# Tutorial 1: Direct3D 11 Basics

## Summary:

在本教程中，我们将介绍创建一个最小D3D11程序所需的元素。每个D3D11程序必须有这些元素才能正常工作。这些元素包括：设置窗口和设备对象，然后在窗口里显示颜色。

## Source

(SDK root)\Samples\C++\Direct3D11\Tutorials\Tutorial01

## Setting Up The Direct3D 11 Device

创建窗口和消息循环的第一步在D3D 9、10、11是相同的。此过程的介绍请参考Direct3D 10 Tutorial 00: Win32 Basics。现在我们已经有一个显示的窗口，我们需要继续建立D3D11的设备。如果我们要渲染任何3D场景，设置是必要的。这第一件事情就是去建立三个对象：a device、an immediate context、a swap chain。The immediate context是D3D11 里面一个新对象。

在D3D10中，Device object 是被用来执行渲染和资源创建。在D3D11中，Immediate context 是被应用程序用来执行渲染到buffer，Device object包含资源创建的方法。

Swap chain负责获取Device渲染的buffer，并将内容在实际的显示器上面显示。Swap chain 包含两个或更多的buffers，主要是front 和 back buffer。这些是Device为了在显示器上显示而渲染的textures。Front buffer是当前正在呈现给用户的内容。这个Buffer是只读的，不能修改。Back buffer是Device将要绘制的渲染目标。一旦它完成了绘制操作，Swap chain将通过交换两个buffer来呈现Back buffer。Back buffer变成Front buffer，反之亦然。

要创建Swap chain，我们要设置DXGI\_SWAPCHAIN\_DESC结构体来描述我们即将创建的Swap chain。有几个Field值得一提。BackBufferUsage 是告诉应用程序Back buffer 将被如何使用的一个标志。在这种情况下，我们想渲染到Back buffer，所以我们将BackBufferUsage 设为 DXGI\_USAGE\_RENDER\_TARGET\_OUTPUT。OutputWindow field 表示Swap chain 将要用来呈现在屏幕图像的窗口。SampleDesc 是用来开启multi-sampling。本教程不使用multi-sampling，所以SampleDesc 的count设为1，Quality 设为0以禁用 multi-sampling。

一旦设置完毕，我们就可以调用D3D11CreateDeviceAndSwapChain 函数去创建Device 和 Swap chain。以下是代码：

1. DXGI\_SWAP\_CHAIN\_DESC sd;
2. ZeroMemory( &sd, **sizeof**(sd) );
3. sd.BufferCount = 1;
4. sd.BufferDesc.Width = 640;
5. sd.BufferDesc.Height = 480;
6. sd.BufferDesc.Format = DXGI\_FORMAT\_R8G8B8A8\_UNORM;
7. sd.BufferDesc.RefreshRate.Numerator = 60;
8. sd.BufferDesc.RefreshRate.Denominator = 1;
9. sd.BufferUsage = DXGI\_USAGE\_RENDER\_TARGET\_OUTPUT;
10. sd.OutputWindow = g\_hWnd;
11. sd.SampleDesc.Count = 1;
12. sd.SampleDesc.Quality = 0;
13. sd.Windowed = TRUE;
15. **if**( FAILED( D3D11CreateDeviceAndSwapChain( NULL, D3D\_DRIVER\_TYPE\_HARDWARE, NULL, 0, featureLevels, numFeatureLevels,
16. D3D11\_SDK\_VERSION, &sd, &g\_pSwapChain, &g\_pd3dDevice, NULL, &g\_pImmediateContext ) ) )
17. {
18. **return** FALSE;
19. }

我们需要做的下一件事情就是去创建Render target view。在D3D 11中Render target view是资源视图的一个类型。资源视图允许将资源绑定到图形学pipeline的特定stage。可以将资源视图和C语言中类型转换相类比。在C语言中，一块原始内存可以转换为任何数据类型。我们可以将原始内存转化为整数数组、浮点数数组、结构体、结构体数组等等。如果我们不知道它的类型，原始内存本身对我们不是很有用。D3D 11中资源视图以类似的方式起作用。例如，类似于原始内存的2D texture是原始底层资源。一旦我们有了这个资源，我们可以创建不同的资源视图，以不同的格式将该texture 绑定到图形学pipeline的不同stage：做为要渲染的render target，做为接受深度信息的depth stencil buffer，或做为纹理资源。C中类型转换运行内存块以不同的方式使用，D3D 11资源视图也是如此。

因为我们想绑定swap chain 的back buffer 做为render target，所以我们需要建立render target view。这使得D3D 11可以渲染到它上面。我们首先调用GetBuffer() 去获得back buffer 对象。视需要的，我们可以通过设置D3D11\_RENDERTARGETVIEW\_DESC 结构体来描述我们即将创建的render target。这个描述性结构体通常是CreateRenderTargetView的第二个参数。然而，对于这些教程，默认的render target view已经足够了。默认的render target view可以通过传递NULL做为第二个参数来获得。一旦我们创建好render target view，我们可以在immediate context中调用OMSetRenderTargets()将其绑定到pipeline。这就确保了pipeline的输出写到back buffer。创建和设置render target view的代码如下：

// Create a render target view

ID3D11Texture2D **\***pBackBuffer**;**

**if(** FAILED**(** g\_pSwapChain**->**GetBuffer**(** 0**,** \_\_uuidof**(** ID3D11Texture2D **),** **(**LPVOID**\*)&**pBackBuffer **)** **)** **)**

**return** FALSE**;**

hr **=** g\_pd3dDevice**->**CreateRenderTargetView**(** pBackBuffer**,** **NULL,** **&**g\_pRenderTargetView **);**

pBackBuffer**->**Release**();**

**if(** FAILED**(** hr **)** **)**

**return** FALSE**;**

g\_pImmediateContext**->**OMSetRenderTargets**(** 1**,** **&**g\_pRenderTargetView**,** **NULL** **);** 、

在D3D 11可以渲染前我们需要做的最后一件事情就是初始化viewport。Viewport将裁剪空间坐标，其中X,Y：[-1,1]，Z[0,1]，映射到render target space，有时称作像素空间。在D3D 9中，如果应用程序没有设置viewport，则将默认viewport设置为与render target相同的大小。在D3D 11中，没有默认的viewport。因此，我们必须这样做才能在屏幕上面看到东西。由于我们将整个render target做为输出，因此我们将左上角点设置为(0,0)，宽高设成和render target一样。为此，使用下面的代码：

D3D11\_VIEWPORT vp**;**

vp**.**Width **=** **(**FLOAT**)**width**;**

vp**.**Height **=** **(**FLOAT**)**height**;**

vp**.**MinDepth **=** 0.0f**;**

vp**.**MaxDepth **=** 1.0f**;**

vp**.**TopLeftX **=** 0**;**

vp**.**TopLeftY **=** 0**;**

g\_pImmediateContext**->**RSSetViewports**(** 1**,** **&**vp **);**

## Modifying the Message Loop

我们已经设置了窗口和D3D 11 device，并且准备去渲染。然而，我们的消息循环仍然存在问题：它使用GetMessage()去获得消息。GetMessage()的问题是如果应用程序窗口队列没有消息，GetMessage() 直到消息可用一直阻塞而且不能返回。因此，当消息队列为空时，我们应用程序一直在GetMessage()等待，而不是做一些类似渲染的工作。通过使用PeekMessage()代替GetMessage()可以解决这个问题。PeekMessage()可以像GetMessage()检索信息，但是当没有消息等待时，PeekMessage()可以立即返回而不是阻塞。我们可以利用这个时间去做一些渲染。使用PeekMessage()修改后的信息循环如下所示：

MSG msg **=** **{**0**};**

**while(** WM\_QUIT **!=** msg**.**message **)**

**{**

**if(** PeekMessage**(** **&**msg**,** **NULL,** 0**,** 0**,** PM\_REMOVE **)** **)**

**{**

TranslateMessage**(** **&**msg **);**

DispatchMessage**(** **&**msg **);**

**}**

**else**

**{**

Render**();** // Do some rendering

**}**

**}**

## The Rendering Code

渲染是在Render()函数中完成的。本教程中，我们渲染最简单的场景，即用单色填充屏幕。在D3D 11中，使用单色填充render target简单的方法是使用immediate context 的ClearRenderTargetView()函数。首先我们定义了一个由四个浮点数组成的数组，这些浮点数描述了我们想要填充屏幕的颜色。然后，我们将它传递给ClearRenderTargetView()。在这个例子里，我们选择了一片蓝色。一旦我们填充我们的back buffer，我们调用swap chain的**Present()**方法来完成渲染。**Present()**负责将swap chain的back buffer内容显示在屏幕上，以便用户可以看到。Render()函数如下：

void Render**()**

**{**

//

// Clear the backbuffer

//

float ClearColor**[**4**]** **=** **{** 0.0f**,** 0.125f**,** 0.6f**,** 1.0f **};** // RGBA

g\_pd3dDevice**->**ClearRenderTargetView**(** g\_pRenderTargetView**,** ClearColor **);**

g\_pSwapChain**->**Present**(** 0**,** 0 **);**

**}**