(pCodecCtx->codec_id);
164.    if (!pCodec){
165.        printf("Can not find encoder! \n");
166.        return -1;
167.    }
168.    if (avcodec_open2(pCodecCtx, pCodec,&param) < 0){
169.        printf("Failed to open encoder! \n");
170.        return -1;
171.    }

```

```

172.
173.
174.     pFrame = av_frame_alloc();
175.     picture_size = avpicture_get_size(pCodecCtx->pix_fmt, pCodecCtx->width, pCodecCtx->height);
176.     picture_buf = (uint8_t *)av_malloc(picture_size);
177.     avpicture_fill((AVPicture *)pFrame, picture_buf, pCodecCtx->pix_fmt, pCodecCtx->width, pCodecCtx->height);
178.
179.     //Write File Header
180.     avformat_write_header(pFormatCtx,NULL);
181.
182.     av_new_packet(&pkt,picture_size);
183.
184.     y_size = pCodecCtx->width * pCodecCtx->height;
185.
186.     for (int i=0; i<framenum; i++){
187.         //Read raw YUV data
188.         if (fread(picture_buf, 1, y_size*3/2, in_file) <= 0){
189.             printf("Failed to read raw data! \n");
190.             return -1;
191.         }else if (feof(in_file)){
192.             break;
193.         }
194.         pFrame->data[0] = picture_buf;           // Y
195.         pFrame->data[1] = picture_buf+ y_size;   // U
196.         pFrame->data[2] = picture_buf+ y_size*5/4; // V
197.         //PTS
198.         //pFrame->pts=i;
199.         pFrame->pts=i*(video_st->time_base.den)/((video_st->time_base.num)*25);
200.         int got_picture=0;
201.         //Encode
202.         int ret = avcodec_encode_video2(pCodecCtx, &pkt,pFrame, &got_picture);
203.         if (ret < 0){
204.             printf("Failed to encode! \n");
205.             return -1;
206.         }
207.         if (got_picture==1){
208.             printf("Succeed to encode frame: %5d\tsize:%5d\n",framecnt,pkt.size);
209.             framecnt++;
210.             pkt.stream_index = video_st->index;
211.             ret = av_write_frame(pFormatCtx, &pkt);
212.             av_free_packet(&pkt);
213.         }
214.     }
215.     //Flush Encoder
216.     int ret = flush_encoder(pFormatCtx,0);
217.     if (ret < 0) {
218.         printf("Flushing encoder failed\n");
219.         return -1;
220.     }
221.
222.     //Write file trailer
223.     av_write_trailer(pFormatCtx);
224.
225.     //Clean
226.     if (video_st){
227.         avcodec_close(video_st->codec);
228.         av_free(pFrame);
229.         av_free(picture_buf);
230.     }
231.     avio_close(pFormatCtx->pb);
232.     avformat_free_context(pFormatCtx);
233.
234.     fclose(in_file);
235.
236.     return 0;
237. }

```

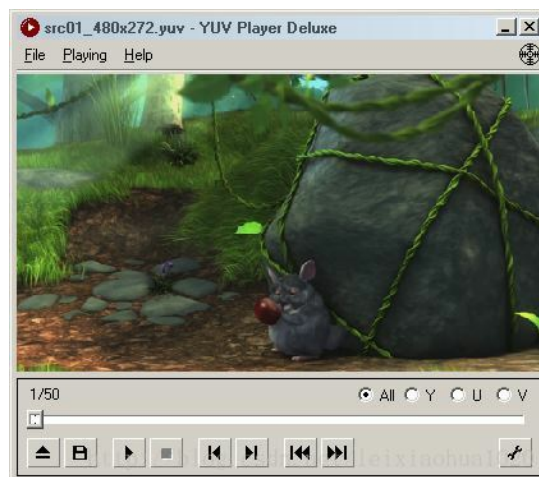
结果

软件运行截图（受限于文件体积，原始YUV帧数很少）：

```
CA\Windows\system32\cmd.exe
[libx264 @ 0015bc40] frame P:8 Avg QP:35.37 size: 534
[libx264 @ 0015bc40] mb I 16..4: 1.6% 60.0% 38.4%
[libx264 @ 0015bc40] mb P 16..4: 0.0% 0.2% 0.0% P16..4: 26.4% 7.0% 1.4%
0.0% 0.0% skip:65.0%
[libx264 @ 0015bc40] final ratefactor: 28.63
[libx264 @ 0015bc40] 8x8 transform intra:60.4% inter:56.4%
[libx264 @ 0015bc40] coded y,uvDC,uvAC intra: 93.7% 84.4% 62.7% inter: 3.4% 3.4%
0.2%
[libx264 @ 0015bc40] i16 v,h,dc,p: 0% 0% 12% 88%
[libx264 @ 0015bc40] i8 v,h,dc,ddl,ddr,vr,hd,vl,hu: 4% 11% 2% 11% 16% 11% 16%
10% 18%
[libx264 @ 0015bc40] i4 v,h,dc,ddl,ddr,vr,hd,vl,hu: 5% 10% 3% 13% 18% 12% 17%
8% 15%
[libx264 @ 0015bc40] i8c dc,h,v,p: 54% 25% 12% 9%
[libx264 @ 0015bc40] Weighted P-Frames: Y:0.0% UV:0.0%
[libx264 @ 0015bc40] kb/s:446.87
请按任意键继续. . .
```

<http://blog.csdn.net/leixiaohua1020>

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。

版权声明：本文为博主原创文章，未经博主允许不得转载。 <https://blog.csdn.net/leixiaohua1020/article/details/25430425>

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](#)

此PDF由spggy生成,请尊重原作者版权!!!

我的邮箱:liushide@163.com