DirectX 11 Dev

郑华

2016年3月5日

1 DirectX11 组件

1.Direct2D 组件 Direct2D 在 Win32 程序中被用于 2D 图形的绘制，它是一个高性能的矢量函数渲染库。

2.DirectWrite 组件 该组件被用于在使用 Direct2D 的应用程序中进行字体和文字的渲染。

3.DXGI 组件 DirectX 图形基础设施库，也就是著名的 DXGI 组件，用于创建 Direct3D 的缓存交换链和枚举设备适配器。

4.Direct3D 组件 Direct3D 组件用于在 DirectX 中构建所有的 3D 图形。它就是最受注意的并且更新最频繁的 API。本书的学习重点就是 Direct3D 组件。

5.XAudio2 组件 XAudio2 是一个低级的音频处理 API，是 XDK 的一部分(Xbox 开发套件)，而现在是 DirectX SDK 的一部分。XAudio2 取代了 DirectSound 组件。 XAudio 的最初版本用于 Xbox 游戏平台。

6.XACT3 组件 XACT3 是一个构建于 XAudio2 之上的高级音频处理 API。XACT3 允许开发者使用跨平台的音频创建工具来构建他们的应用程序中的声音。开发者如果需要在低层次上控制他们的音频系统可以使用 XAudio2 或者使用它来构建类似于 XACT3 的组件。

7.XInput 组件 XInput 组件是 XDK 和 DirectX SDK 中的输入控制 API 部分，被用于处理 Xbox360 游戏机的所有输入操作。本质上，你在 Xbox360 上的任何输入控制器都可被用于 PC 机，而 XInput 就是你用于在这些设备上工作的 API。

8.XNA Math 组件 新的 XNA Math 组件不仅仅是一个 API 而且更像一个在常见的视频游戏中实现了优化操作的数学库。

9.DirectCompute 组件 DirectCompute 组件是一个新加进 DirectX 11 的 API 集，允许使用 GPU 执行通用多线程计算。 GPU 能够并行处理多任务，比如物理模拟，视频压缩及解压，音频处理等等。

10.DirectSetup 组件 一旦你的游戏完成后，你想发布给其他人玩。 DirectSetup 组件提供一些用于在用户计算机上面安装最新版本的 DirectX 运行时的函数。它也能够检测用户电脑所安装的最新版本的 DirectX。

11.DirectInput 组件 DirectInput 组件用来检测键盘，鼠标和游戏操纵杆的输入。现在 XInput 被用于所有游戏的输入控制。对于键盘和鼠标我们可以使用 Win32 函数或者使用 DirectInput 处理，

2 Direct3D 初始化

steop 1. Description of driver types and feature level 检查 设备类型和特征级别

设备类型

①硬件设备hardware device 是一个运行在显卡上的D3D设备，在所有设备中运行速度是最快的

②参考设备reference device 是用于没有可用的硬件支持时在CPU上进行渲染的设备。参考设备就是利用软件，在CPU对硬件渲染设备的一个模拟。

③软件驱动设备software driverdevice 是开发人员自己编写的用于Direct3D的渲染驱动软件

④WARP设备WARPdevice 是一种高效的CPU渲染设备，可以模拟现阶段所有的Direct3D特性。WARP使用了Windows Vista /Windows 7/Winodws 8中的Windows Graphic 运行库中高度优化过的代码作为支撑，这让这种方式出类拔萃，相比与上文提到的参考设备reference device模式更加优秀

特征等级

①Direct3D11设备 Direct3D的特征等级用于指定需要设定的设备目标

②Direct3D10.1设备 Direct3D的特征等级用于指定需要设定的设备目标

③Direct3D10.0设备 Direct3D的特征等级用于指定需要设定的设备目标

④WARP设备或者参考设备 以上三种设备都无法支持的情况下，使用

示例代码

```
RECT dimensions;
GetClientRect( hwnd, &dimensions );
unsigned int width = dimensions.right - dimensions.left;
unsigned int height = dimensions.bottom - dimensions.top;
D3D_DRIVER_TYPE driverTypes[] =
{
 D3D_DRIVER_TYPE_HARDWARE, D3D_DRIVER_TYPE_WARP,D3D_DRIVER_TYPE_SOFTWARE
};
unsigned int totalDriverTypes = ARRAYSIZE( driverTypes );
D3D_FEATURE_LEVEL featureLevels[] =
 {
```

```
        D3D_FEATURE_LEVEL_11_0,
        D3D_FEATURE_LEVEL_10_1,
        D3D_FEATURE_LEVEL_10_0
};
unsigned int totalFeatureLevels = ARRAYSIZE( featureLevels );
```

step 2. create swap-chain, device and context 创建 D3D设备 交换链

通常在游戏中有，有两种颜色缓存，分别叫做主缓存和辅助缓存，他们一起被称为前后台缓存组合。主缓存中的内容（前台缓存）会显示在屏幕上，而辅助缓存（后台缓存）用于绘制下一帧

这种技术在计算机图形学中叫做双缓冲（doublebuffering），或者叫页面翻转（page flipping）（这种技术我们之前的一系列Win32 GDI demo中使用得比较勤，研究了之前的demo的朋友们应该已经耳濡目染了吧）。一个交换链能拥有一个或者多个这样的缓冲。

下一步是创建渲染上下文，渲染设备，以及我们拥有的交换链描述。D3D设备一般都是设备本身和硬件之间的通信，而D3D上下文是一种描述设备如何绘制的渲染设备上下文，这也包含了渲染状态和其他的绘图信息。正如我们讨论过的，交换链是设备和上下文将要绘制的渲染目标。

Direct3D 设备类型是 ID3D11Device，渲染环境类型是 ID3D11Context，交换链类型是 IDXGISwapChain

示例代码

```
bool Dx11DemoBase::CreateDeviceAndSwapChain
( unsigned int totalDriverTypes, D3D_DRIVER_TYPE driverTypes[],
unsigned int totalFeatureLevels, D3D_FEATURE_LEVEL featureLevels[])
{
RECT dimensions;
GetClientRect( hwnd_, &dimensions );
unsigned int width = dimensions.right - dimensions.left;
unsigned int height = dimensions.bottom - dimensions.top;


DXGI_SWAP_CHAIN_DESC swapChainDesc;
ZeroMemory( &swapChainDesc, sizeof( swapChainDesc));
swapChainDesc.BufferCount = 1;
swapChainDesc.BufferDesc.Width = width;
swapChainDesc.BufferDesc.Height = height;
swapChainDesc.BufferDesc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;
swapChainDesc.BufferDesc.RefreshRate.Numerator = 60;
swapChainDesc.BufferDesc.RefreshRate.Denominator = 1;
swapChainDesc.BufferUsage = DXGI_USAGE_RENDER_TARGET_OUTPUT;
swapChainDesc.OutputWindow = hwnd_;
swapChainDesc.Windowed = true;
swapChainDesc.SampleDesc.Count = 1;
swapChainDesc.SampleDesc.Quality = 0;
```

```
unsigned int creationFlags = 0;
#ifdef _DEBUG
creationFlags |= D3D11_CREATE_DEVICE_DEBUG;
#endif
HRESULT result;
unsigned int driver = 0;
for( driver = 0; driver < totalDriverTypes; ++ driver)
{
result = D3D11CreateDeviceAndSwapChain(
0,
driverTypes[ driver],
0,
creationFlags,
featureLevels,
totalFeatureLevels,
D3D11_SDK_VERSION,
&swapChainDesc,
&swapChain_,
&d3dDevice_,
&featureLevel_,
&d3dContext_
);

if( SUCCEEDED( result))
{
driverType_ = driverTypes[ driver];
break;
}
}

if( FAILED( result))
{
DXTRACE_MSG( "Failed to create the Direct3D device!");
return false;
}

return true;
}
```

step 3. set render target view 设置 渲染目标

渲染目标视图是写入联合输出阶段的一种 Direct3D 资源。为了在交换链的向后缓存(辅助缓存)中联合渲染，于是我们创建渲染目标视图

每当我们想渲染具体的渲染目标时，都必须在任何绘制调用之前设置它，通过调用 OMSetRenderTarget 函数来完成，该函数是联合输出的一部分(因此 OM output merger 在 OMSetRenderTarget 中)。

step 4. set viewport 设置 视口

视口 Viewport 定义为我们要在屏幕上渲染的区域