# **Shaders: ShaderLab and fixed function shaders**

着色器：着色器语言&固定功能着色器

This tutorial teaches you the first steps of creating your own shaders, to help you control the look of your game and optimise the performance of the graphics.

这个教程教您怎样制作自己的着色器，帮助您更好的控制您的游戏视觉效果并优化图形的性能。

Unity is equipped with a powerful shading and material language called ****ShaderLab****. In style it is similar to CgFX and Direct3D Effects (.FX) languages - it describes everything needed to display a [Material](http://docs.unity3d.com/540/Documentation/Manual/class-Material.html).

Unity配备了一个强大的阴影着色和材质的语言，称为着色器语言(ShaderLab)。它的语法风格类似CgFX和Direct3D特效（.FX）语言。它描述了显示[材质](http://docs.unity3d.com/540/Documentation/Manual/class-Material.html)(Material) 所需要的一切信息。

Shaders describe properties that are exposed in Unity’s [Material Inspector](http://docs.unity3d.com/540/Documentation/Manual/class-Material.html) and multiple shader implementations (****SubShaders****) targeted at different graphics hardware capabilities, each describing complete graphics hardware rendering state, and vertex/fragment programs to use. Shader programs are written in the high-level [Cg/HLSL](http://docs.unity3d.com/540/Documentation/Manual/SL-ShadingLanguage.html) programming language.

着色器的属性描述可在Unity的[材质检视面板](http://docs.unity3d.com/540/Documentation/Manual/class-Material.html)(Material Inspector)中查看。针对高低端不同的显卡差异，可以在着色器中用多个子着色器(SubShaders)对应实现。显卡渲染的每个状态都是用固定功能管线或者顶点/片段程序。顶点和片段程序是用[Cg/HLSL](http://docs.unity3d.com/540/Documentation/Manual/SL-ShadingLanguage.html)高级语言编写的。

In this tutorial we’ll describe how to write very simple shaders using so-called “fixed function” notation. In the [next chapter](http://docs.unity3d.com/540/Documentation/Manual/ShaderTut2.html) we’ll introduce vertex and fragment[shader programs](http://docs.unity3d.com/540/Documentation/Manual/SL-ShaderPrograms.html). We assume that the reader has a basic understanding of [OpenGL](http://opengl.org/documentation/red_book) or Direct3D render states, and has some knowledge of [HLSL](https://msdn.microsoft.com/en-us/library/bb509561.aspx), [Cg](http://http.developer.nvidia.com/Cg/Cg_language.html), [GLSL](http://www.opengl.org/documentation/glsl) or[Metal](https://developer.apple.com/library/ios/documentation/Metal/Reference/MetalShadingLanguageGuide/Introduction/Introduction.html) shader programming languages.

在本教程中，我们讲述了如何编写非常简单的一种叫做固定功能(fixed function)的着色器。在[下一章](http://docs.unity3d.com/540/Documentation/Manual/ShaderTut2.html)，我们将会介绍可编程管线( programmable pipelines)[着色器编程](http://docs.unity3d.com/540/Documentation/Manual/SL-ShaderPrograms.html)。我们假设读者对 [OpenGL](https://www.opengl.org/sdk/)或Direct3D的渲染状态，有一个基本的了解。并知道一些关于[HLSL](https://msdn.microsoft.com/en-us/library/bb509561.aspx)，[CG](http://http.developer.nvidia.com/Cg/Cg_language.html),[GLSL](https://www.opengl.org/documentation/glsl/)或者[Metal](https://developer.apple.com/library/ios/documentation/Metal/Reference/MetalShadingLanguageGuide/Introduction/Introduction.html)的着色器编程语言的一些知识。

## **Getting started**

开始

To create a new shader, either choose ****Assets > Create > Shader > Unlit Shader**** from the main menu, or duplicate an existing shader and work from that. The new shader can be edited by double-clicking it in the [Project View](http://docs.unity3d.com/540/Documentation/Manual/ProjectView.html).

要创建一个新的着色器，可以从菜单栏选择 Assets->Create->Shader，或者复制一个现有的着色器，在项目视图([Project View](http://docs.unity3d.com/540/Documentation/Manual/ProjectView.html))中双击它进行编辑。

Unity has a way of writing very simple shaders in so-called “fixed-function” notation. We’ll start with this for simplicity. Internally the fixed function shaders are converted to regular [vertex and fragment programs](http://docs.unity3d.com/540/Documentation/Manual/SL-ShaderPrograms.html) at shader import time.

Unity有一种被称为“”固定功能“”的非常简单的着色器，我们将以它作为开始。在内部，固定功能着色器在导入时间会被转化为规则的[顶点和片段着色器语言](http://docs.unity3d.com/540/Documentation/Manual/SL-ShaderPrograms.html)。

We’ll start with a very basic shader:

我们来开始创建一个非常简单的着色器：

Shader "Tutorial/Basic" {

Properties {

\_Color ("Main Color", Color) = (1,0.5,0.5,1)

}

SubShader {

Pass {

Material {

Diffuse [\_Color]

}

Lighting On

}

}

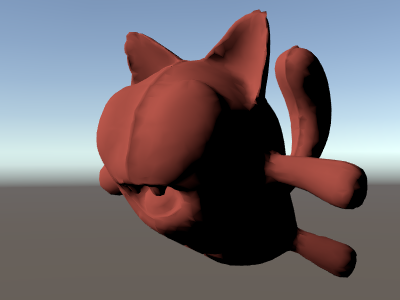
}

This simple shader demonstrates one of the most basic shaders possible. It defines a color property called ****Main Color**** and assigns it a default pink color (red=100% green=50% blue=50% alpha=100%). It then renders the object by invoking a ****Pass**** and in that pass setting the diffuse material component to the property ****\_Color**** and turning on per-vertex lighting.

简单说明一下这个简单但结构完整的着色器脚本。它定义了一个颜色属性，名字叫做Main Color，并赋于了玫瑰色的默认值，(红red=100% green=50% blue=50% alpha=100%)。然后通过Pass渲染物体，并传递设置漫反射材质组件到\_Color属性并打开每顶点光照。

To test this shader, create a new material, select the shader from the drop-down menu (****Tutorial->Basic****) and assign the Material to some object. Tweak the color in the Material Inspector and watch the changes. Time to move onto more complex things!

来测试这个着色器，创建一个新的材质，选从下拉菜单（Tutorial->Basic）择这个着色器文件，然后指定这个材质给某个物体。在材质检视面板来调整颜色，观看变化。随着时间的推移可做更复杂的事情。



## **Basic vertex lighting**

基本顶点光照

If you open an existing complex shader, it can be a bit hard to get a good overview. To get you started, we will dissect the built-in ****VertexLit**** shader that ships with Unity. This shader uses fixed-function pipeline to do standard per-vertex lighting.

如果您打开一个现有的复杂的着色器，您可能看不懂它。为了帮助您理解，我们将剖析Unity里内置的VertexLit着色器。这个着色器使用固定功能管线(fixed function pipeline )产生标准的顶点光照( per-vertex lighting)。

Shader "VertexLit" {

Properties {

\_Color ("Main Color", Color) = (1,1,1,0.5)

\_SpecColor ("Spec Color", Color) = (1,1,1,1)

\_Emission ("Emmisive Color", Color) = (0,0,0,0)

\_Shininess ("Shininess", Range (0.01, 1)) = 0.7

\_MainTex ("Base (RGB)", 2D) = "white" { }

}

SubShader {

Pass {

Material {

Diffuse [\_Color]

Ambient [\_Color]

Shininess [\_Shininess]

Specular [\_SpecColor]

Emission [\_Emission]

}

Lighting On

SeparateSpecular On

SetTexture [\_MainTex] {

constantColor [\_Color]

Combine texture \* primary DOUBLE, texture \* constant

}

}

}

}

All shaders start with the keyword [Shader](http://docs.unity3d.com/540/Documentation/Manual/SL-Shader.html) followed by a string that represents the name of the shader. This is the name that is shown in the ****Inspector****. All code for this shader must be put within the curly braces after it: ****{ }**** (called a block).

所有着色器由[Shader](http://docs.unity3d.com/540/Documentation/Manual/SL-Shader.html)关键字开始，随后的字符串表示着色器的名字。这个名字显示在检视面板。所有用于这个着色器的代码必须放置在之后的大括号中：{}（称为“块”）。

* The name should be short and descriptive. It does not have to match the ****.shader**** file name.
* 该名字应该是短且描述性的文字。它并不必匹配.shader文件名。
* To put shaders in submenus in Unity, use slashes - e.g. ****MyShaders/Test**** would be shown as ****Test**** in a submenu called ****MyShaders****, or ****MyShaders->Test****.
* 要把着色器加入到Unity的子菜单里，名字需要用 斜线(/)。例如：MyShaders/Test是一个名叫Test的着色器在MyShaders的子菜单下。MyShaders->Test。

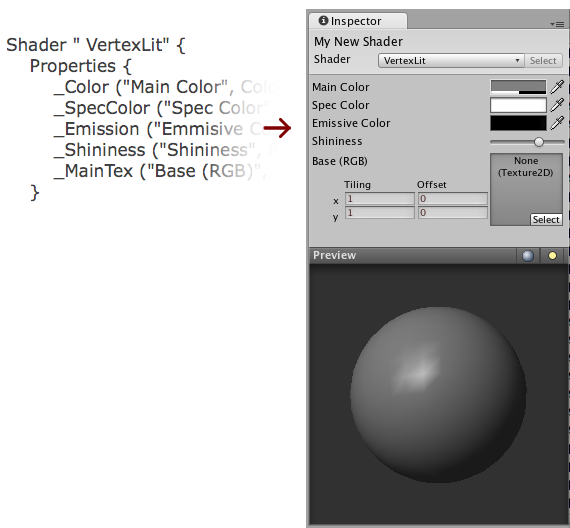
The shader is composed of a ****Properties**** block followed by ****SubShader**** blocks. Each of these is described in sections below.

这个着色器是由 Properties 块和SubShader块组成的。下面来说明这两部分。

## **Properties 属性**

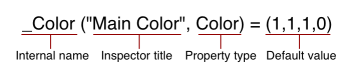
At the beginning of the shader block you can define any properties that artists can edit in the [Material Inspector](http://docs.unity3d.com/540/Documentation/Manual/class-Material.html). In the ****VertexLit**** example the properties look like this:

properties在着色器的起始部分，你可以定义任何属性，用户可以在材质检视面板([Material Inspector](http://docs.unity3d.com/540/Documentation/Manual/class-Material.html))里编辑。在VertexLit 例子里属性与材质检视面板如下图：



The properties are listed on separate lines within the [Properties](http://docs.unity3d.com/540/Documentation/Manual/SL-Properties.html) block. Each property starts with the internal name (****Color****, ****MainTex****). After this in parentheses comes the name shown in the inspector and the type of the property. After that, the default value for this property is listed:

Properties块内的语法都是单行的。每个[属性](http://docs.unity3d.com/540/Documentation/Manual/SL-Properties.html)都是由内部名称 开始，后面括号中是显示在检视面板(Inspector)中的名字和该属性的类型。等号后边跟的是默认值。



The list of possible types are in the [Properties Reference](http://docs.unity3d.com/540/Documentation/Manual/SL-Properties.html). The default value depends on the property type. In the example of a color, the default value should be a four component vector.

更多的属性(Properties)类型请参考：[Properties Reference](http://docs.unity3d.com/540/Documentation/Manual/SL-Properties.html)。默认值取决于属性(Properties)的类型。在上面的例子中属性是Color，默认值应该是一个四维向量。

We now have our properties defined, and are ready to start writing the actual shader.

现在我们已经定义好了需要的属性，可以开始写自己的着色器(shader)了

## **The shader body 着色器结构**

Before we move on, let’s define the basic structure of a shader file.

在我们继续之前，让我们定义着色器文件的基本结构。

Different graphic hardware has different capabilities. For example, some graphics cards support fragment programs and others don’t; some can lay down four textures per pass while the others can do only two or one; etc. To allow you to make full use of whatever hardware your user has, a shader can contain multiple ****SubShaders****. When Unity renders a shader, it will go over all subshaders and use the first one that the hardware supports.

不同的显卡有不同的渲染能力。例如，某些显卡支持片段程序( fragment programs)和别人不一样，有的可以放下四个纹理通道(textures per pass)而有些只放下一个或两个。为了适应不同的显卡，一个着色器可以包含多个子着色器([SubShaders](http://docs.unity3d.com/540/Documentation/Manual/SL-SubShader.html))。Unity在渲染着色器时，它会根据显卡去匹配合适的子着色器(subshaders)。

Shader "Structure Example" {

Properties { /\* ...shader properties... }

SubShader {

// ...subshader that requires fancy DX11 / GLES3.1 hardware...

}

SubShader {

// ...subshader that requires DX9 SM3 / GLES3 hardware...

}

SubShader {

// ...subshader that might look ugly but runs on anything :)

}

}

This system allows Unity to support all existing hardware and maximize the quality on each one. It does, however, result in some long shaders.

这样子就可以最大限度的发挥各种显卡的作用。但是，这样写会导致你的额着色器文件很长。

Inside each SubShader block you set the rendering state shared by all passes; and define rendering passes themselves. A complete list of available commands can be found in the [SubShader Reference](http://docs.unity3d.com/540/Documentation/Manual/SL-SubShader.html).

在每个子着色器(SubShader)块里边，你可以设置渲染状态共享所有通道和自定义渲染。 子着色器(SubShader)可用命令的完整列表，可以参考： SubShader Reference.

## **Passes 通道**

Each subshader is a collection of passes. For each pass, the object geometry is rendered, so there must be at least one pass. Our VertexLit shader has just one pass:

每个子着色器(subshader)都是一个通道集合。对于每一个pass，都要渲染几何对象，所以必须至少有一个pass。我们的VertexLit着色器只有一个pass

// ...snip...

Pass {

Material {

Diffuse [\_Color]

Ambient [\_Color]

Shininess [\_Shininess]

Specular [\_SpecColor]

Emission [\_Emission]

}

Lighting On

SeparateSpecular On

SetTexture [\_MainTex] {

constantColor [\_Color]

Combine texture \* primary DOUBLE, texture \* constant

}

}

// ...snip...

Any commands defined in a pass configures the graphics hardware to render the geometry in a specific way.

在pass内定义的任何一个命令都会在显卡上渲染指定的几何图形。In the example above we have a **[Material](http://docs.unity3d.com/540/Documentation/Manual/SL-Material.html)** block that binds our property values to the fixed function lighting material settings. The command ****Lighting On**** turns on the standard vertex lighting, and ****SeparateSpecular On**** enables the use of a separate color for the specular highlight.

在上面的例子中，我们有一个材质([Material](http://docs.unity3d.com/540/Documentation/Manual/SL-Material.html))块，定义了照明时所需要几项固定参数。 Lighting On 这个命令是打开顶点光照明设备，SeparateSpecular On是指允许使用一个单独的颜色作为镜面高光。

All of these command so far map very directly to the fixed function OpenGL/Direct3D hardware model. Consult [OpenGL red book](http://opengl.org/documentation/red_book) for more information on this.

到目前为止，所有这些命令都i支持OpenGL/Direct3D硬件本身可使用的固定功能。详情请参考：[OpenGL红皮书](https://www.opengl.org/sdk/)。

The next command, **[SetTexture](http://docs.unity3d.com/540/Documentation/Manual/SL-SetTexture.html)**, is very important. These commands define the textures we want to use and how to mix, combine and apply them in our rendering. ****SetTexture**** command is followed by the property name of the texture we would like to use (****\_MainTex**** here) This is followed by a ****combiner block**** that defines how the texture is applied. The commands in the combiner block are executed for each pixel that is rendered on screen.

Within this block we set a constant color value, namely the Color of the Material, ****\_Color****. We’ll use this constant color below.

下一个命令[SetTexture](http://docs.unity3d.com/540/Documentation/Manual/SL-SetTexture.html)是非常重要的。这个命令可以定义影像纹理如何混合、组合以及如何运用于我们的渲染里，SetTexture通常跟随于纹理的属性名称之后(我们在这里使用\_MainTex )，接下来的combiner block也是定义纹理的应用方式，这个combiner block中的命令会对在屏幕上显示的每一个像素执行对应的操作。

在这个块内我们设定了一个颜色值，并命名为\_Color，我们会在后面使用这个颜色。

In the next command we specify how to mix the texture with the color values. We do this with the ****Combine**** command that specifies how to blend the texture with another one or with a color. Generally it looks like this: ****Combine ColorPart, AlphaPart****

****在下个命令，我们指定如何混合纹理以及颜色值。用Combine命令来混合其他纹理或颜色，如下所示：Combine ColorPart, AlphaPart****

Here ****ColorPart**** and ****AlphaPart**** define blending of color (RGB) and alpha (A) components respectively. If ****AlphaPart**** is omitted, then it uses the same blending as ****ColorPart****.

在这里ColorPart与AlphaPart定义了混合的颜色(RGB)和alpha值(A)的信息，如果AlphaPart被省略了，那它将与ColorPart作混合。

In our VertexLit example: ****Combine texture \* primary DOUBLE, texture \* constant****

****在VertexLi的例子中：****

****Combine texture \* primary DOUBLE, texture \* constant****

Here ****texture**** is the color coming from the current texture (here ****\_MainTex****). It is multiplied (\*) with the ****primary**** vertex color. Primary color is the vertex lighting color, calculated from the Material values above. Finally, the result is multiplied by two to increase lighting intensity (****DOUBLE****). The alpha value (after the comma) is ****texture**** multiplied by ****constant**** value (set with ****constantColor**** above). Another often used combiner mode is called ****previous**** (not used in this shader). This is the result of any previous ****SetTexture**** step, and can be used to combine several textures and/or colors with each other.

这里的texture来自当前的纹理(\_MainTex)的颜色，它将与主要的顶点颜色相乘。主色为顶点光照颜色，它是由材质(Material)值计算出来的。最终Double的作用是这两个相乘后的两倍照明强度。 aplha值(在逗号以后)是由texture\*constant得来(constant与constantColor有关)。另一个常用的混合模式称为previous(在这个shader未使用)，这是所有previous SetTexture的结果，并且可以用来混合多种纹理和颜色。

## **Summary 总结**

Our VertexLit shader configures standard vertex lighting and sets up the texture combiners so that the rendered lighting intensity is doubled.

VertexLit着色器配备了标准的顶点光照，并设置了纹理，使呈现的照明强度增加了一倍。

We could put more passes into the shader, they would get rendered one after the other. For now, though, that is not nessesary as we have the desired effect. We only need one SubShader as we make no use of any advanced features - this particular shader will work on any graphics card that Unity supports.

我们可以在着色器中添加更多的通道集合。它们将一前一后的渲染出来。现在这个虽然不是我们所需要的理想效果。我们只需要一个子着色器，因为我们不使用任何高级功能，这个着色器支持任何显卡。

The VertexLit shader is one of the most basic shaders that we can think of. We did not use any hardware specific operations, nor did we utilize any of the more special and cool commands that ShaderLab and Cg/HLSL has to offer.

我们可以认为VertexLit着色器是一个最基本的着色器。我们没有使用任何硬件的具体操作，我们也没有利用任何特殊很酷的命令。这些语法都是ShaderLab和Cg提供的

In the [next chapter](http://docs.unity3d.com/540/Documentation/Manual/ShaderTut2.html) we’ll proceed by explaining how to write custom vertex & fragment programs using Cg/HLSL language.

在[下一章](http://docs.unity3d.com/540/Documentation/Manual/ShaderTut2.html)，我们将解释如何使用Cg/HLSL语言编写自定义的顶点(vertex)和片断程序( fragment programs)。