

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

2014年10月21日 00:14:27 阅读数：24382

=====

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](#)

=====

上一篇文章记录了GDI播放视频的技术。打算接下来写两篇文章记录Direct3D（简称D3D）播放视频的技术。Direct3D应该Windows下最常用的播放视频的技术。实际上视频播放只是Direct3D的“副业”，它主要用于3D游戏制作。当前主流的游戏几乎都是使用Direct3D制作的，例如《地下城与勇士》，《穿越火线》，《英雄联盟》，《魔兽世界》，《QQ飞车》等等。使用Direct3D可以用两种方式渲染视频：Surface和Texture。使用Surface相对来说比使用Texture要简单一些，但是不如使用Texture灵活。鉴于使用Surface更加容易上手，本文记录使用Direct3D中的Surface显示视频的技术。下一篇文章再记录使用Direct3D中的Texture显示视频的技术。



Direct3D简介

下面简单记录一下背景知识。摘录修改了维基上的一部分内容（维基上这部分叙述貌似很不准确...）：

Direct3D（简称：D3D）是微软公司在Microsoft Windows系统上开发的一套3D绘图API，是DirectX的一部份，目前广为各家显示卡所支援。1995年2月，微软收购了英国的Rendermorphics公司，将RealityLab 2.0技术发展成Direct3D标准，并整合到Microsoft Windows中，Direct3D在DirectX 3.0开始出现。后来在DirectX 8.0发表时与DirectDraw编程介面合并并改名为DirectX Graphics。Direct3D与Windows GDI是同层级组件。它可以直接调用底层显卡的功能。与OpenGL同为电脑绘图软件和电脑游戏最常使用的两套绘图API。

抽象概念

Direct3D的抽象概念包括：Devices（设备），Swap Chains（交换链）和Resources（资源）。

Device（设备）用于渲染3D场景。例如单色设备就会渲染黑白图片，而彩色设备则会渲染彩色图片。Device目前我自己了解的有以下2类（还有其他类型，但不是很熟）：

HAL（Hardware Abstraction Layer）：支持硬件加速的设备。在所有设备中运行速度是最快的，也是最常用的。

Reference：模拟一些硬件还不支持的新功能。换言之，就是利用软件，在CPU对硬件渲染设备的一个模拟。

每一个Device至少要有有一个Swap Chain（交换链）。一个Swap Chain由一个或多个Back Buffer Surfaces（后台缓冲表面）组成。渲染在Back Buffer中完成。

此外，Device包含了一系列的Resources（资源），用于定义渲染时候的数据。每个Resources有4个属性：

Type：描述Resource的类型。例如surface，volume，texture，cube texture，volume texture，surface texture，index buffer 或者vertex buffer。

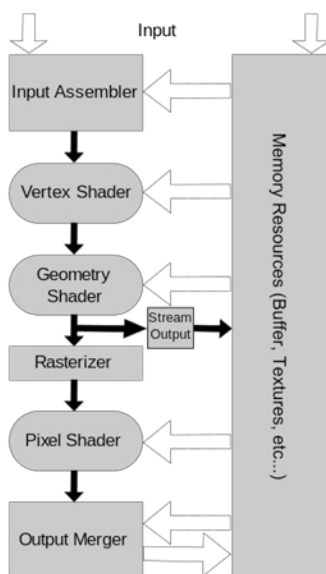
Usage：描述Resource如何被使用。例如指定Resource是以只读方式调用还是以可读写的方式调用。

Format：数据的格式。比如一个二维表面的像素格式。例如，D3DFMT_R8G8B8的Format表明了数据格式是24 bits颜色深度的RGB数据。

Pool：描述Resource如何被管理和存储。默认的情况下Resource会被存储在设备的内存（例如显卡的显存）中。也可以指定Resource存储在系统内存中。

渲染流水线（rendering pipeline）

Direct3D API定义了一组Vertices(顶点)，Textures(纹理)，Buffers(缓冲区) 转换到屏幕上的流程。这样的流程称为Rendering Pipeline（渲染流水线），它的各阶段包括：



Input Assembler（输入组装）：从应用程序里读取vertex数据，将其装进流水线。

Vertex Shader（顶点着色器）：对每个顶点属性进行着色。每次处理一个顶点，比如变换、贴图、光照等。注意这个地方可能需要自己编程。

Geometry Shader（几何着色器）：Shader Model 4.0引进了几何着色器，处理点、线、面的几何坐标变换。此处我自己还不是很了解。

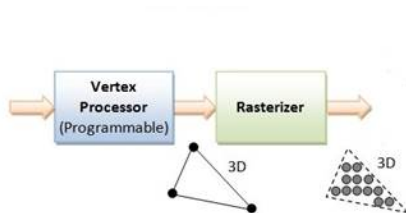
PS：上述处理完后的数据可以理解为以下图片。即包含顶点信息，但不包含颜色信息。



Stream Output（流输出）：将Vertex Shader和Geometry Shader处理完成的数据输出给使用者。

Rasterizer（光栅化）：把算完的顶点转成像素，再将像素（pixels）输出给Pixel Shader。这里也可执行其他工作，比如像素数据的切割，插值等。

PS：光栅化的过程可以理解为下图。即把顶点转换成像素。



Pixel Shader（像素着色器）：对每个像素进行着色。注意这个地方可能需要自己编程。

Output Merger（输出混合）：整合各种不同的数据，输出最后结果。

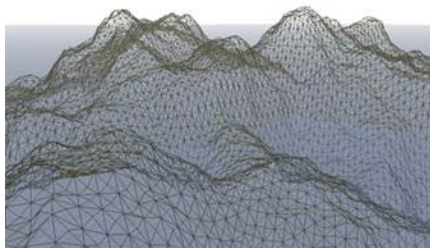
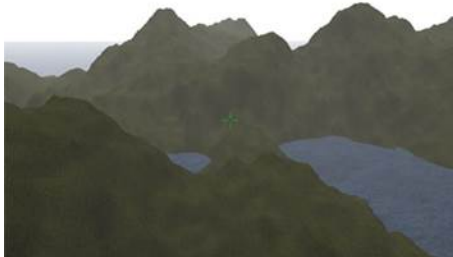
视频显示的基础知识

在记录Direct3D的视频显示技术之前，首先记录一下视频显示的基础知识。我自己归纳总结了以下几点知识。

1.

三角形

在Direct3D中经常会出现“三角形”这个概念。这是因为在3D图形渲染中，所有的物体都是由三角形构成的。因为一个三角形可以表示一个平面，而3D物体就是由一个或多个平面构成的。比如下图表示了一个非常复杂的3D地形，它们也不过是由许许多多三角形表示的。



因此我们只要指定一个或多个三角形，就可以表示任意3D物体。

2.

后台缓冲表面，前台表面，交换链，离屏表面

后台缓冲表面和前台表面的概念总是同时出现的。简单解释一下它们的作用。当我们进行复杂的绘图操作时，画面可能有明显的闪烁。这是由于绘制的东西没有同时出现在屏幕上而导致的。“前台表面”+“后台缓冲表面”的技术可以解决这个问题。前台表面即我们看到的屏幕，后台缓冲表面则在内存当中，对我们来说是不可见的。每次的所有绘图操作不是在屏幕上直接绘制，而是在后台缓冲表面中进行，当绘制完成后，需要的时候再把绘制的最终结果显示到屏幕上。这样就解决了上述的问题。

实际上，上述技术还涉及到一个“交换链”（Swap Chain）的概念。所谓的“链”，指的是一系列的表面组成的一个集合。这些表面中有一个是前台表面（显示在屏幕上），剩下的都是后台缓冲表面。其实，简单的交换链不需要很多表面，只要两个就可以了（虽然感觉不像“链”）。一个后台缓冲表面，一个前台表面。所谓的“交换”，即是在需要呈现后台缓冲表面中的内容的时候，交换这两个表面的“地位”。即前台表面变成后台缓冲表面，后台缓冲表面变成前台表面。如此一来，后台缓冲表面的内容就呈现在屏幕上了。原先的前台表面，则扮演起了新的后台缓冲表面的角色，准备进行新的绘图操作。当下一次需要显示画面的时候，这两个表面再次交换，如此循环往复，永不停止。

此外，还有一个离屏表面。离屏表面是永远看不到的表面（所谓“离屏”），它通常被用来存放位图，并对其中的数据做一些处理。本文介绍的例子中就用到了一个离屏表面。通常的做法是把离屏表面上的位图复制到后台缓冲表面，后台缓冲表面再显示到前台表面。

安装DirectX SDK

使用Direct3D开发之前需要安装DirectX SDK。安装没有难度，一路“Next”即可。

Microsoft DirectX SDK (June 2010)下载地址：

<http://www.microsoft.com/en-us/download/details.aspx?id=6812>

使用VC进行开发的时候，需要在项目的“属性”对话框中配置头文件和库：

头文件配置：C/C++->常规->附加包含目录

库文件配置：

（a）链接器->常规->附加库目录。

（b）链接器->输入->附加依赖项（填写一个d3d9.lib）

编程的时候，添加头文件后即可使用：

```
[cpp] 1. #include <d3d9.h>
```

D3D视频显示的流程

有关Direct3D的知识的介绍还有很多，在这里就不再记录了。正如那句俗话：“Talk is cheap, show me the code.”，光说理论还是会给人一种没有“脚踏实地”的感觉，下文将会结合代码记录Direct3D中使用Surface渲染视频的技术。

使用Direct3D的Surface播放视频一般情况下需要如下步骤：

1. 创建一个窗口（不属于D3D的API）
2. 初始化
 - 1) 创建一个Device
 - 2) 基于Device创建一个Surface（离屏表面）
- 3.

循环显示画面

- 1)
清理
- 2)
一帧视频数据 拷贝至 Surface
- 3)
开始一个Scene
- 4)
Surface 数据拷贝至 后台缓冲表面
- 5)
结束Scene
- 6)
显示（ 后台缓冲表面 -> 前台表面 ）

下面结合Direct3D播放YUV/RGB的示例代码，详细分析一下上文的流程。

1.

创建一个窗口（不属于D3D的API）

建立一个Win32的窗口程序，就可以用于Direct3D的显示。程序的入口函数是WinMain()，调用CreateWindow()即可创建一个窗口。这一步是必须的，不然Direct3D绘制的内容就没有地方显示了。此处不再详述。

2.

初始化

1)

创建一个Device

这一步完成的时候，可以得到一个IDirect3DDevice9接口的指针。创建一个Device又可以分成以下几个详细的步骤：

(a)

通过 Direct3DCreate9()创建一个IDirect3D9接口。

获取IDirect3D9接口的关键实现代码只有一行：

```
1. IDirect3D9 *m_pDirect3D9 = Direct3DCreate9( D3D_SDK_VERSION );
```

IDirect3D9接口是一个代表我们显示3D图形的物理设备的C++对象。它可以用于获得物理设备的信息和创建一个IDirect3DDevice9接口。例如，可以通过它的GetAdapterDisplayMode()函数获取当前主显卡输出的分辨率，刷新频率等参数，实现代码如下。

```
1. D3DDISPLAYMODE d3dDisplayMode;  
2. HRESULT = m_pDirect3D9->GetAdapterDisplayMode( D3DADAPTER_DEFAULT, &d3dDisplayMode );
```

由代码可以看出，获取的信息存储在D3DDISPLAYMODE结构体中。D3DDISPLAYMODE结构体中包含了主显卡的分辨率等信息：

```
1. /* Display Modes */  
2. typedef struct _D3DDISPLAYMODE  
3. {  
4.     UINT Width;  
5.     UINT Height;  
6.     UINT RefreshRate;  
7.     D3DFORMAT Format;  
8. } D3DDISPLAYMODE;
```

也可以用它的GetDeviceCaps()函数搞清楚主显卡是否支持硬件顶点处理，实现的代码如下。

```
1. D3DCAPS9 d3dcaps;  
2. HRESULT=m_pDirect3D9->GetDeviceCaps(D3DADAPTER_DEFAULT,D3DDEVTYPE_HAL,&d3dcaps);  
3. int hal_vp = 0;  
4. if( d3dcaps.DevCaps & D3DDEVCAPS_HWTRANSFORMANDLIGHT ){  
5.     // yes, save in 'vp' the fact that hardware vertex  
6.     // processing is supported.  
7.     hal_vp = D3DCREATE_HARDWARE_VERTEXPROCESSING;  
8. }
```

由代码可以看出，获取的设备信息存储在D3DCAPS9结构体中。D3DCAPS9定义比较长包含了各种各样的信息，不再列出来。从该结构体的DevCaps字段可以判断得

出该设备是否支持硬件顶点处理。

(b)

设置D3DPRESENT_PARAMETERS结构体，为创建Device做准备。

接下来填充一个D3DPRESENT_PARAMETERS结构的实例。这个结构用于设定我们将要创建的IDirect3DDevice9对象的一些特性，它的定义如下。

```
[cpp]
1. typedef struct _D3DPRESENT_PARAMETERS_
2. {
3.     UINT          BackBufferWidth;
4.     UINT          BackBufferHeight;
5.     D3DFORMAT     BackBufferFormat;
6.     UINT          BackBufferCount;
7.
8.
9.     D3DMULTISAMPLE_TYPE MultiSampleType;
10.    DWORD          MultiSampleQuality;
11.
12.
13.    D3DSWAPEFFECT   SwapEffect;
14.    HWND            hDeviceWindow;
15.    BOOL            Windowed;
16.    BOOL            EnableAutoDepthStencil;
17.    D3DFORMAT       AutoDepthStencilFormat;
18.    DWORD           Flags;
19.
20.
21.    /* FullScreen_RefreshRateInHz must be zero for Windowed mode */
22.    UINT            FullScreen_RefreshRateInHz;
23.    UINT            PresentationInterval;
24. } D3DPRESENT_PARAMETERS;
```

D3DPRESENT_PARAMETERS这个结构体比较重要。详细列一下它每个参数的含义：

BackBufferWidth：后台缓冲表面的宽度（以像素为单位）。

BackBufferHeight：后台缓冲表面的高度（以像素为单位）。

BackBufferFormat：后台缓冲表面的像素格式（例如：32位像素格式为D3DFMT_A8R8G8B8）。

BackBufferCount：后台缓冲表面的数量，通常设为“1”，即只有一个后备表面。

MultiSampleType：全屏抗锯齿的类型，显示视频没用到，不详细分析。

MultiSampleQuality：全屏抗锯齿的质量等级，显示视频没用到，不详细分析。

SwapEffect：指定表面在交换链中是如何被交换的。支持以下取值：

*D3DSWAPEFFECT_DISCARD:后台缓冲表面区的东西被复制到屏幕上后,后台缓冲表面区的东西就没有什么用了,可以丢弃了。

*D3DSWAPEFFECT_FLIP:

后台缓冲表面拷贝到前台表面，保持后台缓冲表面内容不变。当后台缓冲表面大于1个时使用。

*D3DSWAPEFFECT_COPY:

同上。当后台缓冲表面等于1个时使用。

一般使用D3DSWAPEFFECT_DISCARD。

hDeviceWindow：与设备相关的窗口句柄，你想在哪个窗口绘制就写那个窗口的句柄

Windowed：BOOL型，设为true则为窗口模式，false则为全屏模式

EnableAutoDepthStencil：设为true，D3D将自动创建深度/模版缓冲。

AutoDepthStencilFormat：深度/模版缓冲的格式

Flags：一些附加特性

FullScreen_RefreshRateInHz：刷新率，设定D3DPRESENT_RATE_DEFAULT使用默认刷新率

PresentationInterval：设置刷新的间隔，可以用以下方式：

*D3DPRESENT_INTERVAL_DEFAULT，则说明在显示一个渲染画面的时候必要等候显示器刷新完一次屏幕。例如显示器刷新率设为80Hz的话，则一秒最多可以显示80个渲染画面。

*D3DPRESENT_INTERVAL_IMMEDIATE:表示可以以实时的方式来显示渲染画面。

下面列出使用Direct3D播放视频的时候D3DPRESENT_PARAMETERS的一个最简单的设置。

```
[cpp]
1. //D3DPRESENT_PARAMETERS Describes the presentation parameters.
2. D3DPRESENT_PARAMETERS d3dpp;
3. ZeroMemory( &d3dpp, sizeof(d3dpp) );
4. d3dpp.Windowed = TRUE;
5. d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;
6. d3dpp.BackBufferFormat = D3DFMT_UNKNOWN;
```

(c)

通过IDirect3D9的CreateDevice ()创建一个Device。

最后就可以调用IDirect3D9的CreateDevice()方法创建Device了。

CreateDevice()的函数原型如下：

```
[cpp]
1. HRESULT CreateDevice(
2.     UINT Adapter,
3.     D3DDEVTYPE DeviceType,
4.     HWND hFocusWindow,
5.     DWORD BehaviorFlags,
6.     D3DPRESENT_PARAMETERS *pPresentationParameters,
7.     IDirect3DDevice9** ppReturnedDeviceInterface
8. );
```

其中每个参数的含义如下所列：

Adapter：指定对象要表示的物理显示设备。D3DADAPTER_DEFAULT始终是主要的显示器适配器。

DeviceType：设备类型，包括D3DDEVTYPE_HAL(Hardware Accelerator，硬件加速)、D3DDEVTYPE_SW(SoftWare，软件)。

hFocusWindow：同我们在前面d3dpp.hDeviceWindow的相同

BehaviorFlags：设定为D3DCREATE_SOFTWARE_VERTEXPROCESSING（软件顶点处理）或者D3DCREATE_HARDWARE_VERTEXPROCESSING（硬件顶点处理），使用前应该用D3DCAPS9来检测用户计算机是否支持硬件顶点处理功能。

pPresentationParameters：指定一个已经初始化好的D3DPRESENT_PARAMETERS实例

ppReturnedDeviceInterface：返回创建的Device

下面列出使用Direct3D播放视频的时候CreateDevice()的一个典型的代码。

```
[cpp]
1. IDirect3DDevice9 *m_pDirect3DDevice;
2. D3DPRESENT_PARAMETERS d3dpp;
3. ...
4. m_pDirect3D9->CreateDevice( D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, hwnd,
5.     D3DCREATE_SOFTWARE_VERTEXPROCESSING,
6.     &d3dpp, &m_pDirect3DDevice );
```

2)

基于Device创建一个Surface

通过IDirect3DDevice9接口的CreateOffscreenPlainSurface ()方法即可创建一个Surface(离屏表面。所谓的“离屏”指的是永远不在屏幕上显示)。

CreateOffscreenPlainSurface ()的函数原型如下所示：

```
[cpp]
1. HRESULT CreateOffscreenPlainSurface(UINT width,
2.     UINT height,
3.     D3DFORMAT format,
4.     D3DPPOOL pool,
5.     IDirect3DSurface9 ** result,
6.     HANDLE * unused
7. );
```

其中每个参数的含义如下所列：

Width：离屏表面的宽。

Height：离屏表面的高。

Format：离屏表面的像素格式（例如：32位像素格式为D3DFMT_A8R8G8B8）

Pool：D3DPOOL定义了资源对应的内存类型，例如如下几种类型。

D3D3POOL_DEFAULT：默认值，表示存在于显卡的显存中。

D3D3POOL_MANAGED：由Direct3D自由调度内存的位置（显存或者缓存中）。

D3DPOOL_SYSTEMMEM：表示位于内存中。

Result：返回创建的Surface。

Unused：还未研究。

下面给出一个使用Direct3D播放视频的时候CreateTexture()的典型代码。该代码创建了一个像素格式为YV12的离屏表面，存储于显卡的显存中。

```
[cpp]
1. IDirect3DDevice9 *m_pDirect3DDevice;
2. IDirect3DSurface9 *m_pDirect3DSurfaceRender;
3. ...
4. m_pDirect3DDevice->CreateOffscreenPlainSurface(
5.     lWidth, lHeight,
6.     (D3DFORMAT)MAKEFOURCC('Y', 'V', '1', '2'),
7.     D3DPOOL_DEFAULT,
8.     &m_pDirect3DSurfaceRender,
9.     NULL);
```

创建Surface完成之后，初始化工作就完成了。

3.

循环显示画面

循环显示画面就是一帧一帧的读取YUV/RGB数据，然后显示在屏幕上的过程，下面详述一下步骤。

1) 清理

在显示之前，通过IDirect3DDevice9接口的Clear()函数可以清理Surface。个人感觉在播放视频的时候用不用这个函数都可以。因为视频本身就是全屏显示的。显示下一帧的时候自然会覆盖前一帧的所有内容。Clear()函数的原型如下所示：

```
[cpp]
1. HRESULT Clear(
2.     DWORD Count,
3.     const D3DRECT *pRects,
4.     DWORD Flags,
5.     D3DCOLOR Color,
6.     float Z,
7.     DWORD Stencil
8. );
```

其中每个参数的含义如下所列：

Count：说明你要清空的矩形数目。如果要清空的是整个客户区窗口，则设为0；

pRects：这是一个D3DRECT结构体的一个数组，如果count中设为5，则这个数组中就得有5个元素。

Flags：一些标记组合。只有三种标记：D3DCLEAR_STENCIL, D3DCLEAR_TARGET, D3DCLEAR_ZBUFFER。

Color：清除目标区域所使用的颜色。

float：设置Z缓冲的Z初始值。Z缓冲还没研究过。

Stencil：这个在播放视频的时候也没有用到。

下面给出一个使用Direct3D播放视频的时候IDirect3DDevice9的Clear()的典型代码。

```
[cpp]
1. IDirect3DDevice9 *m_pDirect3DDevice;
2. m_pDirect3DDevice->Clear(0, NULL, D3DCLEAR_TARGET, D3DCOLOR_XRGB(0, 0, 255), 1.0f, 0);
```

上述代码运行完后，屏幕会变成蓝色（R,G,B取值为0,0,255）。

2) 一帧视频数据拷贝至Surface

操作Surface的像素数据，需要使用IDirect3DSurface9的LockRect()和UnlockRect()方法。使用LockRect()锁定纹理上的一块矩形区域，该矩形区域被映射成像素数组。利用函数返回的D3DLOCKED_RECT结构体，可以对数组中的像素进行直接存取。LockRect()函数原型如下。

```
[cpp]
1. HRESULT LockRect(
2.     D3DLOCKED_RECT *pLockedRect,
3.     const RECT *pRect,
4.     DWORD Flags
5. );
```

每个参数的意义如下：

pLockedRect：返回的一个D3DLOCKED_RECT结构体用于描述被锁定的区域。

pRect：使用一个RECT结构体指定需要锁定的区域。如果为NULL的话就是整个区域。

Flags：暂时还没有细研究。

其中D3DLOCKED_RECT结构体定义如下所示。

```
[cpp]
1. typedef struct _D3DLOCKED_RECT
2. {
3.     INT Pitch;
4.     void* pBits;
5. } D3DLOCKED_RECT;
```

两个参数的意义如下：

Pitch：surface中一行像素的数据量（Bytes）。注意这个的值并不一定等于实际像素数据一行像素的数据量（通常会大一些），它取值一般是4的整数倍。

pBits：指向被锁定的数据。

使用LockRect()函数之后，就可以对其返回的D3DLOCKED_RECT中的数据进行操作了。例如memcpy()等。操作完成后，调用UnlockRect()方法。

下面给出一个使用Direct3D的Surface播放视频的时候IDirect3DSurface9的数据拷贝的典型代码。该代码拷贝了YUV420P的数据至Surface。


```

1. IDirect3DSurface9 *m_pDirect3DSurfaceRender;
2. HRESULT lRet;
3. ...
4. D3DLOCKED_RECT d3d_rect;
5. lRet=m_pDirect3DSurfaceRender->LockRect(&d3d_rect,NULL,D3DLOCK_DONOTWAIT);
6. if(FAILED(lRet))
7.     return -1;
8. byte *pSrc = buffer;
9. byte * pDest = (BYTE *)d3d_rect.pBits;
10. int stride = d3d_rect.Pitch;
11. unsigned long i = 0;
12.
13. //Copy Data (YUV420P)
14. for(i = 0;i < pixel_h;i++){
15.     memcpy(pDest + i * stride,pSrc + i * pixel_w, pixel_w);
16. }
17. for(i = 0;i < pixel_h/2;i++){
18.     memcpy(pDest + stride * pixel_h + i * stride / 2,pSrc + pixel_w * pixel_h + pixel_w * pixel_h / 4 + i * pixel_w / 2, pixel_w / 2)
19. }
20. for(i = 0;i < pixel_h/2;i++){
21.     memcpy(pDest + stride * pixel_h + stride * pixel_h / 4 + i * stride / 2,pSrc + pixel_w * pixel_h + i * pixel_w / 2, pixel_w / 2);
22. }
23.
24. lRet=m_pDirect3DSurfaceRender->UnlockRect();

```

3)

开始一个Scene

使用IDirect3DDevice9接口的BeginScene()开始一个Scene。Direct3D中规定所有绘制方法都必须在BeginScene()和EndScene()之间完成。这个函数没有参数。

4)

Surface数据拷贝至后台缓冲表面

使用IDirect3DDevice9接口的GetBackBuffer() 可以获得后台缓冲表面。然后使用StretchRect()方法可以将Surface的数据拷贝至后台缓冲表面中，等待显示。

GetBackBuffer()函数原型如下。

```

1. HRESULT GetBackBuffer(
2.     UINT iSwapChain,
3.     UINT BackBuffer,
4.     D3DBACKBUFFER_TYPE Type,
5.     IDirect3DSurface9 ** ppBackBuffer
6. );

```

函数中参数含义如下：

iSwapChain：指定正在使用的交换链索引。

BackBuffer：后台缓冲表面索引。

Type：后台缓冲表面的类型。

ppBackBuffer：保存后台缓冲表面的LPDIRECT3DSURFACE9对象。

StretchRect()可以将一个矩形区域的像素从设备内存的一个Surface转移到另一个Surface上。StretchRect()函数的原型如下。

```

1. HRESULT StretchRect(
2.     IDirect3DSurface9 * pSourceSurface,
3.     CONST RECT * pSourceRect,
4.     IDirect3DSurface9 * pDestSurface,
5.     CONST RECT * pDestRect,
6.     D3DTEXTUREFILTERTYPE Filter
7. );

```

函数中参数含义如下：

pSourceSurface：指向源Surface的指针。

pSourceRect：使用一个 RECT结构体指定源Surface需要复制的区域。如果为NULL的话就是整个区域。

pDestSurface：指向目标Surface的指针。

pDestRect：使用一个 RECT结构体指定目标Surface的区域。

Filter：设置图像大小变换的时候的插值方法。例如：

D3DTEXF_POINT：邻域法。质量较差。

D3DTEXF_LINEAR：线性插值，最常用。

下面给出的代码将离屏表面的数据传给了后台缓冲表面。一但传给了后台缓冲表面，就可以用于显示了。


```
[cpp]
1. IDirect3DDevice9 *m_pDirect3DDevice;
2. IDirect3DSurface9 *m_pDirect3DSurfaceRender;
3. IDirect3DSurface9 * pBackBuffer;
4.
5.
6. m_pDirect3DDevice->GetBackBuffer(0,0,D3DBACKBUFFER_TYPE_MONO,&pBackBuffer);
7. m_pDirect3DDevice->StretchRect(m_pDirect3DSurfaceRender,NULL,pBackBuffer,&m_rtViewport,D3DTEXF_LINEAR);
```

5)
结束Scene

EndScene()和BeginScene()是成对出现的，不再解释。

6)
显示

使用IDirect3DDevice9接口的Present ()显示结果。Present ()的原型如下。

```
[cpp]
1. HRESULT Present(
2.     const RECT *pSourceRect,
3.     const RECT *pDestRect,
4.     HWND hDestWindowOverride,
5.     const RGNDATA *pDirtyRegion
6. );
```

几个参数的意义如下：

pSourceRect：你想要显示的后台缓冲表面区的一个矩形区域。设为NULL则表示要把整个后台缓冲表面区的内容都显示。

pDestRect：表示一个显示区域。设为NULL表示整个客户显示区。

hDestWindowOverride：你可以通过它来把显示的内容显示到不同的窗口去。设为NULL则表示显示到主窗口。

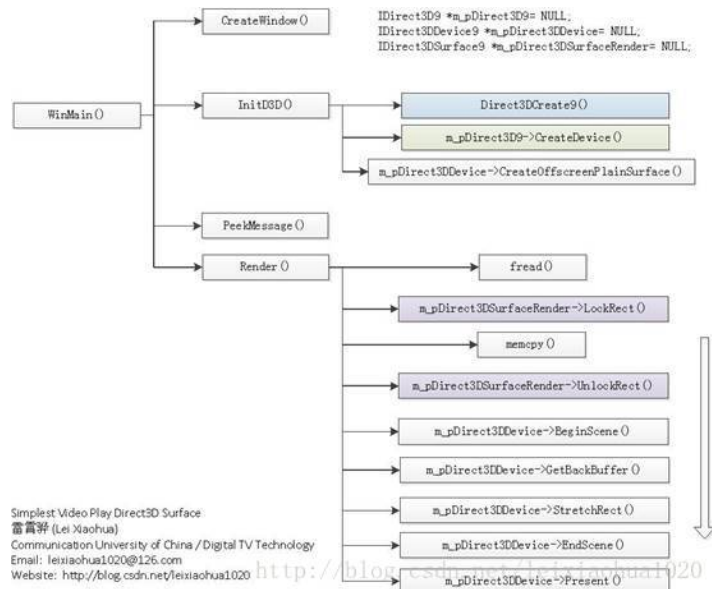
pDirtyRegion：一般设为NULL

下面给出一个使用Direct3D播放视频的时候IDirect3DDevice9的Present ()的典型代码。从代码可以看出，全部设置为NULL就可以了。

```
[cpp]
1. IDirect3DDevice9 *m_pDirect3DDevice;
2. ...
3. m_pDirect3DDevice->Present( NULL, NULL, NULL, NULL );
```

播放视频流程总结

文章至此，使用Direct3D显示YUV/RGB的全部流程就记录完毕了。最后贴一张图总结上述流程。



代码

完整的代码如下所示。

```
[cpp]
1. /**
2.  * 最简单的Direct3D播放视频的例子 (Direct3D播放RGB/YUV) [Surface]
3.  * Simplest Video Play Direct3D (Direct3D play RGB/YUV)[Surface]
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. * 本程序使用Direct3D播放RGB/YUV视频像素数据。使用D3D中的Surface渲染数据。
12. * 使用Surface渲染视频相对于另一种方法（使用Texture）来说，更加简单，适合
13. * 新手学习。
14. * 函数调用步骤如下：
15. *
16. * [初始化]
17. * Direct3DCreate9()：获得IDirect3D9
18. * IDirect3D9->CreateDevice()：通过IDirect3D9创建Device（设备）。
19. * IDirect3DDevice9->CreateOffscreenPlainSurface()：通过Device创建一个Surface（离屏表面）。
20. *
21. * [循环渲染数据]
22. * IDirect3DSurface9->LockRect()：锁定离屏表面。
23. * memcpy()：填充数据
24. * IDirect3DSurface9->UnlockRect()：解锁离屏表面。
25. * IDirect3DDevice9->BeginScene()：开始绘制。
26. * IDirect3DDevice9->GetBackBuffer()：获得后备缓冲。
27. * IDirect3DDevice9->StretchRect()：拷贝Surface数据至后备缓冲。
28. * IDirect3DDevice9->EndScene()：结束绘制。
29. * IDirect3DDevice9->Present()：显示出来。
30. *
31. * This software play RGB/YUV raw video data using Direct3D. It uses Surface
32. * in D3D to render the pixel data. Compared to another method (use Texture),
33. * it is more simple and suitable for the beginner of Direct3D.
34. * The process is shown as follows:
35. *
36. * [Init]
37. * Direct3DCreate9(): Get IDirect3D9.
38. * IDirect3D9->CreateDevice(): Create a Device.
39. * IDirect3DDevice9->CreateOffscreenPlainSurface(): Create a Offscreen Surface.
40. *
41. * [Loop to Render data]
42. * IDirect3DSurface9->LockRect(): Lock the Offscreen Surface.
43. * memcpy(): Fill pixel data...
44. * IDirect3DSurface9->UnlockRect(): Unlock the Offscreen Surface.
45. * IDirect3DDevice9->BeginScene(): Begin drawing.
46. * IDirect3DDevice9->GetBackBuffer(): Get BackBuffer.
47. * IDirect3DDevice9->StretchRect(): Copy Surface data to BackBuffer.
48. * IDirect3DDevice9->EndScene(): End drawing.
49. * IDirect3DDevice9->Present(): Show on the screen.
50. */
51.
52. #include <stdio.h>
53. #include <tchar.h>
54. #include <d3d9.h>
55.
56. CRITICAL_SECTION m_critial;
57.
58. IDirect3D9 *m_pDirect3D9= NULL;
59. IDirect3DDevice9 *m_pDirect3DDevice= NULL;
60. IDirect3DSurface9 *m_pDirect3DSurfaceRender= NULL;
61.
62. RECT m_rtViewport;
63.
64. //set '1' to choose a type of file to play
65. //Read BGRA data
66. #define LOAD_BGRA 0
67. //Read YUV420P data
68. #define LOAD_YUV420P 1
69.
70.
71. //Width, Height
72. const int screen_w=500,screen_h=500;
73. const int pixel_w=320,pixel_h=180;
74. FILE *fp=NULL;
75.
76. //Bit per Pixel
77. #if LOAD_BGRA
78. const int bpp=32;
79. #elif LOAD_YUV420P
80. const int bpp=12;
81. #endif
82.
83. unsigned char buffer[pixel_w*pixel_h*bpp/8];
84.
85.
86. void Cleanup()
87. {
88.     EnterCriticalSection(&m_critial);
89.     if(m_pDirect3DSurfaceRender)
90.         m_pDirect3DSurfaceRender->Release();
91.     if(m_pDirect3DDevice)
92.         m_pDirect3DDevice->Release();
93.     if(m_pDirect3D9)
94.         m_pDirect3D9->Release();
95.     LeaveCriticalSection(&m_critial);
96. }

```

```

97.
98.
99. int InitD3D( HWND hwnd, unsigned long lWidth, unsigned long lHeight )
100. {
101.     HRESULT lRet;
102.     InitializeCriticalSection(&m_critial);
103.     Cleanup();
104.
105.     m_pDirect3D9 = Direct3DCreate9( D3D_SDK_VERSION );
106.     if( m_pDirect3D9 == NULL )
107.         return -1;
108.
109.     D3DPRESENT_PARAMETERS d3dpp;
110.     ZeroMemory( &d3dpp, sizeof(d3dpp) );
111.     d3dpp.Windowed = TRUE;
112.     d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;
113.     d3dpp.BackBufferFormat = D3DFMT_UNKNOWN;
114.
115.     GetClientRect(hwnd,&m_rtViewport);
116.
117.     lRet=m_pDirect3D9->CreateDevice( D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL,hwnd,
118.         D3DCREATE_SOFTWARE_VERTEXPROCESSING,
119.         &d3dpp, &m_pDirect3DDevice );
120.     if(FAILED(lRet))
121.         return -1;
122.
123.     #if LOAD_BGRA
124.         lRet=m_pDirect3DDevice->CreateOffscreenPlainSurface(
125.             lWidth,lHeight,
126.             D3DFMT_X8R8G8B8,
127.             D3DPPOOL_DEFAULT,
128.             &m_pDirect3DSurfaceRender,
129.             NULL);
130.     #elif LOAD_YUV420P
131.         lRet=m_pDirect3DDevice->CreateOffscreenPlainSurface(
132.             lWidth,lHeight,
133.             (D3DFORMAT)MAKEFOURCC('Y', 'V', '1', '2'),
134.             D3DPPOOL_DEFAULT,
135.             &m_pDirect3DSurfaceRender,
136.             NULL);
137.     #endif
138.
139.
140.     if(FAILED(lRet))
141.         return -1;
142.
143.     return 0;
144. }
145.
146.
147. bool Render()
148. {
149.     HRESULT lRet;
150.     //Read Data
151.     //RGB
152.     if (fread(buffer, 1, pixel_w*pixel_h*bpp/8, fp) != pixel_w*pixel_h*bpp/8){
153.         // Loop
154.         fseek(fp, 0, SEEK_SET);
155.         fread(buffer, 1, pixel_w*pixel_h*bpp/8, fp);
156.     }
157.
158.     if(m_pDirect3DSurfaceRender == NULL)
159.         return -1;
160.     D3DLOCKED_RECT d3d_rect;
161.     lRet=m_pDirect3DSurfaceRender->LockRect(&d3d_rect,NULL,D3DLOCK_DONOTWAIT);
162.     if(FAILED(lRet))
163.         return -1;
164.
165.     byte *pSrc = buffer;
166.     byte * pDest = (BYTE *)d3d_rect.pBits;
167.     int stride = d3d_rect.Pitch;
168.     unsigned long i = 0;
169.
170.     //Copy Data
171.     #if LOAD_BGRA
172.         int pixel_w_size=pixel_w*4;
173.         for(i=0; i< pixel_h; i++){
174.             memcpy( pDest, pSrc, pixel_w_size );
175.             pDest += stride;
176.             pSrc += pixel_w_size;
177.         }
178.     #elif LOAD_YUV420P
179.         for(i = 0; i < pixel_h; i++){
180.             memcpy(pDest + i * stride,pSrc + i * pixel_w, pixel_w);
181.         }
182.         for(i = 0; i < pixel_h/2; i++){
183.             memcpy(pDest + stride * pixel_h + i * stride / 2,pSrc + pixel_w * pixel_h + pixel_w * pixel_h / 4 + i * pixel_w / 2, pixel_w
184. );
185.         }
186.         for(i = 0; i < pixel_h/2; i++){
187.             memcpy(pDest + stride * pixel_h + stride * pixel_h / 4 + i * stride / 2,pSrc + pixel_w * pixel_h + i * pixel_w / 2, pixel_w /

```

```

187.     }
188. #endif
189.
190.     lRet=m_pDirect3DSurfaceRender->UnlockRect();
191.     if(FAILED(lRet))
192.         return -1;
193.
194.     if (m_pDirect3DDevice == NULL)
195.         return -1;
196.
197.     m_pDirect3DDevice->Clear( 0, NULL, D3DCLEAR_TARGET, D3DCOLOR_XRGB(0,0,0), 1.0f, 0 );
198.     m_pDirect3DDevice->BeginScene();
199.     IDirect3DSurface9 * pBackBuffer = NULL;
200.
201.     m_pDirect3DDevice->GetBackBuffer(0,0,D3DBACKBUFFER_TYPE_MONO,&pBackBuffer);
202.     m_pDirect3DDevice->StretchRect(m_pDirect3DSurfaceRender,NULL,pBackBuffer,&m_rtViewport,D3DTEXF_LINEAR);
203.     m_pDirect3DDevice->EndScene();
204.     m_pDirect3DDevice->Present( NULL, NULL, NULL, NULL );
205.     pBackBuffer->Release();
206.
207.     return true;
208. }
209.
210.
211. LRESULT WINAPI MyWndProc(HWND hwnd, UINT msg, WPARAM wparam, LPARAM lparam)
212. {
213.     switch(msg){
214.     case WM_DESTROY:
215.         Cleanup();
216.         PostQuitMessage(0);
217.         return 0;
218.     }
219.     return DefWindowProc(hwnd, msg, wparam, lparam);
220. }
221.
222. int WINAPI WinMain( __in HINSTANCE hInstance, __in_opt HINSTANCE hPrevInstance, __in LPSTR lpCmdLine, __in int nShowCmd )
223. {
224.     WNDCLASSEX wc;
225.     ZeroMemory(&wc, sizeof(wc));
226.
227.     wc.cbSize = sizeof(wc);
228.     wc.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
229.     wc.lpfnWndProc = (WNDPROC)MyWndProc;
230.     wc.lpszClassName = L"D3D";
231.     wc.style = CS_HREDRAW | CS_VREDRAW;
232.
233.     RegisterClassEx(&wc);
234.
235.     HWND hwnd = NULL;
236.     hwnd = CreateWindow(L"D3D", L"Simplest Video Play Direct3D (Surface)", WS_OVERLAPPEDWINDOW, 100, 100, 500, 500, NULL, NULL, hInst
nce, NULL);
237.     if (hwnd==NULL){
238.         return -1;
239.     }
240.
241.     if(InitD3D( hwnd, pixel_w, pixel_h)==E_FAIL){
242.         return -1;
243.     }
244.
245.     ShowWindow(hwnd, nShowCmd);
246.     UpdateWindow(hwnd);
247.
248. #if LOAD_BGRA
249.     fp=fopen("../test_bgra_320x180.rgb","rb+");
250. #elif LOAD_YUV420P
251.     fp=fopen("../test_yuv420p_320x180.yuv","rb+");
252. #endif
253.     if(fp==NULL){
254.         printf("Cannot open this file.\n");
255.         return -1;
256.     }
257.
258.     MSG msg;
259.     ZeroMemory(&msg, sizeof(msg));
260.
261.     while (msg.message != WM_QUIT){
262.         //PeekMessage, not GetMessage
263.         if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE)){
264.             TranslateMessage(&msg);
265.             DispatchMessage(&msg);
266.         }
267.         else{
268.             Sleep(40);
269.             Render();
270.         }
271.     }
272.
273.
274.     UnregisterClass(L"D3D", hInstance);
275.     return 0;
276. }

```

代码注意事项

1. 可以通过设置定义在文件开始的宏，决定读取哪个格式的像素数据（bgra，yuv420p）。

```
[cpp]
1. //set '1' to choose a type of file to play
2. //Read BGRA data
3. #define LOAD_BGRA 0
4. //Read YUV420P data
5. #define LOAD_YUV420P 1
```

2. 窗口的宽高为screen_w，screen_h。像素数据的宽高为pixel_w,pixel_h。它们的定义如下。

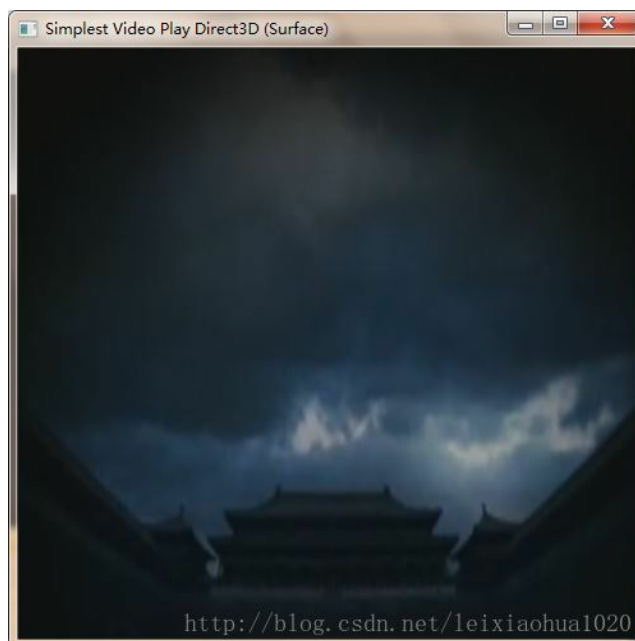
```
[cpp]
1. //Width, Height
2. const int screen_w=500,screen_h=500;
3. const int pixel_w=320,pixel_h=180;
```

3.其他要点

本程序使用的是Win32的API创建的窗口。但注意这个并不是MFC应用程序的窗口。MFC代码量太大，并不适宜用来做教程。因此使用Win32的API创建窗口。程序的入口函数是WinMain()，其中调用了CreateWindow()创建了显示视频的窗口。此外，程序中的消息循环使用的是PeekMessage()而不是GetMessage()。GetMessage()获取消息后，将消息从系统中移除，当系统无消息时，会等待下一条消息，是阻塞函数。而函数PeekMesssge()是以查看的方式从系统中获取消息，可以不将消息从系统中移除（相当于“偷看”消息），是非阻塞函数；当系统无消息时，返回FALSE，继续执行后续代码。使用PeekMessage()的好处是可以保证每隔40ms可以显示下一帧画面。

运行结果

不论选择读取哪个格式的文件，程序的最终输出效果都是一样的，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。

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/40279297>

文章标签：

Direct3D

Surface

渲染

视频

YUV

个人分类：

Direct3D

[我的开源项目](#)

此PDF由spygg生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com