

随机阅读

分配一个合适的着色器，通常这个着色器不需要访问游戏对象的材质，因为它把材质应用到其他对象上，因此这个着色器不需要访问游戏对象的纹理，同时这个着色器也不需要获得光源的任何信息，然而它可以访问到游戏对象的顶点属性以及它自身的着色器属性。

着色器给对象添加光可以用来在这个对象上投影任何的影像，这有点类似于投影仪或者电影放映机，因此我们可以使用类似于《[Cookies](#)》章节中的聚光灯 cookies 纹理图像，但有所不同的是纹理图像的 RGB 颜色也可以被添加，这样做可以用来实现彩色投影，为此我们可以通过将片段颜色设置为纹理图像的 RGBA 颜色并使用混合公式


```
1 | Blend One One
```

这个公式只在帧缓冲的颜色中添加了片段颜色（根据纹理图像的不同，我们可以使用 Blend SrcAlpha One 来移除透明度为 0 的颜色）。


还有一点与聚光灯的 Cookie 不同，我们应该使用 Unity 指定的 uniform `_Projector` 矩阵代替 `_LightMatrix0` 矩阵来把位置从对象空间转换到投影空间，然而投影空间中的坐标与光空间的坐标非常相似——除了把 x 和 y 坐标限制在正确的范围内，因此我们不需要再给 x 和 y 坐标加上 0.5 了，但是还需要除以 w 坐标（对于投影纹理贴图总是如此），因此我们需要把 x 和 y 坐标除以 w 坐标或者使用 `tex2Dproj`。

行 `ZWrite Off` 确保我们不改变深度缓存，因为我们只是给已栅格化的网格添加光，`Offset -1, -1` 用来改变在网格前面添加光的深度，这样确保我们现在进行栅格化没有网格阻塞，着色器的代码如下：


```
01 | Shader "Cg projector shader for adding light"
02 | {
03 |     Properties
04 |     {
05 |         _ShadowTex ("Projected Image", 2D) = "white" {}
06 |     }
07 |     SubShader
08 |     {
09 |         Pass
10 |         {
11 |             Blend One One
12 |             // add color of _ShadowTex to the color in the framebuffer
13 |             ZWrite Off // don't change depths
14 |             Offset -1, -1 // avoid depth fighting
15 |
16 |             CGPROGRAM
17 |
18 |             #pragma vertex vert
19 |             #pragma fragment frag
20 |
21 |             // User-specified properties
22 |             uniform sampler2D _ShadowTex;
23 |
24 |             // Projector-specific uniforms
25 |             uniform float4x4 _Projector; // transformation matrix
26 |             // from object space to projector space
27 |
28 |             struct vertexInput
29 |             {
30 |                 float4 vertex : POSITION;
31 |                 float3 normal : NORMAL;
32 |             };
33 |             struct vertexOutput
34 |             {
35 |                 float4 pos : SV_POSITION;
36 |                 float4 posProj : TEXCOORD0;
37 |                 // position in projector space
38 |             };
39 |
40 |             vertexOutput vert(vertexInput input)
41 |             {
42 |                 vertexOutput output;
43 |
44 |                 output.posProj = mul(_Projector, input.vertex);
45 |                 output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
46 |                 return output;
47 |             }
48 |
49 |             float4 frag(vertexOutput input) : COLOR
50 |             {
51 |                 if (input.posProj.w > 0.0) // in front of projector?
52 |                 {
53 |                     return tex2D(_ShadowTex, input.posProj.xy / input.posProj.w);
54 |                     // alternatively: return tex2Dproj(
55 |                     // _ShadowTex, input.posProj);
56 |                 }
```

暂无图片


【原创】Shader 内置 Shader 之 Bumped Diffuse 学习 - 1707 次阅读

暂无图片


【翻译】第九章节：漫反射（关于每顶点漫反射和多光源漫反射） - 1946 次阅读

暂无图片

【翻译】第二十章节：凹凸表面光照（关于法线贴图） - 1920 次阅读

暂无图片

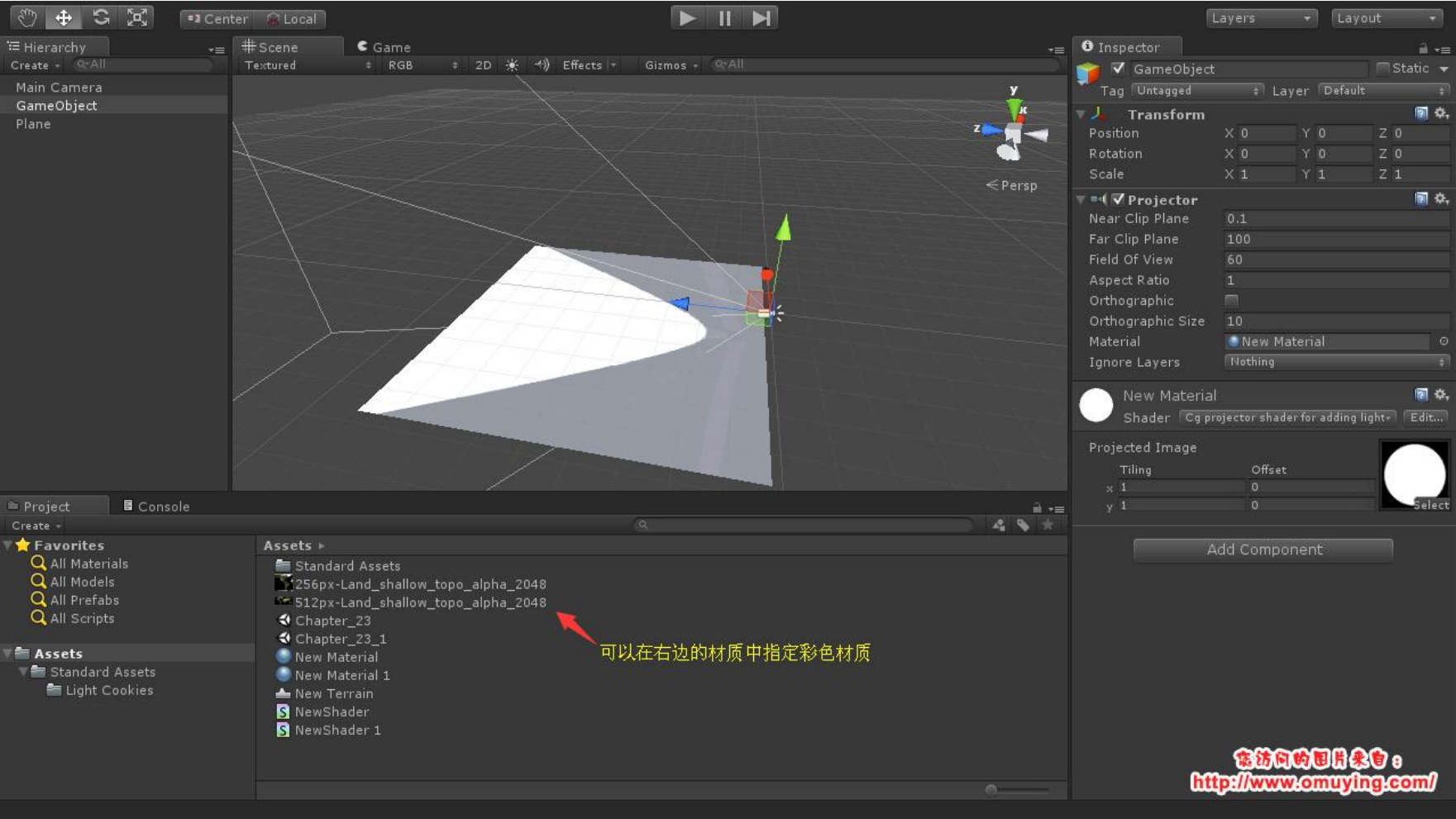
【翻译】第二十四章节：表面反射（关于反射贴图） - 1441 次阅读

暂无图片

【翻译】第十三章节：双面平滑表面（关于双面每像素光照） - 1100 次阅读


```
57         else // behind projector
58         {
59             return float4(0.0, 0.0, 0.0, 0.0);
60         }
61     }
62
63     ENDCG
64 }
65 }
66 // The definition of a fallback shader should be commented out
67 // during development:
68 // Fallback "Projector/Light"
69 }
```

注意，我们必须检测 w 是正值（即在片段在投影的前面不是后面），如果不检测，投影会在物体的背后添加光，此外对于换行模式纹理图像必须是正方形，另外着色器纹理图像命名为 `_ShadowTex` 是为了与内置投影着色器相兼容，这个着色器的效果如图：



衰减颜色实现投影

通过调节颜色来创建投影的步骤与上面基本相同，唯一的区别是着色器的代码，下面的例子通过衰减颜色来添加阴影，特别是地板的颜色，注意在实际应用中，投影对象的颜色不应该被衰减，这可以通过为投影对象指定一个层（在 Inspector 视图中）然后在 Inspector 视图投影的选项中设置忽略这个层来实现。

为了给影子一定的形状，我们使用纹理图像的 alpha 分量来确定阴影的黑暗程度（我们可以使用标准资源库中的光 Cookie 纹理），为了衰减帧缓冲中的颜色，我们需与 1 减 alpha 相乘（即因子 alpha 为 0 时它等于 1），因此混合的公式为：

```
1 Blend Zero OneMinusSrcAlpha
```

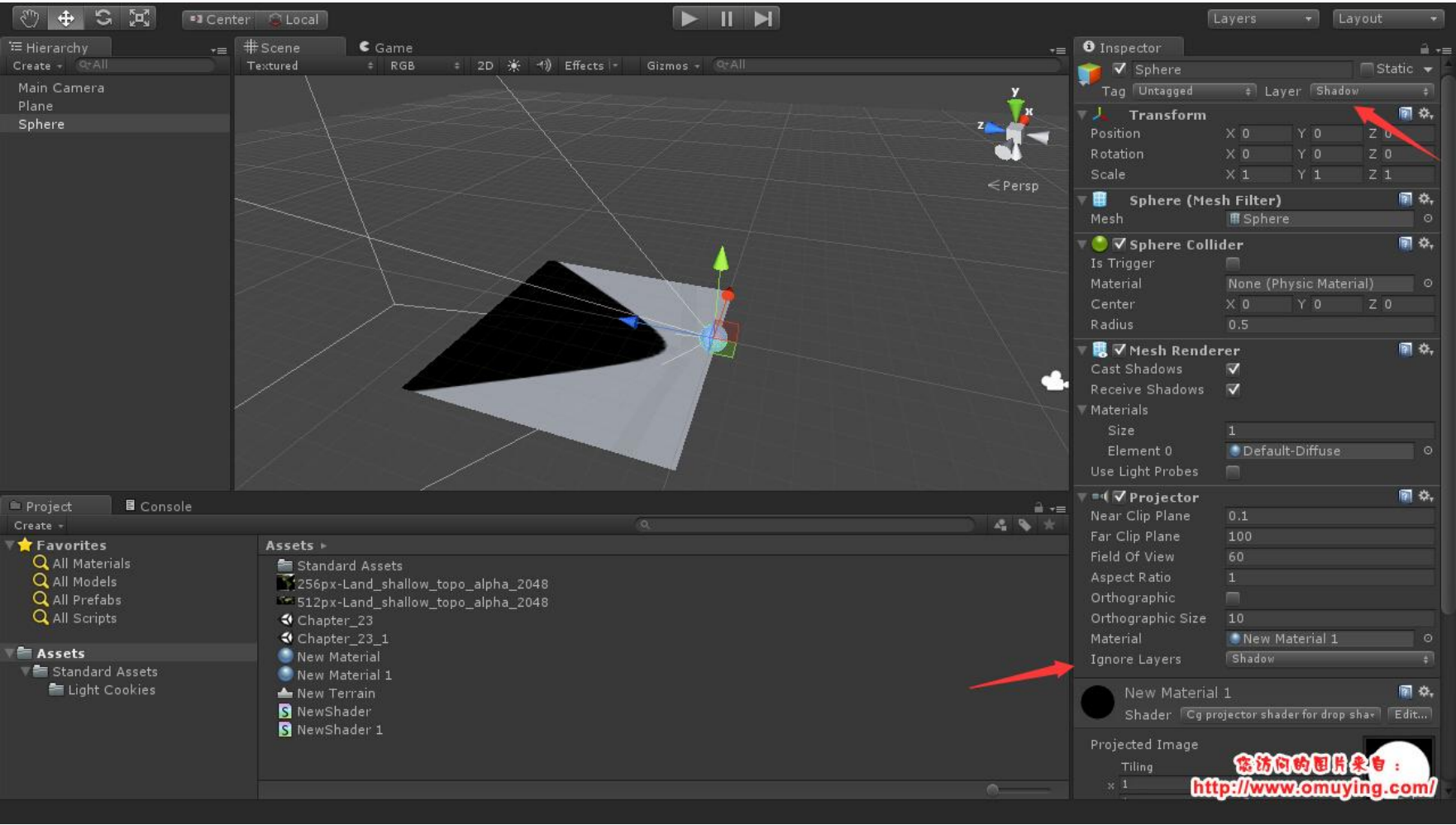
零表明我们不添加任何光线，即使阴影很深也没有光线被添加，相反在片段着色器中 alpha 分量应该被减小，这个可以通过与一个小于 1 的数相乘来实现，在帧缓冲中独立调节颜色分量，我们需要使用混合 `Blend Zero SrcColor` 或者 `Blend Zero OneMinusSrcColor`。

对于添加光使用不同的混合公式实际上只是为了对着色器效果进行比较，着色器的代码如下：

```
01 Shader "Cg projector shader for drop shadows"
02 {
03     Properties
04     {
05         _ShadowTex ("Projected Image", 2D) = "white" {}
06     }
07     SubShader
08     {
09         Pass
10         {
11             Blend Zero OneMinusSrcColor // attenuate color in framebuffer
12             // by 1 minus alpha of _ShadowTex
13             ZWrite Off // don't change depths
14             Offset -1, -1 // avoid depth fighting
15
16             CGPROGRAM
17
18             #pragma vertex vert
```

```
19 #pragma fragment frag
20
21 // User-specified properties
22 uniform sampler2D _ShadowTex;
23
24 // Projector-specific uniforms
25 uniform float4x4 _Projector; // transformation matrix
26 // from object space to projector space
27
28 struct vertexInput
29 {
30     float4 vertex : POSITION;
31     float3 normal : NORMAL;
32 };
33 struct vertexOutput
34 {
35     float4 pos : SV_POSITION;
36     float4 posProj : TEXCOORD0;
37     // position in projector space
38 };
39
40 vertexOutput vert(vertexInput input)
41 {
42     vertexOutput output;
43
44     output.posProj = mul(_Projector, input.vertex);
45     output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
46     return output;
47 }
48
49 float4 frag(vertexOutput input) : COLOR
50 {
51     if (input.posProj.w > 0.0) // in front of projector?
52     {
53         return tex2D(_ShadowTex, input.posProj.xy / input.posProj.w);
54         // alternatively: return tex2Dproj(
55         // _ShadowTex, input.posProj);
56     }
57     else // behind projector
58     {
59         return float4(0.0, 0.0, 0.0, 0.0);
60     }
61 }
62
63 ENDCG
64 }
65 }
66 // The definition of a fallback shader should be commented out
67 // during development:
68 // Fallback "Projector/Light"
69 }
```

这个着色器的效果如图：



恭喜你，在本章节中，你应该了解：

- 1、Unity 投影的工作方式。
- 2、如何通过添加光实现投影。
- 3、如果通过衰减对象颜色实现投影。

前一篇：[第二十二章：Cookies（关于投影纹理贴图塑造光的形状）](#)

后一篇：[第二十四章：表面反射（关于反射贴图）](#)



赞

6 人



打酱油

0 人



呵呵

0 人



鄙视

0 人



正能量

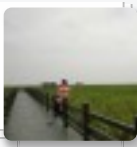
0 人



0

1 条评论

最新 最早 最热



谭陶亮

写的很好

2015年7月24日 [← 回复](#) [♥ 顶](#) [→ 转发](#)

社交帐号登录:

微信

微博

QQ

人人

[更多»](#)



说点什么吧...

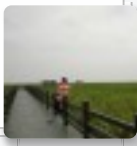


发布

最终幻想正在使用多说

1 条评论

最新 最早 最热



谭陶亮

写的很好

2015年7月24日 [← 回复](#) [♥ 顶](#) [→ 转发](#)

社交帐号登录:

微信

微博

QQ

人人

[更多»](#)



说点什么吧...



发布

最终幻想正在使用多说