# Direct3D 11教程3：Shaders和Effect系统

原文地址：<http://msdn.microsoft.com/en-us/library/ff729720.aspx>。

## 概览

在前面的教程中，我们设置一个顶点缓存并将一个三角形传递到GPU。现在，我们将看一下图形管线的各个阶段是如何工作的，我们会解释shader的概念和effect系统。

**注意**：这个教程的共享前一个教程的代码，但强调不同的部分。



## 源代码

(SDK root)\Samples\C++\Direct3D10\Tutorials\Tutorial03

## 图形管线

在前面的教程中，我们创建了顶点缓存，将一个顶点布局（**vertex layout**）连接到一个technique对象。现在，我们将要解释什么是shader以及它的工作原理。要完全理解shader，我们需要看一下图形管线的整个流程。

在教程2中，当我们调用**VSSetShader**和**PSShader**方法时，实际上是将shader绑定到管线的一个阶段中。然后，当我们调用**Draw**时，就开始处理传递到图形管线的顶点数据。下面的几个部分介绍了**Draw**命令执行之后的过程细节。

## Shader

在Direct3D 11中，shader存在于图形管线的不同阶段。它们是由GPU执行的短小代码程序，获取特定的输入数据，处理这些数据，然后将结果输出到管线的下一阶段。Direct3D 11支持三种基本类型的shader：**顶点着色器（vertex shader）**，**几何着色器（geometry shader）**和**像素着色器（pixel shader）**。顶点着色器以顶点作为输入，当每个顶点通过顶点缓存传递到GPU时顶点着色器会执行一次。几何着色器以图元作为输入，当每个图元传递到GPU时执行一次。图元可以是一个点、一条线或一个三角形。像素着色器以像素（有时又被称为**片段fragment**）作为输入，在你想要绘制的图元上的每个像素上执行一次。当使用Direct3D 11进行绘制时，GPU必须要有一个有效的顶点着色器和像素着色器。几何着色器是Direct3D 11的高级功能，是可选的，所以本教程不讨论几何着色器。在Direct3D 11中，还有外壳（hull）和domain（域）着色器用于曲面细分，计算（computer）着色器用于计算，这些内容请看其他示例。

### 顶点着色器

顶点着色器是执行在GPU上的处理顶点的小程序。可以将顶点着色器看出一个以顶点为参数的C函数，它处理出入的数据，输出为经过修改的顶点。在应用程序以顶点缓存的形式将顶点数据传输入GPU之后，会对每个顶点执行顶点着色器一次，将每个顶点数据作为顶点着色器的输入参数。

顶点着色器可以完成很多任务，最重要的就是变换。变换就是将顶点从一个坐标系统转换为另一个的处理过程。例如，一个3D场景中的三角形顶点的位置为(0, 0, 0) (1, 0, 0) (0, 1, 0)。当三角形绘制在一张2D纹理缓存时，GPU必须知道在一张2D纹理缓存何处绘制三角形，GPU必须知道点的2D坐标，正是变换帮助我们完成了上述工作。变换的细节将在下一个教程中加以讨论。本教程中我们会使用一个简单的顶点着色器，它只是将输入数据直接输出。

在教程中，我们使用**高级着色语言（High-Level Shading Language，HLSL）**编写shader，应用程序会在effect系统中使用这些shader。我们的顶点数据具有一个3D位置元素，本教程中的顶点着色器对输入数据不进行任何操作。顶点着色器的代码如下：

float4 VS( float4 Pos : POSITION ) : SV\_POSITION

{

 return Pos;

}

顶点着色器看起来很像C函数，HLSL使用与C类似的语法，这样C/C++程序员学起来会更容易。我们看到这个名为VS的顶点着色器使用一个float4类型的参数，返回值也是一个float4值。在HLSL中，一个float4类型的数据是一个有4个分量的矢量。每个分量都是浮点数。冒号定义参数的语义和返回值。前面已经提到过，HLSL中的语义表示数据的性质。在上面的shader中，我们选择POSITION作为Pos输入参数的语义，因为这个参数包含顶点位置。返回值的语义，SV\_POSITION，是一个具有特殊含义的预定义语义。这个语义告诉图形管线与这个语义对应的数据定义了剪裁空间的位置，而GPU需要这个位置数据在屏幕上绘制像素。（我们会在下一个教程中讨论剪裁空间。）在上面的shader中，我们获取输入的位置数据，然后将相同的数据返回到管线中。

### 像素着色器

现代计算机显示器通常是光栅显示的，即屏幕是一个由小点构成的二维网格，这些小点称为像素。每个像素包含一个独立于其他像素的颜色。当在屏幕上绘制三角形时，我们实际上并没有绘制一个完整的三角形，只是点亮被三角形区域覆盖的一组像素，如图2所示。



**图2. 左图：我们想绘制的三角形。右图：屏幕上的实际显示。**

将三角形转换为一组像素的过程称为**光栅化（rasterization）**。GPU首先判断哪些像素被三角形覆盖，然后调用这些像素的像素着色器。像素着色器的首要目的就是计算每个像素的颜色。像素着色器获取要着色的像素作为输入数据，计算像素的颜色，然后将输出结果返回到图形管线。输入数据来自于几何着色器，如果几何着色器不存在，就像本教程一样，则直接来自于顶点着色器。

上面的顶点着色器输出了一个语义为SV\_POSITION的float4数据，这个数据就是像素着色器的输入。因为像素着色器输出颜色值，所以输出类型为float4。输出的语义为SV\_TARGET 表示输出到渲染目标格式。代码如下：

float4 PS( float4 Pos : SV\_POSITION ) : SV\_Target

{

 return float4( 1.0f, 1.0f, 0.0f, 1.0f ); // Yellow, with Alpha = 1

}

## 创建Shader

在应用程序的代码中，我们需要创建一个顶点着色器和一个像素着色器，这两个对象即代表shader，由**D3DX11CompileFromFile()**方法创建，代码如下：

// 创建顶点着色器

if( FAILED( D3DX11CompileFromFile( "Tutorial03.fx", NULL, NULL, "VS", "vs\_4\_0", D3DCOMPILE\_ENABLE\_STRICTNESS, NULL, NULL, &pVSBlob, &pErrorBlob, NULL ) ) )

 return FALSE;

// 创建像素着色器

if( FAILED( D3DX11CompileFromFile( "Tutorial03.fx", NULL, NULL, "PS", "ps\_4\_0", D3DCOMPILE\_ENABLE\_STRICTNESS, NULL, NULL, &pPSBlob, &pErrorBlob, NULL ) ) )

 return FALSE;

## 整合在一起

在分析了图形管线之后，我们就会理解绘制一个三角形的过程。创建一个Direct3D程序需要两个步骤。一是创建顶点数据，我们已经在教程2中讨论过了。二是创建着色器转换要绘制的数据，这是在本教程中讨论的。