

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

2013年03月08日 23:57:08 阅读数：145942

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](#)

简介

FFMPEG工程浩大，可以参考的书籍又不是很多，因此很多刚学习FFMPEG的人常常感觉到无从下手。我刚接触FFMPEG的时候也感觉不知从何学起。

因此我把自己做项目过程中实现的一个非常简单的视频播放器（大约100行代码）源代码传上来，以作备忘，同时方便新手学习FFMPEG。

该播放器虽然简单，但是几乎包含了使用FFMPEG播放一个视频所有必备的API，并且使用SDL显示解码出来的视频。

并且支持流媒体等多种视频输入，处于简单考虑，没有音频部分，同时视频播放采用直接延时40ms的方式

平台使用VC2010，使用了新版的FFMPEG类库。

SourceForge项目主页：

<https://sourceforge.net/projects/simplestffmpegplayer/>

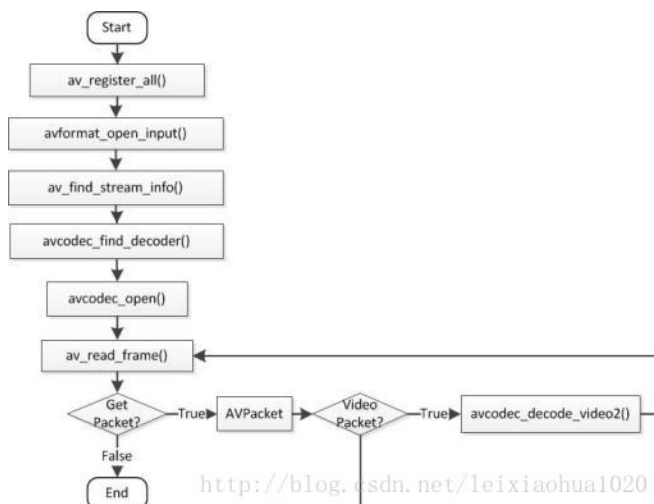
注：本文SDL采用1.x版本。另一版本采用SDL2.0，可参考：

基于FFMPEG+SDL的视频播放器 ver2（采用SDL2.0）：<http://blog.csdn.net/leixiaohua1020/article/details/38868499>

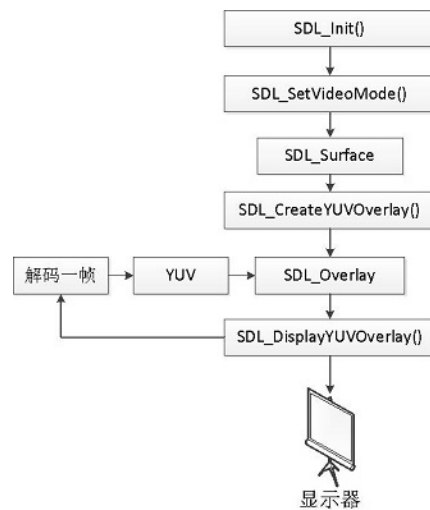
流程图

没想到这篇文章中介绍的播放器挺受FFMPEG初学者的欢迎，因此再次更新两张流程图，方便大家学习。此外在源代码上添加了注释，方便理解。

该播放器解码的流程用图的方式可以表示称如下形式：



SDL显示YUV图像的流程图：



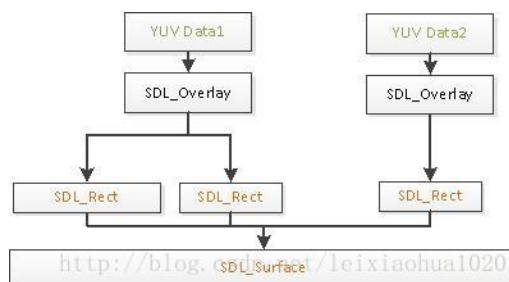
简单解释几句：

`SDL_Surface`就是使用SDL的时候弹出的那个窗口。在SDL1.x版本中，只可以创建一个`SDL_Surface`。

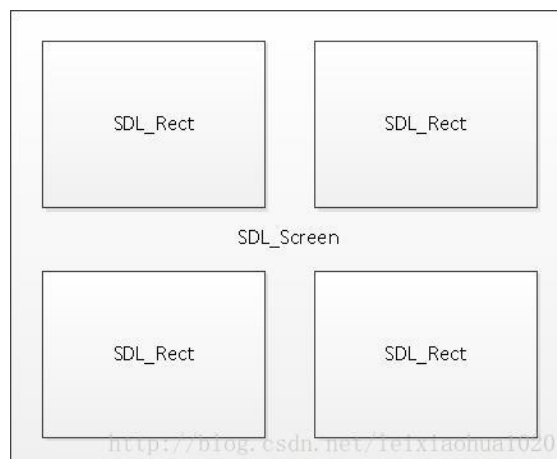
`SDL_Overlay`用于显示YUV数据。一个`SDL_Overlay`对应一帧YUV数据。

`SDL_Rect`用于确定`SDL_Overlay`显示的位置。注意：一个`SDL_Overlay`可以指定多个不同的`SDL_Rect`，这样就可以在`SDL_Surface`不同位置显示相同的内容。

它们的关系如下图所示：



下图举了个例子，指定了4个`SDL_Rect`，可以实现4分屏的显示。



simplest_ffmpeg_player（标准版）代码

```

1.  /**
2.   * 最简单的基于FFmpeg的视频播放器
3.   *  Simplest FFmpeg Player
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.  * 本程序实现了视频文件的解码和显示（支持HEVC, H.264, MPEG2等）。
12.  * 是最简单的FFmpeg视频解码方面的教程。
13.  * 通过学习本例子可以了解FFmpeg的解码流程。
14.  * This software is a simplest video player based on Ffmpeg.
15.  * Suitable for beginner of Ffmpeg.
16.  */
17.
18.
19. #include <stdio.h>
20.
21. #define __STDC_CONSTANT_MACROS
22.
23. #ifdef _WIN32
24. //Windows
25. extern "C"
26. {
27. #include "libavcodec/avcodec.h"
28. #include "libavformat/avformat.h"
29. #include "libswscale/swscale.h"
30. #include "SDL/SDL.h"
31. };
32. #else
33. //Linux...
34. #ifdef __cplusplus
35. extern "C"
36. {
37. #endif
38. #include <libavcodec/avcodec.h>
39. #include <libavformat/avformat.h>
40. #include <libswscale/swscale.h>
41. #include <SDL/SDL.h>
42. #ifdef __cplusplus
43. };
44. #endif
45. #endif
46.
47.
48. //Full Screen
49. #define SHOW_FULLSCREEN 0
50. //Output YUV420P
51. #define OUTPUT_YUV420P 0
52.
53.
54. int main(int argc, char* argv[])
55. {
56. //FFmpeg
57. AVFormatContext *pFormatCtx;
58. int i, videoindex;
59. AVCodecContext *pCodecCtx;
60. AVCodec *pCodec;
61. AVFrame *pFrame,*pFrameYUV;
62. AVPacket *packet;
63. struct SwsContext *img_convert_ctx;
64. //SDL
65. int screen_w,screen_h;
66. SDL_Surface *screen;
67. SDL_VideoInfo *vi;
68. SDL_Overlay *bmp;
69. SDL_Rect rect;
70.
71. FILE *fp_yuv;
72. int ret, got_picture;
73. char filepath[]="bigbuckbunny_480x272.h265";
74.
75. av_register_all();
76. avformat_network_init();
77.
78. pFormatCtx = avformat_alloc_context();
79.
80. if(avformat_open_input(&pFormatCtx,filepath,NULL,NULL)!=0){
81. printf("Couldn't open input stream.\n");
82. return -1;
83. }
84. if(avformat_find_stream_info(pFormatCtx,NULL)<0){
85. printf("Couldn't find stream information.\n");
86. return -1;
87. }
88. videoindex=-1;
89. for(i=0; i<pFormatCtx->nb_streams; i++){
90. if(pFormatCtx->streams[i]->codec->codec_type==AVMEDIA_TYPE_VIDEO){
91. videoindex=i;
92. break;
93. }
94. if(videoindex==-1){
95. printf("Didn't find a video stream.\n");
96. return -1;
97. }
98. pCodecCtx=pFormatCtx->streams[videoindex]->codec;
99. pCodec=avcodec_find_decoder(pCodecCtx->codec_id);
100. if(pCodec==NULL){

```

```

101.     printf("Codec not found.\n");
102.     return -1;
103. }
104. if(avcodec_open2(pCodecCtx, pCodec, NULL) < 0){
105.     printf("Could not open codec.\n");
106.     return -1;
107. }
108.
109. pFrame = av_frame_alloc();
110. pFrameYUV = av_frame_alloc();
111. //uint8_t *out_buffer = (uint8_t *)av_malloc(avpicture_get_size(PIX_FMT_YUV420P, pCodecCtx->width, pCodecCtx->height));
112. //avpicture_fill((AVPicture *)pFrameYUV, out_buffer, PIX_FMT_YUV420P, pCodecCtx->width, pCodecCtx->height);
113. //SDL-----
114. if(SDL_Init(SDL_INIT_VIDEO | SDL_INIT_AUDIO | SDL_INIT_TIMER)) {
115.     printf("Could not initialize SDL - %s\n", SDL_GetError());
116.     return -1;
117. }
118.
119.
120.
121. #if SHOW_FULLSCREEN
122.     vi = SDL_GetVideoInfo();
123.     screen_w = vi->current_w;
124.     screen_h = vi->current_h;
125.     screen = SDL_SetVideoMode(screen_w, screen_h, 0, SDL_FULLSCREEN);
126. #else
127.     screen_w = pCodecCtx->width;
128.     screen_h = pCodecCtx->height;
129.     screen = SDL_SetVideoMode(screen_w, screen_h, 0, 0);
130. #endif
131.
132.     if(!screen) {
133.         printf("SDL: could not set video mode - exiting:%s\n", SDL_GetError());
134.         return -1;
135.     }
136.
137.     bmp = SDL_CreateYUVOverlay(pCodecCtx->width, pCodecCtx->height, SDL_YV12_OVERLAY, screen);
138.
139.     rect.x = 0;
140.     rect.y = 0;
141.     rect.w = screen_w;
142.     rect.h = screen_h;
143.     //SDL End-----
144.
145.
146.     packet = (AVPacket *)av_malloc(sizeof(AVPacket));
147.     //Output Information-----
148.     printf("----- File Information ----- \n");
149.     av_dump_format(pFormatCtx, 0, filepath, 0);
150.     printf("----- \r");
151.
152.     #if OUTPUT_YUV420P
153.         fp_yuv = fopen("output.yuv", "wb+");
154.     #endif
155.
156.     SDL_WM_SetCaption("Simplest FFMpeg Player", NULL);
157.
158.     img_convert_ctx = sws_getContext(pCodecCtx->width, pCodecCtx->height, pCodecCtx->pix_fmt, pCodecCtx->width, pCodecCtx->height, PIX_FMT_YUV420P, SWS_BICUBIC, NULL, NULL, NULL);
159.     //-----
160.     while(av_read_frame(pFormatCtx, packet) >= 0){
161.         if(packet->stream_index == videoindex){
162.             //Decode
163.             ret = avcodec_decode_video2(pCodecCtx, pFrame, &got_picture, packet);
164.             if(ret < 0){
165.                 printf("Decode Error.\n");
166.                 return -1;
167.             }
168.             if(got_picture){
169.                 SDL_LockYUVOverlay(bmp);
170.                 pFrameYUV->data[0] = bmp->pixels[0];
171.                 pFrameYUV->data[1] = bmp->pixels[2];
172.                 pFrameYUV->data[2] = bmp->pixels[1];
173.                 pFrameYUV->linesize[0] = bmp->pitches[0];
174.                 pFrameYUV->linesize[1] = bmp->pitches[2];
175.                 pFrameYUV->linesize[2] = bmp->pitches[1];
176.                 sws_scale(img_convert_ctx, (const uint8_t* const*)pFrame->data, pFrame->linesize, 0,
177.                     pCodecCtx->height, pFrameYUV->data, pFrameYUV->linesize);
178.             #if OUTPUT_YUV420P
179.                 int y_size = pCodecCtx->width * pCodecCtx->height;
180.                 fwrite(pFrameYUV->data[0], 1, y_size, fp_yuv); //Y
181.                 fwrite(pFrameYUV->data[1], 1, y_size/4, fp_yuv); //U
182.                 fwrite(pFrameYUV->data[2], 1, y_size/4, fp_yuv); //V
183.             #endif
184.
185.                 SDL_UnlockYUVOverlay(bmp);
186.
187.                 SDL_DisplayYUVOverlay(bmp, &rect);
188.                 //Delay 40ms
189.                 SDL_Delay(40);
190.             }

```

```

191.     }
192.     av_free_packet(packet);
193. }
194.
195. //FIX: Flush Frames remained in Codec
196. while (1) {
197.     ret = avcodec_decode_video2(pCodecCtx, pFrame, &got_picture, packet);
198.     if (ret < 0)
199.         break;
200.     if (!got_picture)
201.         break;
202.     sws_scale(img_convert_ctx, (const uint8_t* const*)pFrame->data, pFrame->linesize, 0, pCodecCtx->height, pFrameYUV->data, pFrameYUV->linesize);
203.
204.     SDL_LockYUVOverlay(bmp);
205.     pFrameYUV->data[0]=bmp->pixels[0];
206.     pFrameYUV->data[1]=bmp->pixels[2];
207.     pFrameYUV->data[2]=bmp->pixels[1];
208.     pFrameYUV->linesize[0]=bmp->pitches[0];
209.     pFrameYUV->linesize[1]=bmp->pitches[2];
210.     pFrameYUV->linesize[2]=bmp->pitches[1];
211. #if OUTPUT_YUV420P
212.     int y_size=pCodecCtx->width*pCodecCtx->height;
213.     fwrite(pFrameYUV->data[0],1,y_size,fp_yuv); //Y
214.     fwrite(pFrameYUV->data[1],1,y_size/4,fp_yuv); //U
215.     fwrite(pFrameYUV->data[2],1,y_size/4,fp_yuv); //V
216. #endif
217.
218.     SDL_UnlockYUVOverlay(bmp);
219.     SDL_DisplayYUVOverlay(bmp, &rect);
220.     //Delay 40ms
221.     SDL_Delay(40);
222. }
223.
224. sws_freeContext(img_convert_ctx);
225.
226. #if OUTPUT_YUV420P
227.     fclose(fp_yuv);
228. #endif
229.
230.     SDL_Quit();
231.
232.     //av_free(out_buffer);
233.     av_free(pFrameYUV);
234.     avcodec_close(pCodecCtx);
235.     avformat_close_input(&pFormatCtx);
236.
237.     return 0;
238. }

```

1.1版之后，新添加了一个工程：simplest_ffmpeg_player_su（SU版）。

标准版在播放视频的时候，画面显示使用延时40ms的方式。这么做有两个后果：

(1) SDL弹出的窗口无法移动，一直显示是忙碌状态
(2) 画面显示并不是严格的40ms一帧，因为还没有考虑解码的时间。SU（SDL Update）版在视频解码的过程中，不再使用延时40ms的方式，而是创建了一个线程，每隔40ms发送一个自定义的消息，告知主函数进行解码显示。这样做之后：

(1) SDL弹出的窗口可以移动了
(2) 画面显示是严格的40ms一帧

simplest_ffmpeg_player_su（SU版）代码

```

1.  /**
2.   * 最简单的基于FFmpeg的视频播放器SU(SDL升级版)
3.   * Simplest FFMpeg Player (SDL Update)
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.  * 本程序实现了视频文件的解码和显示（支持HEVC, H.264, MPEG2等）。
12.  * 是最简单的FFmpeg视频解码方面的教程。
13.  * 通过学习本例子可以了解FFmpeg的解码流程。
14.  * 本版本中使用SDL消息机制刷新视频画面。
15.  * This software is a simplest video player based on FFmpeg.
16.  * Suitable for beginner of FFmpeg.
17.  *
18.  * Version:1.2
19.  *
20.  * 备注：

```

```

21.  * 标准版在播放视频的时候，画面显示使用延时40ms的方式。这么做有两个后果：
22.  *  (1) SDL弹出的窗口无法移动，一直显示是忙碌状态
23.  *  (2) 画面显示并不是严格的40ms一帧，因为还没有考虑解码的时间。
24.  *  SU (SDL Update) 版在视频解码的过程中，不再使用延时40ms的方式，而是创建了
25.  *  一个线程，每隔40ms发送一个自定义的消息，告知主函数进行解码显示。这样做之后：
26.  *  (1) SDL弹出的窗口可以移动了
27.  *  (2) 画面显示是严格的40ms一帧
28.  *  Remark:
29.  *  Standard Version use's SDL_Delay() to control video's frame rate, it has 2
30.  *  disadvantages:
31.  *  (1)SDL's Screen can't be moved and always "Busy".
32.  *  (2)Frame rate can't be accurate because it doesn't consider the time consumed
33.  *  by avcodec_decode_video2()
34.  *  SU (SDL Update) Version solved 2 problems above. It create a thread to send SDL
35.  *  Event every 40ms to tell the main loop to decode and show video frames.
36.  */
37.
38.
39. #include <stdio.h>
40.
41. #define __STDC_CONSTANT_MACROS
42.
43. #ifdef _WIN32
44. //Windows
45. extern "C"
46. {
47. #include "libavcodec/avcodec.h"
48. #include "libavformat/avformat.h"
49. #include "libswscale/swscale.h"
50. #include "SDL/SDL.h"
51. };
52. #else
53. //Linux...
54. #ifdef __cplusplus
55. extern "C"
56. {
57. #endif
58. #include <libavcodec/avcodec.h>
59. #include <libavformat/avformat.h>
60. #include <libswscale/swscale.h>
61. #include <SDL/SDL.h>
62. #ifdef __cplusplus
63. };
64. #endif
65. #endif
66.
67. //Refresh
68. #define SFM_REFRESH_EVENT (SDL_USEREVENT + 1)
69.
70. int thread_exit=0;
71. //Thread
72. int sfp_refresh_thread(void *opaque)
73. {
74.     SDL_Event event;
75.     while (thread_exit==0) {
76.         event.type = SFM_REFRESH_EVENT;
77.         SDL_PushEvent(&event);
78.         //Wait 40 ms
79.         SDL_Delay(40);
80.     }
81.     return 0;
82. }
83.
84.
85. int main(int argc, char* argv[])
86. {
87.     AVFormatContext *pFormatCtx;
88.     int i, videoindex;
89.     AVCodecContext *pCodecCtx;
90.     AVCodec *pCodec;
91.     AVFrame *pFrame,*pFrameYUV;
92.     AVPacket *packet;
93.     struct SwsContext *img_convert_ctx;
94.     //SDL
95.     int ret, got_picture;
96.     int screen_w=0,screen_h=0;
97.     SDL_Surface *screen;
98.     SDL_Overlay *bmp;
99.     SDL_Rect rect;
100.    SDL_Thread *video_tid;
101.    SDL_Event event;
102.
103.    char filepath[]="bigbuckbunny_480x272.h265";
104.    av_register_all();
105.    avformat_network_init();
106.    pFormatCtx = avformat_alloc_context();
107.
108.    if(avformat_open_input(&pFormatCtx,filepath,NULL,NULL)!=0){
109.        printf("Couldn't open input stream.\n");
110.        return -1;
111.    }
112.    if(avformat_find_stream_info(pFormatCtx,NULL)<0){

```

```

112.     if(avformat_find_stream_info(pFormatCtx,NULL)<0){
113.         printf("Couldn't find stream information.\n");
114.         return -1;
115.     }
116.     videoindex=-1;
117.     for(i=0; i<pFormatCtx->nb_streams; i++){
118.         if(pFormatCtx->streams[i]->codec->codec_type==AVMEDIA_TYPE_VIDEO){
119.             videoindex=i;
120.             break;
121.         }
122.     }
123.     if(videoindex==-1){
124.         printf("Didn't find a video stream.\n");
125.         return -1;
126.     }
127.     pCodecCtx=pFormatCtx->streams[videoindex]->codec;
128.     pCodec=avcodec_find_decoder(pCodecCtx->codec_id);
129.     if(pCodec==NULL)
130.     {
131.         printf("Codec not found.\n");
132.         return -1;
133.     }
134.     if(avcodec_open2(pCodecCtx, pCodec,NULL)<0)
135.     {
136.         printf("Could not open codec.\n");
137.         return -1;
138.     }
139.     pFrame=av_frame_alloc();
140.     pFrameYUV=av_frame_alloc();
141.     //uint8_t *out_buffer=(uint8_t *)av_malloc(avpicture_get_size(PIX_FMT_YUV420P, pCodecCtx->width, pCodecCtx->height));
142.     //avpicture_fill((AVPicture *)pFrameYUV, out_buffer, PIX_FMT_YUV420P, pCodecCtx->width, pCodecCtx->height);
143.     //-----SDL-----
144.     if(SDL_Init(SDL_INIT_VIDEO | SDL_INIT_AUDIO | SDL_INIT_TIMER)) {
145.         printf( "Could not initialize SDL - %s\n", SDL_GetError());
146.         return -1;
147.     }
148.
149.
150.     screen_w = pCodecCtx->width;
151.     screen_h = pCodecCtx->height;
152.     screen = SDL_SetVideoMode(screen_w, screen_h, 0,0);
153.
154.     if(!screen) {
155.         printf("SDL: could not set video mode - exiting:%s\n",SDL_GetError());
156.         return -1;
157.     }
158.
159.     bmp = SDL_CreateYUVOverlay(pCodecCtx->width, pCodecCtx->height,SDL_YV12_OVERLAY, screen);
160.
161.     rect.x = 0;
162.     rect.y = 0;
163.     rect.w = screen_w;
164.     rect.h = screen_h;
165.
166.     packet=(AVPacket *)av_malloc(sizeof(AVPacket));
167.
168.     printf("-----File Information-----\n");
169.     av_dump_format(pFormatCtx,0,filepath,0);
170.     printf("-----\n");
171.
172.
173.     img_convert_ctx = sws_getContext(pCodecCtx->width, pCodecCtx->height, pCodecCtx->pix_fmt, pCodecCtx->width, pCodecCtx->height, P
IX_FMT_YUV420P, SWS_BICUBIC, NULL, NULL, NULL);
174.     //-----
175.     video_tid = SDL_CreateThread(sfp_refresh_thread,NULL);
176.     //
177.     SDL_WM_SetCaption("Simple FFmpeg Player (SDL Update)",NULL);
178.
179.     //Event Loop
180.
181.     for (;;) {
182.         //Wait
183.         SDL_WaitEvent(&event);
184.         if(event.type==SFM_REFRESH_EVENT){
185.             //-----
186.             if(av_read_frame(pFormatCtx, packet)>=0){
187.                 if(packet->stream_index==videoindex){
188.                     ret = avcodec_decode_video2(pCodecCtx, pFrame, &got_picture, packet);
189.                     if(ret < 0){
190.                         printf("Decode Error.\n");
191.                         return -1;
192.                     }
193.                     if(got_picture){
194.
195.                         SDL_LockYUVOverlay(bmp);
196.                         pFrameYUV->data[0]=bmp->pixels[0];
197.                         pFrameYUV->data[1]=bmp->pixels[2];
198.                         pFrameYUV->data[2]=bmp->pixels[1];
199.                         pFrameYUV->linesize[0]=bmp->pitches[0];
200.                         pFrameYUV->linesize[1]=bmp->pitches[2];
201.                         pFrameYUV->linesize[2]=bmp->pitches[1];
202.                         sws_scale(img_convert_ctx, (const uint8_t**)pFrame->data, pFrame->linesize, 0, pCodecCtx->height, pFra

```

```

202.     sws_freeContext(img_convert_ctx); (const uint8_t *const /pFrame->data, pFrame->linesize, 0, pCodecCtx->height, pFrameYUV->data, pFrameYUV->linesize);
203.
204.         SDL_UnlockYUVOverlay(bmp);
205.
206.         SDL_DisplayYUVOverlay(bmp, &rect);
207.
208.     }
209. }
210.     av_free_packet(packet);
211. }else{
212.     //Exit Thread
213.     thread_exit=1;
214.     break;
215. }
216. }
217.
218. }
219.
220. SDL_Quit();
221.
222. sws_freeContext(img_convert_ctx);
223.
224. //-----
225. //av_free(out_buffer);
226. av_free(pFrameYUV);
227. avcodec_close(pCodecCtx);
228. avformat_close_input(&pFormatCtx);
229.
230. return 0;
231. }

```

simplest_ffmpeg_player_su（SU版）中将simplest_ffmpeg_player（标准版）中的循环做了更改。标准版中为播放视频的循环如下代码所示。

```

1. main(){
2.     //...
3.     while(av_read_frame(pFormatCtx, packet)>=0)
4.     {
5.         //Decode...
6.         SDL_Delay(40);
7.     }
8.     //...
9. }

```

可以看出标准版中使用SDL_Delay(40)控制视频的播放速度。这样有一些问题在前文中已经叙述。SU版定义了一个函数专门用于发送“解码和显示”的Event。

```

1. //自定义事件
2. //刷新画面
3. #define SFM_REFRESH_EVENT (SDL_USEREVENT + 1)
4.
5. int thread_exit=0;
6. //Thread
7. int sfp_refresh_thread(void *opaque)
8. {
9.     while (thread_exit==0) {
10.         SDL_Event event;
11.         event.type = SFM_REFRESH_EVENT;
12.         SDL_PushEvent(&event);
13.         //Wait 40 ms
14.         SDL_Delay(40);
15.     }
16.     return 0;
17. }

```

主函数形式如下。使用SDL_WaitEvent()等待Event进行解码和显示。

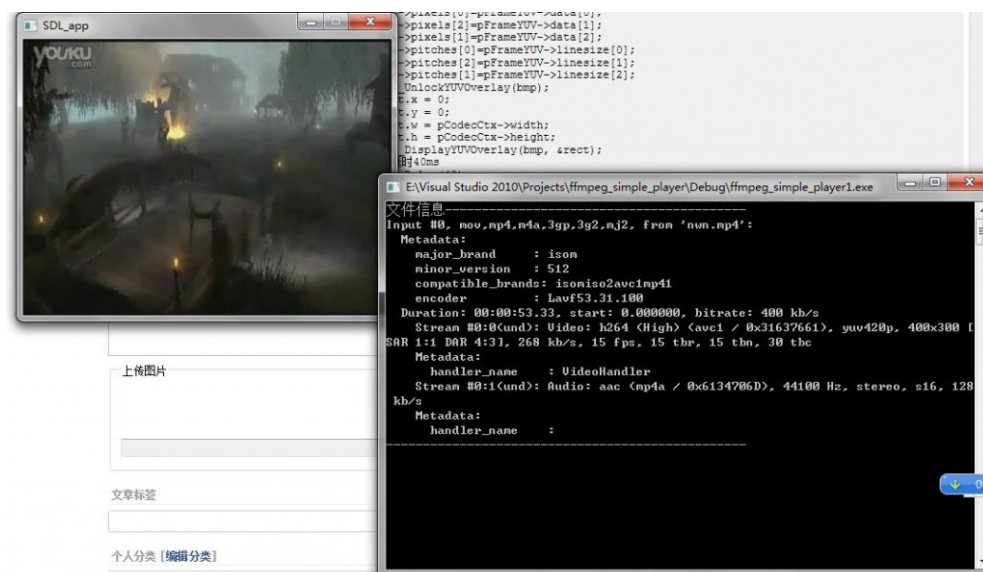

```

1.  main(){
2.      //...
3.      SDL_Thread *video_tid = SDL_CreateThread(sfp_refresh_thread,NULL);
4.      //Event Loop
5.      SDL_Event event;
6.      for (;;) {
7.          //Wait
8.          SDL_WaitEvent(&event);
9.          if(event.type==SFM_REFRESH_EVENT){
10.             //Decode...
11.         }
12.     }
13. }
14. //...
15. }

```

结果

软件运行截图：



完整工程下载地址：

<http://download.csdn.net/detail/leixiaohua1020/5122959>

更新（2014.5.10）=====

完整工程（更新版）下载地址：

<http://download.csdn.net/detail/leixiaohua1020/7319153>

注1：类库版本2014.5.6，已经支持HEVC以及VP9的解码，附带了这两种视频编码的码流文件。此外修改了个别变更的API函数，并且提高了一些程序的效率。

注2：新版FFmpeg类库Release下出现错误的解决方法如下：

（注：此方法适用于所有近期发布的FFmpeg类库）

VC工程属性里，linker->Optimization->References 选项，改成No(/OPT:NOREF)即可。

更新（2014.8.25）=====

simplest ffmpeg player 1.1

版本升级至1.1，变为2个项目：

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_player (标准版) 增加了以下两个选项（当然，代码量超过了100行）

1.输出解码后的YUV420P像素数据文件

2.全屏播放

以上两项可以通过文件前面的宏进行控制：

```
[cpp]
1. #define SHOW_FULLSCREEN 0
2. #define OUTPUT_YUV420P 0
```

另外修补了几个的函数，例如增加了SDL_Quit()等。

simplest_ffmpeg_player_su (SU版) 具体情况在上文中已经说明。

1.1版下载地址：<http://download.csdn.net/detail/leixiaohua1020/7814403>

SourceForge上已经更新。

更新 (2014.10.4) =====

simplest ffmpeg player 1.2

版本升级至1.2。

1.新版本在原版本的基础上增加了“flush_decoder”功能。当av_read_frame()循环退出的时候，实际上解码器中可能还包含剩余的几帧数据。因此需要通过“flush_decoder”将这几帧数据输出。“flush_decoder”功能简而言之即直接调用avcodec_decode_video2()获得AVFrame，而不再向解码器传递AVPacket。参考代码如下：

```
[cpp]
1. //FIX: Flush Frames remained in Codec
2. while (1) {
3.     ret = avcodec_decode_video2(pCodecCtx, pFrame, &got_picture, packet);
4.     if (ret < 0)
5.         break;
6.     if (!got_picture)
7.         break;
8.     sws_scale(img_convert_ctx, (const uint8_t* const*)pFrame->data, pFrame->linesize, 0, pCodecCtx->height, pFrameYUV->data, pFrameYUV->linesize);
9.     //处理...
10. }
```

具体信息参见文章：[avcodec_decode_video2\(\)解码视频后丢帧的问题解决](#)

2.为了更好地适应Linux等其他操作系统，做到可以跨平台，去掉了VC特有的一些函数。比如“#include <stdafx.h>”，“_tmain()”等等。

1.2版下载地址：<http://download.csdn.net/detail/leixiaohua1020/8001575>

SourceForge上已经更新。

Linux版本=====

Linux下代码下载地址：

<http://download.csdn.net/detail/leixiaohua1020/7696879>

这个是Linux下的代码，在Ubuntu下测试可以运行，前提是安装了FFmpeg和SDL（版本1.2）。

编译命令：

```
[plain]
1. gcc simplest_ffmpeg_player.c -g -o smp.out -lSDLmain -lSDL -lavformat -lavcodec -lavutil -lswscale
```

使用方法：

下列命令即可播放同一目录下的test.flv文件。

```
[plain]
1. ./smp.out test.flv
```

更新- 最终版 (2015.2.12) =====

simplest ffmpeg player 1 final

这是该播放器源代码的最后一次更新，以后会把更新的重点集中在基于FFmpeg和SDL2.0的视频播放器。这次考虑到了跨平台的要求，源代码的调整幅度比较大。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_player.cpp /MD /link SDL.lib SDLmain.lib avcodec.lib ^
9.  avformat.lib avutil.lib avdevice.lib avfilter.lib postproc.lib swscale.lib ^
10. /SUBSYSTEM:WINDOWS /OPT:NOREF
11. exit
```

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_player.cpp -g -o simplest_ffmpeg_player.exe \
2.  -I /usr/local/include -L /usr/local/lib \
3.  -lmingw32 -lSDLmain -lSDL -lavformat -lavcodec -lavutil -lswscale
```

GCC(Linux)：Linux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_player.cpp -g -o simplest_ffmpeg_player.out \
2.  -I /usr/local/include -L /usr/local/lib -lSDLmain -lSDL -lavformat -lavcodec -lavutil -lswscale
```

GCC(MacOS)：Mac终端下运行compile_gcc_mac.sh即可使用Mac 的GCC进行编译，Mac的GCC和Linux的GCC差别不大，但是使用SDL1.2的时候，必须加上“-framework Cocoa”参数，否则编译无法通过。编译命令如下。

```
[plain]
1.  gcc simplest_ffmpeg_player.cpp -g -o simplest_ffmpeg_player.out \
2.  -framework Cocoa -I /usr/local/include -L /usr/local/lib \
3.  -lSDLmain -lSDL -lavformat -lavcodec -lavutil -lswscale
```

PS：相关的编译命令已经保存到了工程文件夹中

此外，该版本修正了在某些系统下（例如部分Ubuntu）SDL绿屏显示的问题，经过测试已经不再有绿屏现象。

CSDN下载地址：<http://download.csdn.net/detail/leixiaohua1020/8443253>

SourceForge上已经更新。

FFMPEG相关资料

SDL GUIDE 中文译本

<http://download.csdn.net/detail/leixiaohua1020/6389841>

ffdoc（FFMPEG的最完整教程）

<http://download.csdn.net/detail/leixiaohua1020/6377803>

如何用FFmpeg编写一个简单播放器

<http://download.csdn.net/detail/leixiaohua1020/6373783>

补充问题

补充1：旧版程序有一个小BUG，就是sws_getContext()之后，需要调用sws_freeContext()。否则长时间运行的话，会出现内存泄露的状况。更新版已经修复。

补充2：有人会疑惑，为什么解码后的pFrame不直接用于显示，而是调用swscale()转换之后进行显示？

如果不进行转换，而是直接调用SDL进行显示的话，会发现显示出来的图像是混乱的。关键在于解码后的pFrame的linesize里存储的不是图像的宽度，而是比宽度大一些的一个值。其原因目前还没有仔细调查（大概是出于性能的考虑）。例如分辨率为480x272的图像，解码后的视频的linesize[0]为512，而不是480。以第1行亮度像素（pFrame->data[0]）为例，从0-480存储的是亮度数据，而从480-512则存储的是无效的数据。因此需要使用swscale()进行转换。转换后去除了无效数据，linesize[0]变为480。就可以正常显示了。



文章标签： FFMPEG 播放器 解码 SDL

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