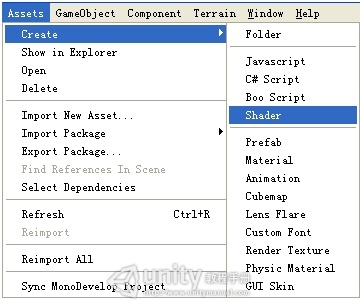
**Unity3D教程：教你如何利用Shader来进行3D角色的渲染**

Posted on 2013年05月09日 by U3d / [Unity3D 基础教程](http://www.unitymanual.com/category/manual/unity3d-%e5%9f%ba%e7%a1%80%e6%95%99%e7%a8%8b)/被围观 30 次

本文主要介绍一下如何利用Shader来渲染游戏中的3D角色，以及如何利用Unity提供的Surface Shader来书写自定义Shader。

**一、从Shader开始**

1、通过Assets->Create->Shader来创建一个默认的Shader，并取名“MyShader”。

[](http://www.unitymanual.com/wp-content/uploads/2013/05/bd56822dtdbb8267fed33690.jpg)

Unity3D教程：3D角色的渲染

2、将MyShader打开即可看见Unity默认的Shader代码

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | Shader "Custom/MyShader" { |
| 02 | Properties { |
| 03 | \_MainTex ("Base (RGB)", 2D) = "white" {} |
| 04 | } |
| 05 | SubShader { |
| 06 | Tags { "RenderType"="Opaque" } |
| 07 | LOD 200 |
| 08 | CGPROGRAM |
| 09 | #pragma surface surf Lambert |
| 10 | sampler2D \_MainTex; |
| 11 | **struct** Input { |
| 12 | float2 uv\_MainTex; |
| 13 | }; |
| 14 | **void** surf (Input **IN**, inout SurfaceOutput o) { |
| 15 | half4 c = tex2D (\_MainTex, **IN**.uv\_MainTex); |
| 16 | o.Albedo = c.rgb; |
| 17 | o.Alpha = c.a; |
| 18 | } |
| 19 | ENDCG |
| 20 | } |
| 21 | FallBack "Diffuse" |
| 22 | } |

3、将该Shader赋给一个角色，就可以看到该Shader所能表达出的Diffuse渲染效果。

[](http://www.unitymanual.com/wp-content/uploads/2013/05/bd56822dtdbb829d88e0d690.jpg)

Unity3D教程：3D角色的渲染

4、接来我们将以此默认Shader作为蓝本，编写出自定义的Shader。另外，该Shader所用到的参数，我们将在下一章节进行说明。

**二、实现多种自定义渲染效果**

**1、 BumpMap效果**

如果想实现Bump Map效果，可对上述的Shader做如下修改：

1.1 在属性Properties中加入：

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 1 | Properties { |
| 2 | \_MainTex ("Base (RGB)", 2D) = "white" {} |
| 3 | \_BumpMap("Bumpmap", 2D) = "bump" {} |
| 4 | } |

1.2 在SubShader的变量中也进行相应修改：

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 1 | sampler2D \_MainTex; |
| 2 | sampler2D \_BumpMap; |
| 3 | **struct** Input { |
| 4 | float2 uv\_MainTex; |
| 5 | float2 uv\_BumpMap; |
| 6 | }; |

1.3 最后修改surf函数，加入对Normal分量的计算：

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 1 | **void** surf (Input **IN**, inout SurfaceOutput o) { |
| 2 | "white-space: pre;"> half4 c = tex2D (\_MainTex, **IN**.uv\_MainTex); |
| 3 | o.Albedo = c.rgb; |
| 4 | o.Alpha = c.a; |
| 5 | o.Normal = UnpackNormal (tex2D (\_BumpMap, **IN**.uv\_BumpMap)); |
| 6 | } |

这样，角色的材质部分即可变为如下形式（暂定BumpMap的Shader名为“MyShader1”）：

[](http://www.unitymanual.com/wp-content/uploads/2013/05/bd56822dt7c5f37dc332a690.jpg)

Unity3D教程：3D角色的渲染

然后，根据Base图来创建其Normal Map图，并拖入到BumpMap中即可。BumpMap的效果显示如下：

[](http://www.unitymanual.com/wp-content/uploads/2013/05/12.jpg)

Unity3D教程：3D角色的渲染

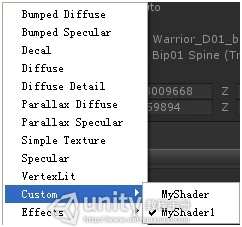
**说明：**

（1）首先是title的解释

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 1 | Shader&nbsp;"Custom/MyShader1" |

这种表示表明了该Shader在编辑器中的显示位置，例如我们可在如下地方找到该Shader。

[](http://www.unitymanual.com/wp-content/uploads/2013/05/bd56822dt7c5f383e677b690.jpg)

Unity3D教程：3D角色的渲染

（2）其次是Properties

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 1 | Properties { |
| 2 | \_MainTex ("Base (RGB)", 2D) = "white" {} |
| 3 | \_BumpMap("Bumpmap", 2D) = "bump" {} |
| 4 | } |

Properties可通过如下语义进行声明：

name ("displayname", property type) = default value

“name” 是与Shader脚本中对应的名字

“display name”是在材质视图中所显示的名字

“propertytype”是指该property的类型，一般可有如下几种类型：Range，Color，2D，Rect，Cube，Float和Vector

“defaultvalue”是指该property的默认值

这里需要注意的是，如果你在Properties中加入了新的属性，那么你需要在CGPROGRAM中的SubShader中加入同样名字的参数。

（3）接下来是“LOD”语义词的解释。

这里的“LOD”主要是指Shader的LOD程度，即对于超出该范围的物体将不再通过该Shader进行渲染，具体的Shader LOD说明可以参见：[Unity3D翻译——Shader Level of Detail](http://blog.csdn.net/amazonzx/article/details/7614399)

（4）我们在SubShader中还加入了

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 1 | sampler2D \_BumpMap; |
| 2 | float2 uv\_BumpMap; |

其中，\_BumpMap是为了关联Properties中的\_BumpMap属性。

而uv\_BumpMap，是为了获取BumpMap图中的uv坐标。

（5）最后，我们在surf函数中获取每个顶点的纹理信息以及法线信息，这些信息将被应用于接下来的Vertex Fragment和Pixel Fragment。

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 1 | **void** surf (Input **IN**, inout SurfaceOutput o) { |
| 2 | half4 c = tex2D (\_MainTex, **IN**.uv\_MainTex); |
| 3 | o.Albedo = c.rgb; |
| 4 | o.Alpha = c.a; |
| 5 | o.Normal = UnpackNormal (tex2D (\_BumpMap, **IN**.uv\_BumpMap)); |
| 6 | } |

其中，tex2D函数可以读取纹理\_MainTex中的IN.uv\_MainTex坐标位置的像素颜色值。

Albedo和Alpha分别获取该像素的RGB值和Alpha值，其中“Albedo”是一个漫反射参数，它表示一个表面的漫反射能力，即一个表面上出射光强与入射光强的比值。具体介绍可见：<http://en.wikipedia.org/wiki/Albedo>。

**2、 Blinn-Phong效果**

如果想实现Blinn-Phong效果，可对上述的Shader做如下修改：

2.1 在属性Properties中加入：

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 1 | \_AmbientColor ("Ambient Color", Color) = (0.1, 0.1, 0.1, 1.0) |
| 2 | \_SpecularColor ("Specular Color", Color) = (0.12, 0.31, 0.47, 1.0) |
| 3 | \_Glossiness ("Gloss", Range(1.0,512.0)) = 80.0 |

2.2 在SubShader的变量中也加入相应修改：

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 1 | fixed4 \_AmbientColor; |
| 2 | fixed4 \_SpecularColor; |
| 3 | half \_Glossiness; |

2.3 最后修改surf函数，进行如下修改：

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 1 | fixed4 c = tex2D (\_MainTex, **IN**.uv\_MainTex); |

这里将原有的half4替换为fixed4，这样做是为了提高渲染的性能，因为fixed的精度较之half要低，更高的精度意味着更大的计算量，而这里fixed的精度已经足够，所以使用fixed替代half4，从而来降低计算消耗，增加渲染性能。

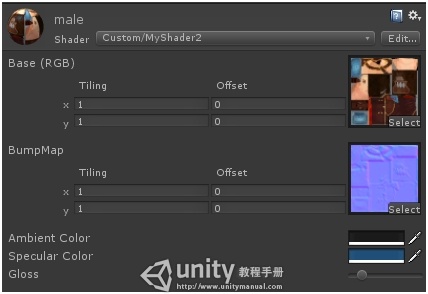
2.4 将“#pragma surface surf Lamber”改成“#pragma surfacesurf CustomBlinnPhong”，同时加入与其对应的LightingCustomBlinnPhong函数来计算顶点光照。

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | inline fixed4 LightingCustomBlinnPhong (SurfaceOutput s, fixed3 lightDir, fixed3 viewDir, **fixed** atten) |
| 02 | { |
| 03 | fixed3 ambient = s.Albedo \* \_AmbientColor.rgb; |
| 04 |  |
| 05 | **fixed** NdotL = saturate(dot (s.Normal, lightDir)); |
| 06 | fixed3 diffuse = s.Albedo \* \_LightColor0.rgb \* NdotL; |
| 07 |  |
| 08 | fixed3 h = normalize (lightDir + viewDir); |
| 09 | **float** nh = saturate(dot (s.Normal, h)); |
| 10 | **float** specPower = pow (nh, \_Glossiness); |
| 11 | fixed3 specular = \_LightColor0.rgb \* specPower \* \_SpecularColor.rgb; |
| 12 |  |
| 13 | fixed4 c; |
| 14 | c.rgb = (ambient + diffuse + specular) \* (atten \* 2); |
| 15 | c.a = s.Alpha + (\_LightColor0.a \* \_SpecularColor.a \* specPower \* atten); |
| 16 | **return** c; |
| 17 | } |

该函数的名称为什么不是“CustomBlinnPhong”呢？这是因为该函数虽然是由“#pragma surface surf CustomBlinnPhong”来调用，但是为了让该函数可以正常工作，我们需要在其名称前加入“Lighting”关键字，这样Unity才能识别出这是一个自定义的光照函数。

通过以上设置，角色的材质部分即可变为如下形式（暂定该Shader名为“MyShader2”）：

[](http://www.unitymanual.com/wp-content/uploads/2013/05/1346590072_3526.jpg)

Unity3D教程：3D角色的渲染

其显示效果如下：

[](http://www.unitymanual.com/wp-content/uploads/2013/05/24.jpg)

Unity3D教程：3D角色的渲染

**3、 边缘光照（Rim Light）和卡通渲染（Toon Shading）**

可以通过对上述Shader做以下改进，来达到这种效果：

3.1 在属性Properties中加入：

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 1 | \_RimColor ("Rim Color", Color) = (0.12, 0.31, 0.47, 1.0) |
| 2 | \_RimPower ("Rim Power", Range(0.5, 8.0)) = 3.0 |
| 3 | \_Ramp ("Shading Ramp", 2D) = "gray" {} |

3.2 在SubShader的变量中也加入相应修改：

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | sampler2D \_MainTex; |
| 02 | sampler2D \_BumpMap; |
| 03 | sampler2D \_Ramp; |
| 04 |  |
| 05 | fixed4 \_AmbientColor; |
| 06 | fixed4 \_SpecularColor; |
| 07 | half \_Glossiness; |
| 08 |  |
| 09 | fixed4 \_RimColor; |
| 10 | half \_RimPower; |
| 11 |  |
| 12 | **struct** Input { |
| 13 | float2 uv\_MainTex; |
| 14 | float2 uv\_BumpMap; |
| 15 | half3 viewDir; |
| 16 | }; |

3.3 修改surf函数，进行如下修改：

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 1 | **void** surf (Input **IN**, inout SurfaceOutput o) { |
| 2 | fixed4 c = tex2D (\_MainTex, **IN**.uv\_MainTex); |
| 3 | o.Albedo = c.rgb; |
| 4 | o.Alpha = c.a; |
| 5 | o.Normal = UnpackNormal (tex2D (\_BumpMap, **IN**.uv\_BumpMap)); |
| 6 | **fixed** rim = 1.0 - saturate (dot (normalize(**IN**.viewDir), o.Normal)); |
| 7 | o.Emission = (\_RimColor.rgb \* pow (rim, \_RimPower)); |
| 8 | } |

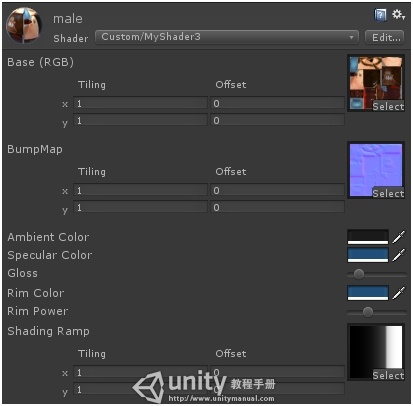
这里主要是用来计算边缘光照的，首先通过视线与法线的夹角来找到模型的边缘，然后再根据距离的远近来控制发射光的强度。

3.4 将“#pragma surface surf CustomBlinnPhong”改成“#pragma surfacesurf CustomBlinnPhong exclude\_path:prepass”，同时在LightingCustomBlinnPhong函数来修改漫反射光的计算，来达到卡通渲染的效果。

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 1 | **fixed** NdotL = saturate(dot (s.Normal, lightDir)); |
| 2 | **fixed** diff = NdotL \* 0.5 + 0.5; |
| 3 | fixed3 ramp = tex2D (\_Ramp, float2(diff, diff)).rgb; |
| 4 | **fixed** diffuse = s.Albedo \* LightColor0.rgb \* ramp; |

通过以上设置，角色的材质部分即可变为如下形式（暂定该Shader名为“MyShader3”）：

[](http://www.unitymanual.com/wp-content/uploads/2013/05/33.jpg)

Unity3D教程：3D角色的渲染

其显示效果如下：

[](http://www.unitymanual.com/wp-content/uploads/2013/05/41.jpg)

Unity3D教程：3D角色的渲染

可以看出边缘光照的效果，同时还可以看出明显的明暗变化的卡通渲染效果。

**三、 小结**

综上所述，本文已经给出了人物的几种基本渲染方法及其Shader实现，在这里我并没有去分析每种渲染效果的原理，而仅是从实际出发，直接给出对应的简单实现方法。如果想要对光照模型进行深入理解，可以Google搜索其原理进行了解。最后，给出各种渲染方法的对比图，显示如下：

[](http://www.unitymanual.com/wp-content/uploads/2013/05/51.jpg)

Unity3D教程：3D角色的渲染