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

2014年08月31日 01:20:14　阅读数：79214

==========

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

之前做过一个FFMPEG+SDL的简单播放器:《 100行代码实现最简单的基于FFMPEG+SDL的视频播放器 》。该播放器采用SDL1.2显示视频。最近有不少人反映SDL已经升级到2.0版本了，甚至官网的Wiki上都只有SDL2.0的文档了，因此下载了SDL 2.0 并且进行了简单的研究。随后对此前的播放器进行了修改，将SDL1.2换成了SDL2.0。

注：《 100行代码实现最简单的基于FFMPEG+SDL的视频播放器 》文章中提到的很多知识这里不再重复记录。本文重点记录SDL1.2与SDL2.0的不同。

平台使用了VC2010，FFmpeg类库使用了最近的版本，SDL使用2.0版本。

Simplest FFmpeg Player 2

项目主页

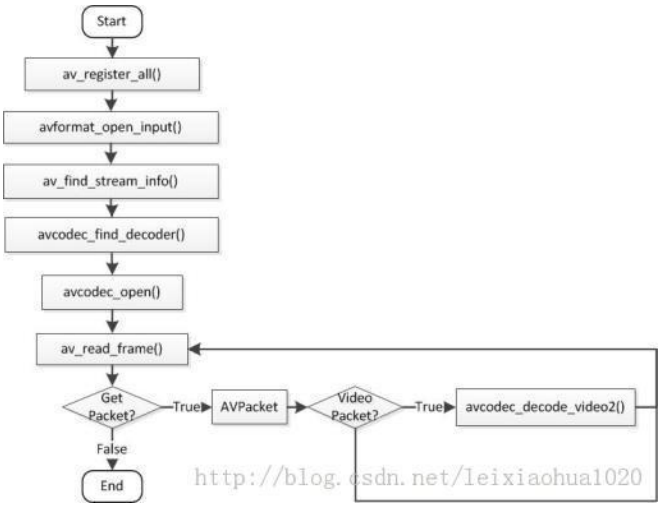
SourceForge： https://sourceforge.net/projects/simplestffmpegplayer/

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

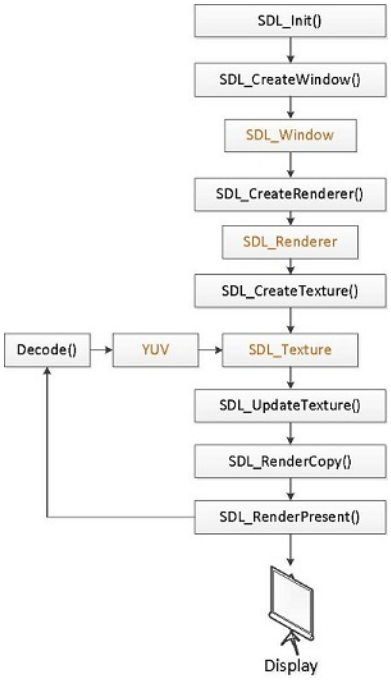
开源中国： http://git.oschina.net/leixiaohua1020/simplest_ffmpeg_player

流程图

FFmpeg解码一个视频流程如下图所示：



SDL2.0显示YUV的流程图：



对比SDL1.2的流程图，发现变化还是很大的。几乎所有的API都发生了变化。但是函数和变量有一定的对应关系：

SDL_SetVideoMode()————SDL_CreateWindow()

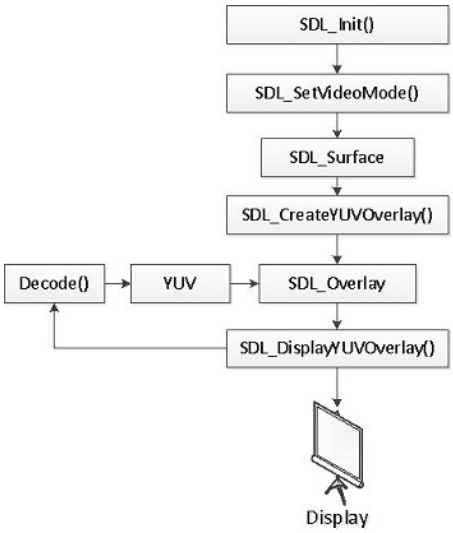
SDL_Surface————SDL_Window

SDL_CreateYUVOverlay()————SDL_CreateTexture()

SDL_Overlay————SDL_Texture

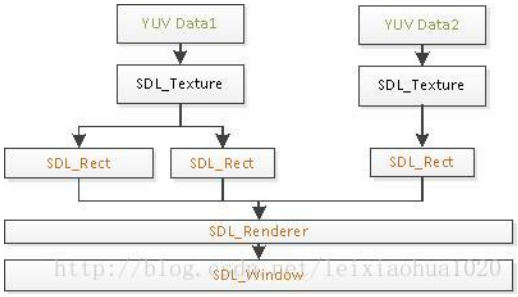
不再一一例举。

下图为SDL1.x显示YUV的流程图。

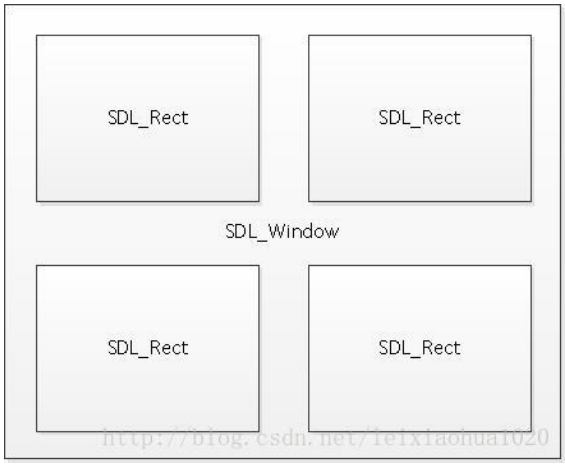


简单解释各个变量的作用：

SDL_Window就是使用SDL的时候弹出的那个窗口。在SDL1.x版本中，只可以创建一个一个窗口。在SDL2.0版本中，可以创建多个窗口。
SDL_Texture用于显示YUV数据。一个SDL_Texture对应一帧YUV数据。
SDL_Renderer用于渲染SDL_Texture至SDL_Window。
SDL_Rect用于确定SDL_Texture显示的位置。注意：一个SDL_Texture可以指定多个不同的SDL_Rect，这样就可以在SDL_Window不同位置显示相同的内容（使用SDL_RenderCopy()函数）。
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

最基础的版本，学习的开始。

```cpp
/**
 * 最简单的基于FFmpeg的视频播放器 2
 * Simplest FFmpeg Player 2
 *
 * 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
 *
 * 第2版使用SDL2.0取代了第一版中的SDL1.2
 * Version 2 use SDL 2.0 instead of SDL 1.2 in version 1.
 *
 * 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。
 * This software is a simplest video player based on FFmpeg.
 * Suitable for beginner of FFmpeg.
 *
 */



#include <stdio.h>

#define __STDC_CONSTANT_MACROS

#ifdef _WIN32
//Windows
extern "C"
{
#include "libavcodec/avcodec.h"
#include "libavformat/avformat.h"
#include "libswscale/swscale.h"
#include "libavutil/imgutils.h"
#include "SDL2/SDL.h"
};
#else
//Linux...
#ifdef __cplusplus
extern "C"
{
#endif
#include <libavcodec/avcodec.h>
#include <libavformat/avformat.h>
#include <libswscale/swscale.h>
#include <SDL2/SDL.h>
#include <libavutil/imgutils.h>
#ifdef __cplusplus
};
#endif
#endif

//Output YUV420P data as a file
#define OUTPUT_YUV420P 0

int main(int argc, char* argv[])
{
    AVFormatContext *pFormatCtx;
    int             i, videoindex;
    AVCodecContext  *pCodecCtx;
    AVCodec         *pCodec;
    AVFrame *pFrame,*pFrameYUV;
    unsigned char *out_buffer;
    AVPacket *packet;
    int y_size;
    int ret, got_picture;
    struct SwsContext *img_convert_ctx;

    char filepath[]="bigbuckbunny_480x272.h265";
    //SDL---------------------------
    int screen_w=0,screen_h=0;
    SDL_Window *screen;
    SDL_Renderer* sdlRenderer;
    SDL_Texture* sdlTexture;
    SDL_Rect sdlRect;

    FILE *fp_yuv;

    av_register_all();
    avformat_network_init();
    pFormatCtx = avformat_alloc_context();

    if(avformat_open_input(&pFormatCtx,filepath,NULL,NULL)!=0){
        printf("Couldn't open input stream.\n");
        return -1;
    }
    if(avformat_find_stream_info(pFormatCtx,NULL)<0){
        printf("Couldn't find stream information.\n");
        return -1;
    }
    videoindex=-1;
    for(i=0; i<pFormatCtx->nb_streams; i++)
        if(pFormatCtx->streams[i]->codec->codec_type==AVMEDIA_TYPE_VIDEO){
            videoindex=i;
            break;
        }
```

```
    if(videoindex==-1){
        printf("Didn't find a video stream.\n");
        return -1;
    }

    pCodecCtx=pFormatCtx->streams[videoindex]->codec;
    pCodec=avcodec_find_decoder(pCodecCtx->codec_id);
    if(pCodec==NULL){
        printf("Codec not found.\n");
        return -1;
    }
    if(avcodec_open2(pCodecCtx, pCodec,NULL)<0){
        printf("Could not open codec.\n");
        return -1;
    }

    pFrame=av_frame_alloc();
    pFrameYUV=av_frame_alloc();
    out_buffer=(unsigned char *)av_malloc(av_image_get_buffer_size(AV_PIX_FMT_YUV420P,  pCodecCtx->width, pCodecCtx->height,1));
    av_image_fill_arrays(pFrameYUV->data, pFrameYUV->linesize,out_buffer,
        AV_PIX_FMT_YUV420P,pCodecCtx->width, pCodecCtx->height,1);

    packet=(AVPacket *)av_malloc(sizeof(AVPacket));
    //Output Info-----------------------------
    printf("--------------- File Information ----------------\n");
    av_dump_format(pFormatCtx,0,filepath,0);
    printf("-------------------------------------------------\n");
    img_convert_ctx = sws_getContext(pCodecCtx->width, pCodecCtx->height, pCodecCtx->pix_fmt,
        pCodecCtx->width, pCodecCtx->height, AV_PIX_FMT_YUV420P, SWS_BICUBIC, NULL, NULL, NULL);

#if OUTPUT_YUV420P
    fp_yuv=fopen("output.yuv","wb+");
#endif

    if(SDL_Init(SDL_INIT_VIDEO | SDL_INIT_AUDIO | SDL_INIT_TIMER)) {
        printf( "Could not initialize SDL - %s\n", SDL_GetError());
        return -1;
    }

    screen_w = pCodecCtx->width;
    screen_h = pCodecCtx->height;
    //SDL 2.0 Support for multiple windows
    screen = SDL_CreateWindow("Simplest ffmpeg player's Window", SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED,
        screen_w, screen_h,
        SDL_WINDOW_OPENGL);

    if(!screen) {
        printf("SDL: could not create window - exiting:%s\n",SDL_GetError());
        return -1;
    }

    sdlRenderer = SDL_CreateRenderer(screen, -1, 0);
    //IYUV: Y + U + V  (3 planes)
    //YV12: Y + V + U  (3 planes)
    sdlTexture = SDL_CreateTexture(sdlRenderer, SDL_PIXELFORMAT_IYUV, SDL_TEXTUREACCESS_STREAMING,pCodecCtx->width,pCodecCtx->height
);

    sdlRect.x=0;
    sdlRect.y=0;
    sdlRect.w=screen_w;
    sdlRect.h=screen_h;

    //SDL End----------------------
    while(av_read_frame(pFormatCtx, packet)>=0){
        if(packet->stream_index==videoindex){
            ret = avcodec_decode_video2(pCodecCtx, pFrame, &got_picture, packet);
            if(ret < 0){
                printf("Decode Error.\n");
                return -1;
            }
            if(got_picture){
                sws_scale(img_convert_ctx, (const unsigned char* const*)pFrame->data, pFrame->linesize, 0, pCodecCtx->height,
                    pFrameYUV->data, pFrameYUV->linesize);

#if OUTPUT_YUV420P
                y_size=pCodecCtx->width*pCodecCtx->height;
                fwrite(pFrameYUV->data[0],1,y_size,fp_yuv);    //Y
                fwrite(pFrameYUV->data[1],1,y_size/4,fp_yuv);  //U
                fwrite(pFrameYUV->data[2],1,y_size/4,fp_yuv);  //V
#endif
                //SDL---------------------------
#if 0
                SDL_UpdateTexture( sdlTexture, NULL, pFrameYUV->data[0], pFrameYUV->linesize[0] );
#else
                SDL_UpdateYUVTexture(sdlTexture, &sdlRect,
                pFrameYUV->data[0], pFrameYUV->linesize[0],
                pFrameYUV->data[1], pFrameYUV->linesize[1],
                pFrameYUV->data[2], pFrameYUV->linesize[2]);
#endif

                SDL_RenderClear( sdlRenderer );
```

```
                SDL_RenderCopy( sdlRenderer, sdlTexture,  NULL, &sdlRect);
                SDL_RenderPresent( sdlRenderer );
                //SDL End-----------------------
                //Delay 40ms
                SDL_Delay(40);
            }
        }
        av_free_packet(packet);
    }
    //flush decoder
    //FIX: Flush Frames remained in Codec
    while (1) {
        ret = avcodec_decode_video2(pCodecCtx, pFrame, &got_picture, packet);
        if (ret < 0)
            break;
        if (!got_picture)
            break;
        sws_scale(img_convert_ctx, (const unsigned char* const*)pFrame->data, pFrame->linesize, 0, pCodecCtx->height,
            pFrameYUV->data, pFrameYUV->linesize);
#if OUTPUT_YUV420P
        int y_size=pCodecCtx->width*pCodecCtx->height;
        fwrite(pFrameYUV->data[0],1,y_size,fp_yuv);    //Y
        fwrite(pFrameYUV->data[1],1,y_size/4,fp_yuv);  //U
        fwrite(pFrameYUV->data[2],1,y_size/4,fp_yuv);  //V
#endif
        //SDL---------------------------
        SDL_UpdateTexture( sdlTexture, &sdlRect, pFrameYUV->data[0], pFrameYUV->linesize[0] );
        SDL_RenderClear( sdlRenderer );
        SDL_RenderCopy( sdlRenderer, sdlTexture,  NULL, &sdlRect);
        SDL_RenderPresent( sdlRenderer );
        //SDL End-----------------------
        //Delay 40ms
        SDL_Delay(40);
    }

    sws_freeContext(img_convert_ctx);

#if OUTPUT_YUV420P
    fclose(fp_yuv);
#endif

    SDL_Quit();

    av_frame_free(&pFrameYUV);
    av_frame_free(&pFrame);
    avcodec_close(pCodecCtx);
    avformat_close_input(&pFormatCtx);

    return 0;
}
```

simplest_ffmpeg_player_su（SU版）代码

标准版的基础之上引入了SDL的Event。

效果如下：

（1）SDL弹出的窗口可以移动了
（2）画面显示是严格的40ms一帧

```cpp
/**
 * 最简单的基于FFmpeg的视频播放器2(SDL升级版)
 * Simplest FFmpeg Player 2(SDL Update)
 *
 * 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
 *
 * 第2版使用SDL2.0取代了第一版中的SDL1.2
 * Version 2 use SDL 2.0 instead of SDL 1.2 in version 1.
 *
 * 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。
 * 本版本中使用SDL消息机制刷新视频画面。
 * This software is a simplest video player based on FFmpeg.
 * Suitable for beginner of FFmpeg.
 *
 * 备注:
 * 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
```

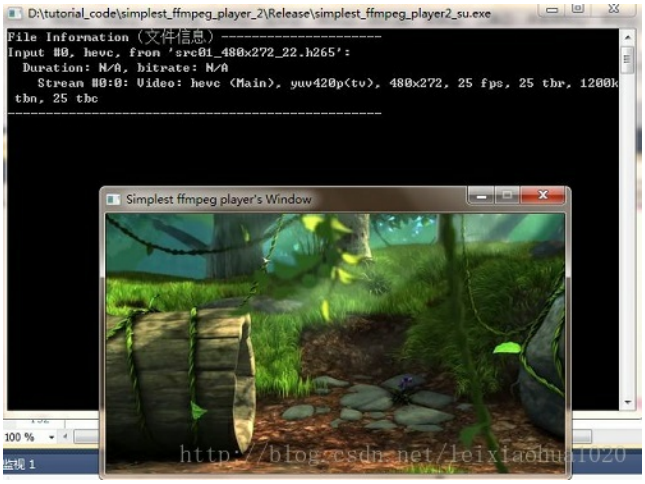
```
 * （2）画面显示并不是严格的40ms一帧，因为还没有考虑解码的时间。
 * SU（SDL Update）版在视频解码的过程中，不再使用延时40ms的方式，而是创建了
 * 一个线程，每隔40ms发送一个自定义的消息，告知主函数进行解码显示。这样做之后：
 * （1）SDL弹出的窗口可以移动了
 * （2）画面显示是严格的40ms一帧
 * Remark:
 * Standard Version use's SDL_Delay() to control video's frame rate, it has 2
 * disadvantages:
 * (1)SDL's Screen can't be moved and always "Busy".
 * (2)Frame rate can't be accurate because it doesn't consider the time consumed
 * by avcodec_decode_video2()
 * SU（SDL Update）Version solved 2 problems above. It create a thread to send SDL
 * Event every 40ms to tell the main loop to decode and show video frames.
 */

#include <stdio.h>

#define __STDC_CONSTANT_MACROS

#ifdef _WIN32
//Windows
extern "C"
{
#include "libavcodec/avcodec.h"
#include "libavformat/avformat.h"
#include "libswscale/swscale.h"
#include "libavutil/imgutils.h"
#include "SDL2/SDL.h"
};
#else
//Linux...
#ifdef __cplusplus
extern "C"
{
#endif
#include <libavcodec/avcodec.h>
#include <libavformat/avformat.h>
#include <libswscale/swscale.h>
#include <libavutil/imgutils.h>
#include <SDL2/SDL.h>
#ifdef __cplusplus
};
#endif
#endif

//Refresh Event
#define SFM_REFRESH_EVENT  (SDL_USEREVENT + 1)

#define SFM_BREAK_EVENT  (SDL_USEREVENT + 2)

int thread_exit=0;
int thread_pause=0;

int sfp_refresh_thread(void *opaque){
    thread_exit=0;
    thread_pause=0;

    while (!thread_exit) {
        if(!thread_pause){
            SDL_Event event;
            event.type = SFM_REFRESH_EVENT;
            SDL_PushEvent(&event);
        }
        SDL_Delay(40);
    }
    thread_exit=0;
    thread_pause=0;
    //Break
    SDL_Event event;
    event.type = SFM_BREAK_EVENT;
    SDL_PushEvent(&event);

    return 0;
}


int main(int argc, char* argv[])
{

    AVFormatContext *pFormatCtx;
    int             i, videoindex;
    AVCodecContext  *pCodecCtx;
    AVCodec         *pCodec;
    AVFrame *pFrame,*pFrameYUV;
    unsigned char *out_buffer;
    AVPacket *packet;
    int ret, got_picture;

    //------------SDL----------------
    int screen_w,screen_h;
    SDL_Window *screen;
```

```
    SDL_Renderer* sdlRenderer;
    SDL_Texture* sdlTexture;
    SDL_Rect sdlRect;
    SDL_Thread *video_tid;
    SDL_Event event;

    struct SwsContext *img_convert_ctx;

    //char filepath[]="bigbuckbunny_480x272.h265";
    char filepath[]="Titanic.ts";

    av_register_all();
    avformat_network_init();
    pFormatCtx = avformat_alloc_context();

    if(avformat_open_input(&pFormatCtx,filepath,NULL,NULL)!=0){
        printf("Couldn't open input stream.\n");
        return -1;
    }
    if(avformat_find_stream_info(pFormatCtx,NULL)<0){
        printf("Couldn't find stream information.\n");
        return -1;
    }
    videoindex=-1;
    for(i=0; i<pFormatCtx->nb_streams; i++)
        if(pFormatCtx->streams[i]->codec->codec_type==AVMEDIA_TYPE_VIDEO){
            videoindex=i;
            break;
        }
    if(videoindex==-1){
        printf("Didn't find a video stream.\n");
        return -1;
    }
    pCodecCtx=pFormatCtx->streams[videoindex]->codec;
    pCodec=avcodec_find_decoder(pCodecCtx->codec_id);
    if(pCodec==NULL){
        printf("Codec not found.\n");
        return -1;
    }
    if(avcodec_open2(pCodecCtx, pCodec,NULL)<0){
        printf("Could not open codec.\n");
        return -1;
    }
    pFrame=av_frame_alloc();
    pFrameYUV=av_frame_alloc();

    out_buffer=(unsigned char *)av_malloc(av_image_get_buffer_size(AV_PIX_FMT_YUV420P,  pCodecCtx->width, pCodecCtx->height,1));
    av_image_fill_arrays(pFrameYUV->data, pFrameYUV->linesize,out_buffer,
        AV_PIX_FMT_YUV420P,pCodecCtx->width, pCodecCtx->height,1);

    //Output Info-----------------------------
    printf("--------------- File Information --------------\n");
    av_dump_format(pFormatCtx,0,filepath,0);
    printf("-------------------------------------------------\r");

    img_convert_ctx = sws_getContext(pCodecCtx->width, pCodecCtx->height, pCodecCtx->pix_fmt,
        pCodecCtx->width, pCodecCtx->height, AV_PIX_FMT_YUV420P, SWS_BICUBIC, NULL, NULL, NULL);


    if(SDL_Init(SDL_INIT_VIDEO | SDL_INIT_AUDIO | SDL_INIT_TIMER)) {
        printf( "Could not initialize SDL - %s\n", SDL_GetError());
        return -1;
    }
    //SDL 2.0 Support for multiple windows
    screen_w = pCodecCtx->width;
    screen_h = pCodecCtx->height;
    screen = SDL_CreateWindow("Simplest ffmpeg player's Window", SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED,
        screen_w, screen_h,SDL_WINDOW_OPENGL);

    if(!screen) {
        printf("SDL: could not create window - exiting:%s\n",SDL_GetError());
        return -1;
    }
    sdlRenderer = SDL_CreateRenderer(screen, -1, 0);
    //IYUV: Y + U + V  (3 planes)
    //YV12: Y + V + U  (3 planes)
    sdlTexture = SDL_CreateTexture(sdlRenderer, SDL_PIXELFORMAT_IYUV, SDL_TEXTUREACCESS_STREAMING,pCodecCtx->width,pCodecCtx->height
);

    sdlRect.x=0;
    sdlRect.y=0;
    sdlRect.w=screen_w;
    sdlRect.h=screen_h;

    packet=(AVPacket *)av_malloc(sizeof(AVPacket));

    video_tid = SDL_CreateThread(sfp_refresh_thread,NULL,NULL);
    //------------SDL End------------
    //Event Loop

    for (;;) {
```

```
        //Wait
        SDL_WaitEvent(&event);
        if(event.type==SFM_REFRESH_EVENT){
            while(1){
                if(av_read_frame(pFormatCtx, packet)<0)
                    thread_exit=1;

                if(packet->stream_index==videoindex)
                    break;
            }
            ret = avcodec_decode_video2(pCodecCtx, pFrame, &got_picture, packet);
            if(ret < 0){
                printf("Decode Error.\n");
                return -1;
            }
            if(got_picture){
                sws_scale(img_convert_ctx, (const unsigned char* const*)pFrame->data, pFrame->linesize, 0, pCodecCtx->height, pFrameYUV->data, pFrameYUV->linesize);
                //SDL---------------------------
                SDL_UpdateTexture( sdlTexture, NULL, pFrameYUV->data[0], pFrameYUV->linesize[0] );
                SDL_RenderClear( sdlRenderer );
                //SDL_RenderCopy( sdlRenderer, sdlTexture, &sdlRect, &sdlRect );
                SDL_RenderCopy( sdlRenderer, sdlTexture, NULL, NULL);
                SDL_RenderPresent( sdlRenderer );
                //SDL End-----------------------
            }
            av_free_packet(packet);
        }else if(event.type==SDL_KEYDOWN){
            //Pause
            if(event.key.keysym.sym==SDLK_SPACE)
                thread_pause=!thread_pause;
        }else if(event.type==SDL_QUIT){
            thread_exit=1;
        }else if(event.type==SFM_BREAK_EVENT){
            break;
        }

    }

    sws_freeContext(img_convert_ctx);

    SDL_Quit();
    //--------------
    av_frame_free(&pFrameYUV);
    av_frame_free(&pFrame);
    avcodec_close(pCodecCtx);
    avformat_close_input(&pFormatCtx);

    return 0;
}
```

运行结果

程序运行后，会在命令行窗口打印一些视频信息，同时会弹出一个窗口播放视频内容。



下载

CSDN完整工程下载地址：

http://download.csdn.net/detail/leixiaohua1020/7826277

更新（2014.10.5）==========

版本升级至2.2。

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
//FIX: Flush Frames remained in Codec
while (1) {
    ret = avcodec_decode_video2(pCodecCtx, pFrame, &got_picture, packet);
    if (ret < 0)
        break;
    if (!got_picture)
        break;
    sws_scale(img_convert_ctx, (const uint8_t* const*)pFrame->data, pFrame->linesize, 0, pCodecCtx->height, pFrameYUV->data, pFrameYUV->linesize);
    //处理...
}
```

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

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

2.2版下载地址： http://download.csdn.net/detail/leixiaohua1020/8002337

更新-2.3（2015.1.03）==========

新增一个工程： 最简单的基于FFmpeg的解码器-纯净版（不包含libavformat）

2.3版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
::VS2010 Environment
call "D:\Program Files\Microsoft Visual Studio 10.0\VC\vcvarsall.bat"
::include
@set INCLUDE=include;%INCLUDE%
::lib
@set LIB=lib;%LIB%
::compile and link
cl simplest_ffmpeg_player.cpp /MD /link SDL2.lib SDL2main.lib avcodec.lib ^
avformat.lib avutil.lib avdevice.lib avfilter.lib postproc.lib swresample.lib swscale.lib ^
/SUBSYSTEM:WINDOWS /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
g++ simplest_ffmpeg_player.cpp -g -o simplest_ffmpeg_player.exe \
-I /usr/local/include -L /usr/local/lib \
-lmingw32 -lSDL2main -lSDL2 -lavformat -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
gcc simplest_ffmpeg_player.cpp -g -o simplest_ffmpeg_player.out \
-I /usr/local/include -L /usr/local/lib -lSDL2main -lSDL2 -lavformat -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 视频 解码 播放器

个人分类： FFMPEG 我的开源项目

所属专栏： FFmpeg