

最简单的基于FFmpeg的内存读写的例子：内存转码器

2014年10月05日 13:30:35 阅读数：18391

最简单的基于FFmpeg的内存读写的例子系列文章列表：

[最简单的基于FFmpeg的内存读写的例子：内存播放器](#)

[最简单的基于FFmpeg的内存读写的例子：内存转码器](#)

上篇文章记录了一个基于FFmpeg的内存播放器，可以使用FFmpeg读取并播放内存中的数据。这篇文章记录一个基于FFmpeg的内存转码器。该转码器可以使用FFmpeg读取内存中的数据，转码为H.264之后再将数据输出到内存。

关于如何从内存读取数据，以及如何将数据输出到内存，可以参考文章：

[ffmpeg 从内存中读取数据\(或将数据输出到内存\)](#)

FFmpeg读写内存的关键点有2个：

1. 初始化自定义的AVIOContext，指定自定义的回调函数。
2. 自己写回调函数。注意函数的参数和返回值（尤其是返回值）。

转码实际上就是解码和编码的结合。该方面的知识可以参考文章：

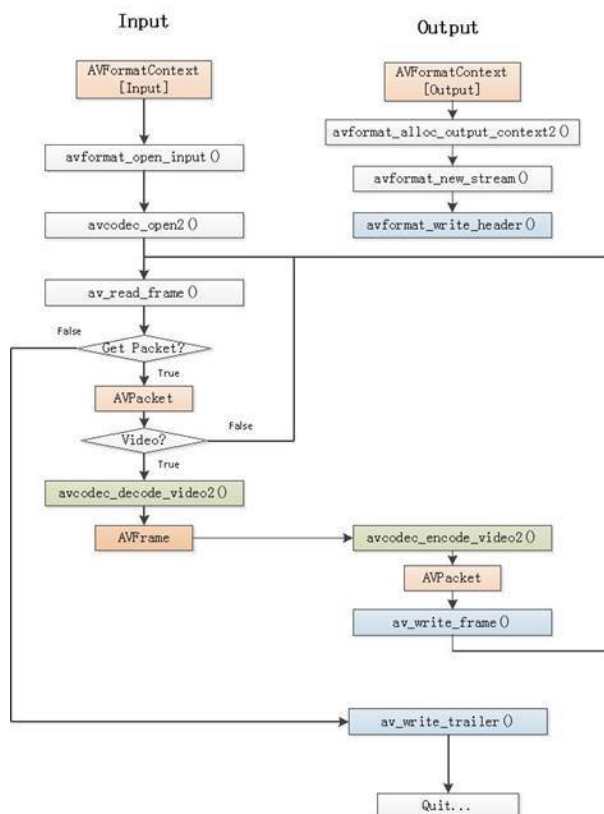
解码：[100行代码实现最简单的基于FFMPEG+SDL的视频播放器（SDL1.x）](#)

编码：[最简单的基于FFMPEG的视频编码器（YUV编码为H.264）](#)

转码：[最简单的基于FFMPEG的转码程序](#)

流程

程序的流程图如下图所示。从图中可以看出，首先分别初始化了输入和输出的AVFormatContext。然后首先解码输入的AVPacket，得到存储像素数据（YUV420P格式）的AVFrame，然后编码AVFrame为H.264的AVPacket，最后将编码后的AVPacket输出。



Simplest Ffmpeg Transcoder Simple
雷霄骅 (Lei Xiaohua)
Communication University of China / Digital TV Technology
Email: leixiaohua1020@126.com blog.csdn.net/leixiaohua1020
Website: <http://blog.csdn.net/leixiaohua1020>

代码

下面直接贴上代码：

```
[cpp]    
1.  /**  
2.  * 最简单的基于FFmpeg的内存读写例子（内存转码器）  
3.  *  Simplest FFmpeg mem Transcoder  
4.  *  
5.  *  雷霄骅，张晖  
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. *  本程序实现了任意格式视频数据（例如MPEG2）转码为H.264码流数据。  
12. *  本程序并不是对文件进行处理，而是对内存中的视频数据进行处理。  
13. *  它从内存读取数据，并且将转码后的数据输出到内存中。  
14. *  是最简单的使用FFmpeg读写内存的例子。  
15. *  
16. *  This software convert video bitstream (Such as MPEG2) to H.264  
17. *  bitstream. It read video bitstream from memory (not from a file),  
18. *  convert it to H.264 bitstream, and finally output to another memory.  
19. *  It's the simplest example to use FFmpeg to read (or write) from  
20. *  memory.  
21. *  
22. */  
23. #include <stdio.h>  
24.  
25. extern "C"  
26. {  
27.     #include "libavcodec/avcodec.h"  
28.     #include "libavformat/avformat.h"  
29.     #include "libavutil/avutil.h"  
30.     #include "libavutil/opt.h"  
31.     #include "libavutil/pixdesc.h"  
32. };  
33.  
34. FILE *fp_open;  
35. FILE *fp_write;  
36.  
37. //Read File  
38. int read_buffer(void *opaque, uint8_t *buf, int buf_size){  
39.     if(!feof(fp_open)){  
40.         int true_size=fread(buf,1,buf_size,fp_open);  
41.         return true_size;  
42.     }else{  
43.         return -1;  
44.     }  
45. }  
46.  
47. //Write File  
48. int write_buffer(void *opaque, uint8_t *buf, int buf_size){  
49.     if(!feof(fp_write)){  
50.         int true_size=fwrite(buf,1,buf_size,fp_write);  
51.         return true_size;  
52.     }else{  
53.         return -1;  
54.     }  
55. }  
56.  
57.  
58.  
59. int flush_encoder(AVFormatContext *fmt_ctx,unsigned int stream_index)  
60. {  
61.     int ret;  
62.     int got_frame;  
63.     AVPacket enc_pkt;  
64.     if (!(fmt_ctx->streams[stream_index]->codec->capabilities &  
65.         CODEC_CAP_DELAY))  
66.         return 0;  
67.     while (1) {  
68.         av_log(NULL, AV_LOG_INFO, "Flushing stream #%u encoder\n", stream_index);  
69.         //ret = encode_write_frame(NULL, stream_index, &got_frame);  
70.         enc_pkt.data = NULL;  
71.         enc_pkt.size = 0;  
72.         av_init_packet(&enc_pkt);  
73.         ret = avcodec_encode_video2 (fmt_ctx->streams[stream_index]->codec, &enc_pkt,  
74.             NULL, &got_frame);  
75.         av_frame_free(NULL);  
76.         if (ret < 0)  
77.             break;  
78.         if (!got_frame)  
79.             {ret=0;break;}  
80.         /* prepare packet for muxing */  
81.         enc_pkt.stream_index = stream_index;  
82.         enc_pkt.dts = av_rescale_q_rnd(enc_pkt.dts,  
83.             fmt_ctx->streams[stream_index]->codec->time_base,  
84.             fmt_ctx->streams[stream_index]->time_base,
```

```

84.         tmt_ctx->streams[stream_index]->time_base,
85.         (AVRounding)(AV_ROUND_NEAR_INF|AV_ROUND_PASS_MINMAX));
86.     enc_pkt.pts = av_rescale_q_rnd(enc_pkt.pts,
87.     fmt_ctx->streams[stream_index]->codec->time_base,
88.     fmt_ctx->streams[stream_index]->time_base,
89.     (AVRounding)(AV_ROUND_NEAR_INF|AV_ROUND_PASS_MINMAX));
90.     enc_pkt.duration = av_rescale_q(enc_pkt.duration,
91.     fmt_ctx->streams[stream_index]->codec->time_base,
92.     fmt_ctx->streams[stream_index]->time_base);
93.     av_log(NULL, AV_LOG_DEBUG, "Muxing frame\n");
94.     /* mux encoded frame */
95.     ret = av_write_frame(fmt_ctx, &enc_pkt);
96.     if (ret < 0)
97.         break;
98. }
99. return ret;
100. }
101.
102.
103. int main(int argc, char* argv[])
104. {
105.     int ret;
106.     AVFormatContext* ifmt_ctx=NULL;
107.     AVFormatContext* ofmt_ctx=NULL;
108.     AVPacket packet,enc_pkt;
109.     AVFrame *frame = NULL;
110.     enum AVMediaType type;
111.     unsigned int stream_index;
112.     unsigned int i=0;
113.     int got_frame,enc_got_frame;
114.
115.     AVStream *out_stream;
116.     AVStream *in_stream;
117.     AVCodecContext *dec_ctx, *enc_ctx;
118.     AVCodec *encoder;
119.
120.     fp_open = fopen("cuc60anniversary_start.ts", "rb"); //视频源文件
121.     fp_write=fopen("cuc60anniversary_start.h264","wb+"); //输出文件
122.
123.     av_register_all();
124.     ifmt_ctx=avformat_alloc_context();
125.     avformat_alloc_output_context2(&ofmt_ctx, NULL, "h264", NULL);
126.
127.     unsigned char* inbuffer=NULL;
128.     unsigned char* outbuffer=NULL;
129.     inbuffer=(unsigned char*)av_malloc(32768);
130.     outbuffer=(unsigned char*)av_malloc(32768);
131.
132.     /*open input file*/
133.     AVIOContext *avio_in =avio_alloc_context(inbuffer, 32768,0,NULL,read_buffer,NULL,NULL);
134.     if(avio_in==NULL)
135.         goto end;
136.     ifmt_ctx->pb=avio_in;
137.     ifmt_ctx->flags=AVFMT_FLAG_CUSTOM_IO;
138.     if ((ret = avformat_open_input(&ifmt_ctx, "whatever", NULL, NULL)) < 0) {
139.         av_log(NULL, AV_LOG_ERROR, "Cannot open input file\n");
140.         return ret;
141.     }
142.     if ((ret = avformat_find_stream_info(ifmt_ctx, NULL)) < 0) {
143.         av_log(NULL, AV_LOG_ERROR, "Cannot find stream information\n");
144.         return ret;
145.     }
146.     for (i = 0; i < ifmt_ctx->nb_streams; i++) {
147.         AVStream *stream;
148.         AVCodecContext *codec_ctx;
149.         stream = ifmt_ctx->streams[i];
150.         codec_ctx = stream->codec;
151.         /* Reencode video & audio and remux subtitles etc. */
152.         if (codec_ctx->codec_type == AVMEDIA_TYPE_VIDEO){
153.             /* Open decoder */
154.             ret = avcodec_open2(codec_ctx,
155.             avcodec_find_decoder(codec_ctx->codec_id), NULL);
156.             if (ret < 0) {
157.                 av_log(NULL, AV_LOG_ERROR, "Failed to open decoder for stream %u\n", i);
158.                 return ret;
159.             }
160.         }
161.     }
162.     //av_dump_format(ifmt_ctx, 0, "whatever", 0);
163.
164.     /*open output file*/
165.     AVIOContext *avio_out =avio_alloc_context(outbuffer, 32768,1,NULL,NULL,write_buffer,NULL);
166.     if(avio_out==NULL)
167.         goto end;
168.     //avio_out->write_packet=write_packet;
169.     ofmt_ctx->pb=avio_out;
170.     ofmt_ctx->flags=AVFMT_FLAG_CUSTOM_IO;
171.     for (i = 0; i < 1; i++) {
172.         out_stream = avformat_new_stream(ofmt_ctx, NULL);
173.         if (!out_stream) {
174.             av_log(NULL, AV_LOG_ERROR, "Failed allocating output stream\n");
175.             return AVERROR_UNKNOWN;

```

```

176.     }
177.     in_stream = ifmt_ctx->streams[i];
178.     dec_ctx = in_stream->codec;
179.     enc_ctx = out_stream->codec;
180.     if (dec_ctx->codec_type == AVMEDIA_TYPE_VIDEO)
181.     {
182.         encoder = avcodec_find_encoder(AV_CODEC_ID_H264);
183.         enc_ctx->height = dec_ctx->height;
184.         enc_ctx->width = dec_ctx->width;
185.         enc_ctx->sample_aspect_ratio = dec_ctx->sample_aspect_ratio;
186.         enc_ctx->pix_fmt = encoder->pix_fmts[0];
187.         enc_ctx->time_base = dec_ctx->time_base;
188.         //enc_ctx->time_base.num = 1;
189.         //enc_ctx->time_base.den = 25;
190.         //H264的必备选项，没有就会错
191.         enc_ctx->me_range=16;
192.         enc_ctx->max_qdiff = 4;
193.         enc_ctx->qmin = 10;
194.         enc_ctx->qmax = 51;
195.         enc_ctx->qcompress = 0.6;
196.         enc_ctx->refs=3;
197.         enc_ctx->bit_rate = 500000;
198.
199.         ret = avcodec_open2(enc_ctx, encoder, NULL);
200.         if (ret < 0) {
201.             av_log(NULL, AV_LOG_ERROR, "Cannot open video encoder for stream #%u\n", i);
202.             return ret;
203.         }
204.     }
205.     else if (dec_ctx->codec_type == AVMEDIA_TYPE_UNKNOWN) {
206.         av_log(NULL, AV_LOG_FATAL, "Elementary stream #%d is of unknown type, cannot proceed\n", i);
207.         return AVERROR_INVALIDDATA;
208.     } else {
209.         /* if this stream must be remuxed */
210.         ret = avcodec_copy_context(ofmt_ctx->streams[i]->codec,
211.                                     ifmt_ctx->streams[i]->codec);
212.         if (ret < 0) {
213.             av_log(NULL, AV_LOG_ERROR, "Copying stream context failed\n");
214.             return ret;
215.         }
216.     }
217.     if (ofmt_ctx->oformat->flags & AVFMT_GLOBALHEADER)
218.         enc_ctx->flags |= CODEC_FLAG_GLOBAL_HEADER;
219. }
220. //av_dump_format(ofmt_ctx, 0, "whatever", 1);
221. /* init muxer, write output file header */
222. ret = avformat_write_header(ofmt_ctx, NULL);
223. if (ret < 0) {
224.     av_log(NULL, AV_LOG_ERROR, "Error occurred when opening output file\n");
225.     return ret;
226. }
227.
228. i=0;
229. /* read all packets */
230. while (1) {
231.     i++;
232.     if ((ret = av_read_frame(ifmt_ctx, &packet)) < 0)
233.         break;
234.     stream_index = packet.stream_index;
235.     if (stream_index!=0)
236.         continue;
237.     type = ifmt_ctx->streams[packet.stream_index]->codec->codec_type;
238.     av_log(NULL, AV_LOG_DEBUG, "Demuxer gave frame of stream_index %u\n",
239.             stream_index);
240.
241.     av_log(NULL, AV_LOG_DEBUG, "Going to reencode the frame\n");
242.     frame = av_frame_alloc();
243.     if (!frame) {
244.         ret = AVERROR(ENOMEM);
245.         break;
246.     }
247.     packet.dts = av_rescale_q_rnd(packet.dts,
248.                                     ifmt_ctx->streams[stream_index]->time_base,
249.                                     ifmt_ctx->streams[stream_index]->codec->time_base,
250.                                     (AVRounding)(AV_ROUND_NEAR_INF|AV_ROUND_PASS_MINMAX));
251.     packet.pts = av_rescale_q_rnd(packet.pts,
252.                                     ifmt_ctx->streams[stream_index]->time_base,
253.                                     ifmt_ctx->streams[stream_index]->codec->time_base,
254.                                     (AVRounding)(AV_ROUND_NEAR_INF|AV_ROUND_PASS_MINMAX));
255.     ret = avcodec_decode_video2(ifmt_ctx->streams[stream_index]->codec, frame,
256.                                 &got_frame, &packet);
257.     printf("Decode 1 Packet\tsize:%d\tpts:%d\n", packet.size, packet.pts);
258.
259.     if (ret < 0) {
260.         av_frame_free(&frame);
261.         av_log(NULL, AV_LOG_ERROR, "Decoding failed\n");
262.         break;
263.     }
264.     if (got_frame) {
265.         frame->pts = av_frame_get_best_effort_timestamp(frame);
266.         frame->pict_type=AV_PICTURE_TYPE_NONE;

```

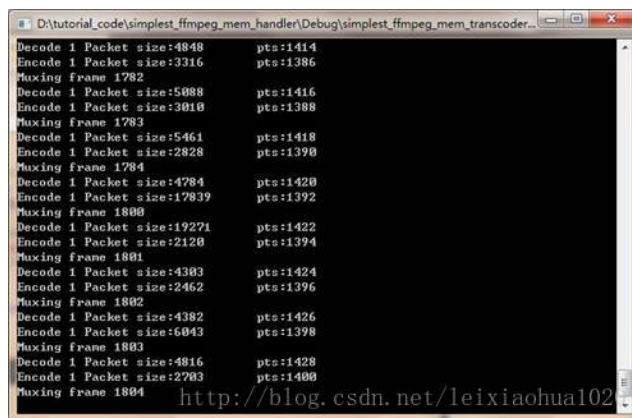
```

267.
268.     enc_pkt.data = NULL;
269.     enc_pkt.size = 0;
270.     av_init_packet(&enc_pkt);
271.     ret = avcodec_encode_video2 (ofmt_ctx->streams[stream_index]->codec, &enc_pkt,
272.         frame, &enc_got_frame);
273.
274.     printf("Encode 1 Packet\tsize:%d\tpts:%d\n",enc_pkt.size,enc_pkt.pts);
275.
276.     av_frame_free(&frame);
277.     if (ret < 0)
278.         goto end;
279.     if (!enc_got_frame)
280.         continue;
281.     /* prepare packet for muxing */
282.     enc_pkt.stream_index = stream_index;
283.     enc_pkt.dts = av_rescale_q_rnd(enc_pkt.dts,
284.         ofmt_ctx->streams[stream_index]->codec->time_base,
285.         ofmt_ctx->streams[stream_index]->time_base,
286.         (AVRounding)(AV_ROUND_NEAR_INF|AV_ROUND_PASS_MINMAX));
287.     enc_pkt.pts = av_rescale_q_rnd(enc_pkt.pts,
288.         ofmt_ctx->streams[stream_index]->codec->time_base,
289.         ofmt_ctx->streams[stream_index]->time_base,
290.         (AVRounding)(AV_ROUND_NEAR_INF|AV_ROUND_PASS_MINMAX));
291.     enc_pkt.duration = av_rescale_q(enc_pkt.duration,
292.         ofmt_ctx->streams[stream_index]->codec->time_base,
293.         ofmt_ctx->streams[stream_index]->time_base);
294.     av_log(NULL, AV_LOG_INFO, "Muxing frame %d\n",i);
295.     /* mux encoded frame */
296.     av_write_frame(ofmt_ctx,&enc_pkt);
297.     if (ret < 0)
298.         goto end;
299.     } else {
300.         av_frame_free(&frame);
301.     }
302.
303.     av_free_packet(&packet);
304. }
305.
306. /* flush encoders */
307. for (i = 0; i < 1; i++) {
308.     /* flush encoder */
309.     ret = flush_encoder(ofmt_ctx,i);
310.     if (ret < 0) {
311.         av_log(NULL, AV_LOG_ERROR, "Flushing encoder failed\n");
312.         goto end;
313.     }
314. }
315. av_write_trailer(ofmt_ctx);
316. end:
317. av_freep(&avio_in);
318. av_freep(&avio_out);
319. av_free(inbuffer);
320. av_free(outbuffer);
321. av_free_packet(&packet);
322. av_frame_free(&frame);
323. avformat_close_input(&ifmt_ctx);
324. avformat_free_context(ofmt_ctx);
325.
326. fcloseall();
327.
328. if (ret < 0)
329.     av_log(NULL, AV_LOG_ERROR, "Error occurred\n");
330. return (ret? 1:0);
331. }

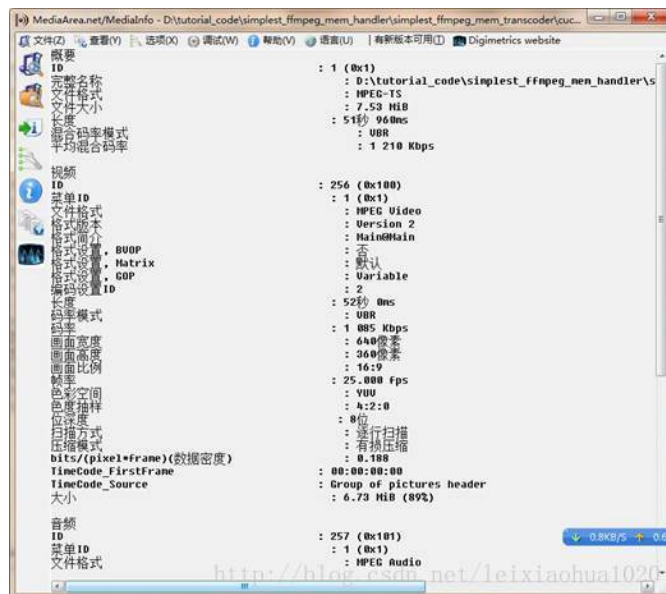
```

结果

程序运行的结果如下图所示。



转码前的视频信息使用MediaInfo查看如下图所示。



转码后的视频信息使用MediaInfo查看如下图所示。



下载

simplest ffmpeg mem handler

项目主页

SourceForge : <https://sourceforge.net/projects/simplestffmpegmemhandler/>

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

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

CSDN下载地址：

<http://download.csdn.net/detail/leixiaohua1020/8003731>

本工程包含两个FFmpeg读写内存的例子：

simplest_ffmpeg_mem_player：基于FFmpeg的内存播放器。

simplest_ffmpeg_mem_transcoder：基于FFmpeg的内存转码器。

更新-1.1 (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_mem_transcoder.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_mem_transcoder.cpp -g -o simplest_ffmpeg_mem_transcoder.exe \
2.  -I /usr/local/include -L /usr/local/lib \
3.  -lavcodec -lavformat -lavutil -lavdevice -lavfilter -lpostproc -lswresample -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_mem_transcoder.cpp -g -o simplest_ffmpeg_mem_transcoder.out \
2.  -I /usr/local/include -L /usr/local/lib \
3.  -lavcodec -lavformat -lavutil -lavdevice -lavfilter -lpostproc -lswresample -lswscale
```

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/leixiaohua1020/8445795>

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/39759623>

文章标签：[ffmpeg](#) [内存](#) [转码](#) [H.264](#)

个人分类：[我的开源项目](#) [FFMPEG](#)

所属专栏：[FFmpeg](#)

此PDF由spygg生成, 请尊重原作者版权!!!

我的邮箱:liushidc@163.com