最简单的视音频播放示例7：SDL2播放RGB/YUV

2014年10月29日 00:18:23　阅读数：17129

==========

最简单的视音频播放示例系列文章列表：

最简单的视音频播放示例1：总述

最简单的视音频播放示例2：GDI播放YUV, RGB

最简单的视音频播放示例3：Direct3D播放YUV，RGB（通过Surface）

最简单的视音频播放示例4：Direct3D播放RGB（通过Texture）

最简单的视音频播放示例5：OpenGL播放RGB/YUV

最简单的视音频播放示例6：OpenGL播放YUV420P（通过Texture，使用Shader）

最简单的视音频播放示例7：SDL2播放RGB/YUV

最简单的视音频播放示例8：DirectSound播放PCM

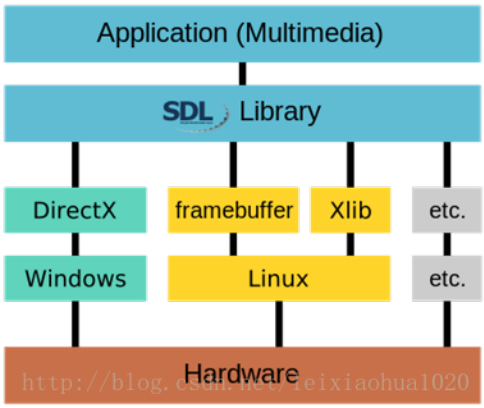
最简单的视音频播放示例9：SDL2播放PCM

==========

本文记录SDL播放视频的技术。在这里使用的版本是SDL2。实际上SDL本身并不提供视音频播放的功能，它只是封装了视音频播放的底层API。在Windows平台下，SDL封装了Direct3D这类的API用于播放视频；封装了DirectSound这类的API用于播放音频。因为SDL的编写目的就是简化视音频播放的开发难度，所以使用SDL播放视频（YUV/RGB）和音频（PCM）数据非常的容易。下文记录一下使用SDL播放视频数据的技术。

SDL简介

SDL（Simple DirectMedia Layer）是一套开放源代码的跨平台多媒体开发库，使用C语言写成。SDL提供了数种控制图像、声音、输出入的函数，让开发者只要用相同或是相似的代码就可以开发出跨多个平台（Linux、Windows、Mac OS X等）的应用软件。目前SDL多用于开发游戏、模拟器、媒体播放器等多媒体应用领域。用下面这张图可以很明确地说明SDL的位置。



SDL实际上并不限于视音频的播放，它将功能分成下列数个子系统（subsystem）：

Video（图像）：图像控制以及线程（thread）和事件管理（event）。

Audio（声音）：声音控制

Joystick（摇杆）：游戏摇杆控制

CD-ROM（光盘驱动器）：光盘媒体控制

Window Management（视窗管理）：与视窗程序设计集成

Event（事件驱动）：处理事件驱动

在Windows下，SDL与DirectX的对应关系如下。

SDL	DirectX
SDL_Video、SDL_Image	DirectDraw、Direct3D
SDL_Audio、SDL_Mixer	DirectSound
SDL_Joystick、SDL_Base	DirectInput
SDL_Net	DirectPlay

SDL播放视频的流程

SDL播放视频的技术在此前做的FFmpeg的示例程序中已经多次用到。在这里重新总结一下流程。

1. 初始化
   1) 初始化SDL
   2) 创建窗口（Window）
   3) 基于窗口创建渲染器（Render）
   4) 创建纹理（Texture）
2. 循环显示画面
   1) 设置纹理的数据
   2) 纹理复制给渲染目标
   3) 显示

下面详细分析一下上文的流程。

1. 初始化

1) 初始化SDL

使用SDL_Init()初始化SDL。该函数可以确定希望激活的子系统。SDL_Init()函数原型如下：

```cpp
int SDLCALL SDL_Init(Uint32 flags)
```

其中，flags可以取下列值：

SDL_INIT_TIMER：定时器
SDL_INIT_AUDIO：音频
SDL_INIT_VIDEO：视频
SDL_INIT_JOYSTICK：摇杆
SDL_INIT_HAPTIC：触摸屏
SDL_INIT_GAMECONTROLLER：游戏控制器
SDL_INIT_EVENTS：事件
SDL_INIT_NOPARACHUTE：不捕获关键信号（这个没研究过）
SDL_INIT_EVERYTHING：包含上述所有选项

有关SDL_Init()有一点需要注意：初始化的时候尽量做到“够用就好”，而不要用SDL_INIT_EVERYTHING。因为有些情况下使用SDL_INIT_EVERYTHING会出现一些不可预知的问题。例如，在MFC应用程序中播放纯音频，如果初始化SDL的时候使用SDL_INIT_EVERYTHING，那么就会出现听不到声音的情况。后来发现，去掉了SDL_INIT_VIDEO之后，问题才得以解决。

2) 创建窗口（Window）

使用SDL_CreateWindow()创建一个用于视频播放的窗口。SDL_CreateWindow()的原型如下。

```cpp
SDL_Window * SDLCALL SDL_CreateWindow(const char *title,
                                      int x, int y, int w,
                                      int h, Uint32 flags);
```

参数含义如下。

title ：窗口标题
x ：窗口位置x坐标。也可以设置为SDL_WINDOWPOS_CENTERED或SDL_WINDOWPOS_UNDEFINED。
y ：窗口位置y坐标。同上。

w　：窗口的宽
h　：窗口的高
flags：支持下列标识。包括了窗口的是否最大化、最小化，能否调整边界等等属性。
::SDL_WINDOW_FULLSCREEN,　::SDL_WINDOW_OPENGL,
::SDL_WINDOW_HIDDEN,　::SDL_WINDOW_BORDERLESS,
::SDL_WINDOW_RESIZABLE,　::SDL_WINDOW_MAXIMIZED,
::SDL_WINDOW_MINIMIZED,　::SDL_WINDOW_INPUT_GRABBED,
::SDL_WINDOW_ALLOW_HIGHDPI.
返回创建完成的窗口的ID。如果创建失败则返回0。

3)　基于窗口创建渲染器（Render）

使用SDL_CreateRenderer()基于窗口创建渲染器。SDL_CreateRenderer()原型如下。

```cpp
SDL_Renderer * SDLCALL SDL_CreateRenderer(SDL_Window * window,
                                          int index, Uint32 flags);
```

参数含义如下。
window　：渲染的目标窗口。
index　：打算初始化的渲染设备的索引。设置“-1”则初始化默认的渲染设备。
flags　：支持以下值（位于SDL_RendererFlags定义中）
SDL_RENDERER_SOFTWARE ：使用软件渲染
SDL_RENDERER_ACCELERATED ：使用硬件加速
SDL_RENDERER_PRESENTVSYNC：和显示器的刷新率同步
SDL_RENDERER_TARGETTEXTURE ：不太懂
返回创建完成的渲染器的ID。如果创建失败则返回NULL。

4)　创建纹理（Texture）

使用SDL_CreateTexture()基于渲染器创建一个纹理。SDL_CreateTexture()的原型如下。

```cpp
SDL_Texture * SDLCALL SDL_CreateTexture(SDL_Renderer * renderer,
                                        Uint32 format,
                                        int access, int w,
                                        int h);
```

参数的含义如下。
renderer：目标渲染器。
format　：纹理的格式。后面会详述。
access　：可以取以下值（定义位于SDL_TextureAccess中）
SDL_TEXTUREACCESS_STATIC　：变化极少
SDL_TEXTUREACCESS_STREAMING　：变化频繁
SDL_TEXTUREACCESS_TARGET　：暂时没有理解
w　：纹理的宽
h　：纹理的高
创建成功则返回纹理的ID，失败返回0。

在纹理的创建过程中，需要指定纹理的格式（即第二个参数）。SDL的中的格式很多，如下所列。

```cpp
SDL_PIXELFORMAT_UNKNOWN,
SDL_PIXELFORMAT_INDEX1LSB =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_INDEX1, SDL_BITMAPORDER_4321, 0,
                           1, 0),
SDL_PIXELFORMAT_INDEX1MSB =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_INDEX1, SDL_BITMAPORDER_1234, 0,
                           1, 0),
SDL_PIXELFORMAT_INDEX4LSB =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_INDEX4, SDL_BITMAPORDER_4321, 0,
                           4, 0),
SDL_PIXELFORMAT_INDEX4MSB =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_INDEX4, SDL_BITMAPORDER_1234, 0,
                           4, 0),
SDL_PIXELFORMAT_INDEX8 =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_INDEX8, 0, 0, 8, 1),
SDL_PIXELFORMAT_RGB332 =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_PACKED8, SDL_PACKEDORDER_XRGB,
                           SDL_PACKEDLAYOUT_332, 8, 1),
SDL_PIXELFORMAT_RGB444 =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_PACKED16, SDL_PACKEDORDER_XRGB,
                           SDL_PACKEDLAYOUT_4444, 12, 2),
SDL_PIXELFORMAT_RGB555 =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_PACKED16, SDL_PACKEDORDER_XRGB,
                           SDL_PACKEDLAYOUT_1555, 15, 2),
SDL_PIXELFORMAT_BGR555 =
```

```
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_PACKED16, SDL_PACKEDORDER_XBGR,
                           SDL_PACKEDLAYOUT_1555, 15, 2),
SDL_PIXELFORMAT_ARGB4444 =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_PACKED16, SDL_PACKEDORDER_ARGB,
                           SDL_PACKEDLAYOUT_4444, 16, 2),
SDL_PIXELFORMAT_RGBA4444 =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_PACKED16, SDL_PACKEDORDER_RGBA,
                           SDL_PACKEDLAYOUT_4444, 16, 2),
SDL_PIXELFORMAT_ABGR4444 =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_PACKED16, SDL_PACKEDORDER_ABGR,
                           SDL_PACKEDLAYOUT_4444, 16, 2),
SDL_PIXELFORMAT_BGRA4444 =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_PACKED16, SDL_PACKEDORDER_BGRA,
                           SDL_PACKEDLAYOUT_4444, 16, 2),
SDL_PIXELFORMAT_ARGB1555 =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_PACKED16, SDL_PACKEDORDER_ARGB,
                           SDL_PACKEDLAYOUT_1555, 16, 2),
SDL_PIXELFORMAT_RGBA5551 =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_PACKED16, SDL_PACKEDORDER_RGBA,
                           SDL_PACKEDLAYOUT_5551, 16, 2),
SDL_PIXELFORMAT_ABGR1555 =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_PACKED16, SDL_PACKEDORDER_ABGR,
                           SDL_PACKEDLAYOUT_1555, 16, 2),
SDL_PIXELFORMAT_BGRA5551 =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_PACKED16, SDL_PACKEDORDER_BGRA,
                           SDL_PACKEDLAYOUT_5551, 16, 2),
SDL_PIXELFORMAT_RGB565 =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_PACKED16, SDL_PACKEDORDER_XRGB,
                           SDL_PACKEDLAYOUT_565, 16, 2),
SDL_PIXELFORMAT_BGR565 =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_PACKED16, SDL_PACKEDORDER_XBGR,
                           SDL_PACKEDLAYOUT_565, 16, 2),
SDL_PIXELFORMAT_RGB24 =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_ARRAYU8, SDL_ARRAYORDER_RGB, 0,
                           24, 3),
SDL_PIXELFORMAT_BGR24 =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_ARRAYU8, SDL_ARRAYORDER_BGR, 0,
                           24, 3),
SDL_PIXELFORMAT_RGB888 =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_PACKED32, SDL_PACKEDORDER_XRGB,
                           SDL_PACKEDLAYOUT_8888, 24, 4),
SDL_PIXELFORMAT_RGBX8888 =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_PACKED32, SDL_PACKEDORDER_RGBX,
                           SDL_PACKEDLAYOUT_8888, 24, 4),
SDL_PIXELFORMAT_BGR888 =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_PACKED32, SDL_PACKEDORDER_XBGR,
                           SDL_PACKEDLAYOUT_8888, 24, 4),
SDL_PIXELFORMAT_BGRX8888 =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_PACKED32, SDL_PACKEDORDER_BGRX,
                           SDL_PACKEDLAYOUT_8888, 24, 4),
SDL_PIXELFORMAT_ARGB8888 =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_PACKED32, SDL_PACKEDORDER_ARGB,
                           SDL_PACKEDLAYOUT_8888, 32, 4),
SDL_PIXELFORMAT_RGBA8888 =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_PACKED32, SDL_PACKEDORDER_RGBA,
                           SDL_PACKEDLAYOUT_8888, 32, 4),
SDL_PIXELFORMAT_ABGR8888 =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_PACKED32, SDL_PACKEDORDER_ABGR,
                           SDL_PACKEDLAYOUT_8888, 32, 4),
SDL_PIXELFORMAT_BGRA8888 =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_PACKED32, SDL_PACKEDORDER_BGRA,
                           SDL_PACKEDLAYOUT_8888, 32, 4),
SDL_PIXELFORMAT_ARGB2101010 =
    SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_PACKED32, SDL_PACKEDORDER_ARGB,
                           SDL_PACKEDLAYOUT_2101010, 32, 4),

SDL_PIXELFORMAT_YV12 =      /**< Planar mode: Y + V + U  (3 planes) */
    SDL_DEFINE_PIXELFOURCC('Y', 'V', '1', '2'),
SDL_PIXELFORMAT_IYUV =      /**< Planar mode: Y + U + V  (3 planes) */
    SDL_DEFINE_PIXELFOURCC('I', 'Y', 'U', 'V'),
SDL_PIXELFORMAT_YUY2 =      /**< Packed mode: Y0+U0+Y1+V0 (1 plane) */
    SDL_DEFINE_PIXELFOURCC('Y', 'U', 'Y', '2'),
SDL_PIXELFORMAT_UYVY =      /**< Packed mode: U0+Y0+V0+Y1 (1 plane) */
    SDL_DEFINE_PIXELFOURCC('U', 'Y', 'V', 'Y'),
SDL_PIXELFORMAT_YVYU =      /**< Packed mode: Y0+V0+Y1+U0 (1 plane) */
    SDL_DEFINE_PIXELFOURCC('Y', 'V', 'Y', 'U')
```

这一看确实给人一种"眼花缭乱"的感觉。简单分析一下其中的定义吧。例如ARGB8888的定义如下。

```cpp
SDL_PIXELFORMAT_ARGB8888 =
        SDL_DEFINE_PIXELFORMAT(SDL_PIXELTYPE_PACKED32, SDL_PACKEDORDER_ARGB,
                               SDL_PACKEDLAYOUT_8888, 32, 4),
```

其中用了一个宏SDL_DEFINE_PIXELFORMAT用于将几种属性合并到一个格式中。下面我们看看一个格式都包含哪些属性：
SDL_PIXELTYPE_PACKED32 ：代表了像素分量的存储方式。PACKED代表了像素的几个分量是一起存储的，内存中存储方式如下：R1|G1|B1，R2|G2|B2…；ARRAY则代表了像素的几个分量是分开存储的，内存中存储方式如下：R1|R2|R3…，G1|G2|G3…，B1|B2|B3…

SDL_PACKEDORDER_ARGB ：代表了PACKED存储方式下像素分量的顺序。注意，这里所说的顺序涉及到了一个“大端”和“小端”的问题。这个问题在《 最简单的视音频播放示例2：GDI播放YUV, RGB 》中已经叙述，不再重复记录。对于Windows这样的“小端”系统，“ARGB”格式在内存中的存储顺序是B|G|R|A。
SDL_PACKEDLAYOUT_8888 ：说明了每个分量占据的比特数。例如ARGB格式每个分量分别占据了8bit。
32 ：每个像素占用的比特数。例如ARGB格式占用了32bit（每个分量占据8bit）。
4 ：每个像素占用的字节数。例如ARGB格式占用了4Byte（每个分量占据1Byte）。

2. 循环显示画面

1) 设置纹理的数据

使用SDL_UpdateTexture()设置纹理的像素数据。SDL_UpdateTexture()的原型如下。

```cpp
int SDLCALL SDL_UpdateTexture(SDL_Texture * texture,
                              const SDL_Rect * rect,
                              const void *pixels, int pitch);
```

参数的含义如下。
texture：目标纹理。
rect：更新像素的矩形区域。设置为NULL的时候更新整个区域。
pixels：像素数据。
pitch：一行像素数据的字节数。
成功的话返回0，失败的话返回-1。

2) 纹理复制给渲染目标
使用SDL_RenderCopy()将纹理数据复制给渲染目标。在使用SDL_RenderCopy()之前，可以使用SDL_RenderClear()先使用清空渲染目标。实际上视频播放的时候不使用SDL_RenderClear()也是可以的，因为视频的后一帧会完全覆盖前一帧。
SDL_RenderClear()原型如下。

```cpp
int SDLCALL SDL_RenderClear(SDL_Renderer * renderer);
```

参数renderer用于指定渲染目标。
SDL_RenderCopy()原型如下。

```cpp
int SDLCALL SDL_RenderCopy(SDL_Renderer * renderer,
                           SDL_Texture * texture,
                           const SDL_Rect * srcrect,
                           const SDL_Rect * dstrect);
```

参数的含义如下。
renderer：渲染目标。
texture：输入纹理。
srcrect：选择输入纹理的一块矩形区域作为输入。设置为NULL的时候整个纹理作为输入。
dstrect：选择渲染目标的一块矩形区域作为输出。设置为NULL的时候整个渲染目标作为输出。
成功的话返回0，失败的话返回-1。

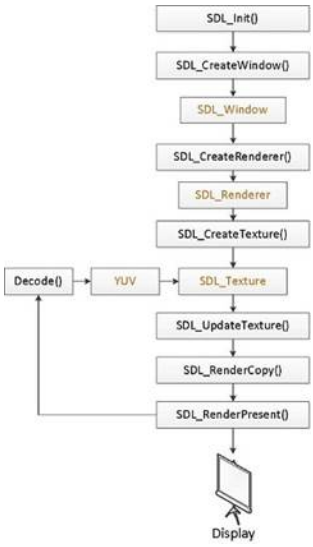
3) 显示
使用SDL_RenderPresent()显示画面。SDL_RenderPresent()原型如下。

```cpp
void SDLCALL SDL_RenderPresent(SDL_Renderer * renderer);
```

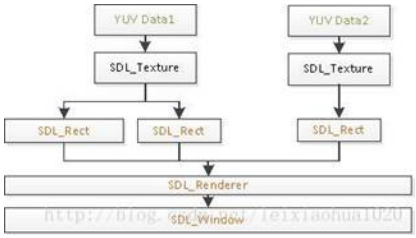
参数renderer用于指定渲染目标。

流程总结

在《最简单的基于FFMPEG+SDL的视频播放器 ver2（采用SDL2.0）》中总结过SDL2播放视频的流程，在这里简单复制过来。
使用SDL播放视频的流程可以概括为下图。



SDL中几个关键的结构体之间的关系可以用下图概述。



简单解释一下各变量的作用：
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。

代码

贴出源代码。

```cpp
/**
 * 最简单的SDL2播放视频的例子（SDL2播放RGB/YUV）
 * Simplest Video Play SDL2 (SDL2 play RGB/YUV)
 *
 * 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
 *
 * 本程序使用SDL2播放RGB/YUV视频像素数据。SDL实际上是对底层绘图
 * API（Direct3D，OpenGL）的封装，使用起来明显简单于直接调用底层
 * API。
 *
 * 函数调用步骤如下：
 *
 * [初始化]
 * SDL_Init(): 初始化SDL。
 * SDL_CreateWindow(): 创建窗口（Window）。
 * SDL_CreateRenderer(): 基于窗口创建渲染器（Render）。
 * SDL_CreateTexture(): 创建纹理（Texture）。
 *
 * [循环渲染数据]
 * SDL_UpdateTexture(): 设置纹理的数据。
 * SDL_RenderCopy(): 纹理复制给渲染器。
 * SDL_RenderPresent(): 显示。
 *
 * This software plays RGB/YUV raw video data using SDL2.
 * SDL is a wrapper of low-level API (Direct3D, OpenGL).
 * Use SDL is much easier than directly call these low-level API.
 *
 * The process is shown as follows:
 *
 * [Init]
 * SDL_Init(): Init SDL.
 * SDL_CreateWindow(): Create a Window.
```

```
 * SDL_CreateWindow(): Create a Window.
 * SDL_CreateRenderer(): Create a Render.
 * SDL_CreateTexture(): Create a Texture.
 *
 * [Loop to Render data]
 * SDL_UpdateTexture(): Set Texture's data.
 * SDL_RenderCopy(): Copy Texture to Render.
 * SDL_RenderPresent(): Show.
 */

#include <stdio.h>

extern "C"
{
#include "sdl/SDL.h"
};

//set '1' to choose a type of file to play
#define LOAD_BGRA    1
#define LOAD_RGB24   0
#define LOAD_BGR24   0
#define LOAD_YUV420P 0

//Bit per Pixel
#if LOAD_BGRA
const int bpp=32;
#elif LOAD_RGB24|LOAD_BGR24
const int bpp=24;
#elif LOAD_YUV420P
const int bpp=12;
#endif

int screen_w=500,screen_h=500;
const int pixel_w=320,pixel_h=180;

unsigned char buffer[pixel_w*pixel_h*bpp/8];
//BPP=32
unsigned char buffer_convert[pixel_w*pixel_h*4];

//Convert RGB24/BGR24 to RGB32/BGR32
//And change Endian if needed
void CONVERT_24to32(unsigned char *image_in,unsigned char *image_out,int w,int h){
    for(int i =0;i<h;i++)
        for(int j=0;j<w;j++){
            //Big Endian or Small Endian?
            //"ARGB" order:high bit -> low bit.
            //ARGB Format Big Endian (low address save high MSB, here is A) in memory : A|R|G|B
            //ARGB Format Little Endian (low address save low MSB, here is B) in memory : B|G|R|A
            if(SDL_BYTEORDER==SDL_LIL_ENDIAN){
                //Little Endian (x86): R|G|B --> B|G|R|A
                image_out[(i*w+j)*4+0]=image_in[(i*w+j)*3+2];
                image_out[(i*w+j)*4+1]=image_in[(i*w+j)*3+1];
                image_out[(i*w+j)*4+2]=image_in[(i*w+j)*3];
                image_out[(i*w+j)*4+3]='0';
            }else{
                //Big Endian: R|G|B --> A|R|G|B
                image_out[(i*w+j)*4]='0';
                memcpy(image_out+(i*w+j)*4+1,image_in+(i*w+j)*3,3);
            }
        }
}


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

int thread_exit=0;

int refresh_video(void *opaque){
    while (thread_exit==0) {
        SDL_Event event;
        event.type = REFRESH_EVENT;
        SDL_PushEvent(&event);
        SDL_Delay(40);
    }
    return 0;
}

int main(int argc, char* argv[])
{
    if(SDL_Init(SDL_INIT_VIDEO)) {
        printf( "Could not initialize SDL - %s\n", SDL_GetError());
        return -1;
    }

    SDL_Window *screen;
    //SDL 2.0 Support for multiple windows
    screen = SDL_CreateWindow("Simplest Video Play SDL2", SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED,
        screen_w, screen_h,SDL_WINDOW_OPENGL|SDL_WINDOW_RESIZABLE);
    if(!screen) {
        printf("SDL: could not create window - exiting:%s\n",SDL_GetError());
        return -1;
```

```
    }
    SDL_Renderer* sdlRenderer = SDL_CreateRenderer(screen, -1, 0);

    Uint32 pixformat=0;
#if LOAD_BGRA
    //Note: ARGB8888 in "Little Endian" system stores as B|G|R|A
    pixformat= SDL_PIXELFORMAT_ARGB8888;
#elif LOAD_RGB24
    pixformat= SDL_PIXELFORMAT_RGB888;
#elif LOAD_BGR24
    pixformat= SDL_PIXELFORMAT_BGR888;
#elif LOAD_YUV420P
    //IYUV: Y + U + V  (3 planes)
    //YV12: Y + V + U  (3 planes)
    pixformat= SDL_PIXELFORMAT_IYUV;
#endif

    SDL_Texture* sdlTexture = SDL_CreateTexture(sdlRenderer,pixformat, SDL_TEXTUREACCESS_STREAMING,pixel_w,pixel_h);



    FILE *fp=NULL;
#if LOAD_BGRA
    fp=fopen("../test_bgra_320x180.rgb","rb+");
#elif LOAD_RGB24
    fp=fopen("../test_rgb24_320x180.rgb","rb+");
#elif LOAD_BGR24
    fp=fopen("../test_bgr24_320x180.rgb","rb+");
#elif LOAD_YUV420P
    fp=fopen("../test_yuv420p_320x180.yuv","rb+");
#endif
    if(fp==NULL){
        printf("cannot open this file\n");
        return -1;
    }

    SDL_Rect sdlRect;

    SDL_Thread *refresh_thread = SDL_CreateThread(refresh_video,NULL,NULL);
    SDL_Event event;
    while(1){
        //Wait
        SDL_WaitEvent(&event);
        if(event.type==REFRESH_EVENT){
            if (fread(buffer, 1, pixel_w*pixel_h*bpp/8, fp) != pixel_w*pixel_h*bpp/8){
                // Loop
                fseek(fp, 0, SEEK_SET);
                fread(buffer, 1, pixel_w*pixel_h*bpp/8, fp);
            }

#if LOAD_BGRA
            //We don't need to change Endian
            //Because input BGRA pixel data(B|G|R|A) is same as ARGB8888 in Little Endian (B|G|R|A)
            SDL_UpdateTexture( sdlTexture, NULL, buffer, pixel_w*4);
#elif LOAD_RGB24|LOAD_BGR24
            //change 24bit to 32 bit
            //and in Windows we need to change Endian
            CONVERT_24to32(buffer,buffer_convert,pixel_w,pixel_h);
            SDL_UpdateTexture( sdlTexture, NULL, buffer_convert, pixel_w*4);
#elif LOAD_YUV420P
            SDL_UpdateTexture( sdlTexture, NULL, buffer, pixel_w);
#endif
            //FIX: If window is resize
            sdlRect.x = 0;
            sdlRect.y = 0;
            sdlRect.w = screen_w;
            sdlRect.h = screen_h;

            SDL_RenderClear( sdlRenderer );
            SDL_RenderCopy( sdlRenderer, sdlTexture, NULL, &sdlRect);
            SDL_RenderPresent( sdlRenderer );
            //Delay 40ms
            SDL_Delay(40);

        }else if(event.type==SDL_WINDOWEVENT){
            //If Resize
            SDL_GetWindowSize(screen,&screen_w,&screen_h);
        }else if(event.type==SDL_QUIT){
            break;
        }
    }

    return 0;
}
```

运行结果

程序的运行结果如下图所示。

下载

代码位于“Simplest Media Play”中

SourceForge项目地址： https://sourceforge.net/projects/simplestmediaplay/

CSDN下载地址： http://download.csdn.net/detail/leixiaohua1020/8054395

注：

该项目会不定时的更新并修复一些小问题，最新的版本请参考该系列文章的总述页面：

《最简单的视音频播放示例1：总述》

上述工程包含了使用各种API（Direct3D，OpenGL，GDI，DirectSound，SDL2）播放多媒体例子。其中音频输入为PCM采样数据。输出至系统的声卡播放出来。视频输入为YUV/RGB像素数据。输出至显示器上的一个窗口播放出来。
通过本工程的代码初学者可以快速学习使用这几个API播放视频和音频的技术。
一共包括了如下几个子工程：
simplest_audio_play_directsound: 使用DirectSound播放PCM音频采样数据。
simplest_audio_play_sdl2: 使用SDL2播放PCM音频采样数据。
simplest_video_play_direct3d: 使用Direct3D的Surface播放RGB/YUV视频像素数据。
simplest_video_play_direct3d_texture:使用Direct3D的Texture播放RGB视频像素数据。
simplest_video_play_gdi: 使用GDI播放RGB/YUV视频像素数据。
simplest_video_play_opengl: 使用OpenGL播放RGB/YUV视频像素数据。
simplest_video_play_opengl_texture: 使用OpenGL的Texture播放YUV视频像素数据。
simplest_video_play_sdl2: 使用SDL2播放RGB/YUV视频像素数据。

文章标签： SDL YUV RGB 视频 显示

个人分类： SDL 我的开源项目