

随机阅读

```
27 float4x4 modelMatrixInverse = _World2Object;
28 // multiplication with unity_Scale.w is unnecessary
29 // because we normalize transformed vectors
30
31 output.posWorld = mul(modelMatrix, input.vertex);
32 output.posLight = mul(_LightMatrix0, output.posWorld);
33 output.normalDir = normalize(mul(float4(input.normal, 0.0),
modelMatrixInverse).xyz);
34 output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
35 return output;
36 }
```

除了声明 uniform \_LightMatrix0 和定义一个新的输出参数 posLight 以及通过指令计算 posLight 之外，其他的代码与《平滑镜面高光》章节中的相同。

### 定向光源中的 Cookie

对于定向光源中的 Cookie，我们可以只使用 posLight 中的 x 和 y 光坐标在 Cookie 纹理 \_LightTexture0 中查找，这个纹理查找应该在片段着色器中执行，之后我们将得到的纹理 alpha 分量与原来的光照计算相乘，代码如下：

```
01 // compute diffuseReflection and specularReflection
02
03 float cookieAttenuation = 1.0;
04 if (0.0 == _WorldSpaceLightPos0.w) // directional light?
05 {
06     cookieAttenuation = tex2D(_LightTexture0, input.posLight.xy).a;
07 }
08 // compute cookieAttenuation for spotlights here
09
10 return float4(cookieAttenuation * (diffuseReflection + specularReflection),
1.0);
```

### 聚光灯中的 Cookie

对于聚光灯，posLight 中的光坐标 x 和 y 必须除以 w 光坐标，投影纹理贴图的特征就是除以 w 光坐标，并且这样做也与相机的透视除法相对应，详情可以查看《Vertex Transformations》章节，我们必须在除法之后对 x 和 y 坐标都加上 0.5，这样我们才可以在 Unity 声明的矩阵 \_LightMatrix0 中查找纹理，代码如下：

```
1 cookieAttenuation = tex2D(_LightTexture0, input.posLight.xy / input.posLight.w
+ float2(0.5, 0.5)).a;
```

为了在一些图形处理器中更加高效，我们可以使用内置的函数 tex2Dproj，它使用三个 float3 类型的纹理坐标，并且在纹理查找之前首先用前两个坐标除以第三个坐标，但这样有一个问题，因为我们必须在除以 posLight.w 之后再加上 0.5，而 tex2Dproj 内部不允许我们在除以第三个坐标之后加上任何数据，解决的办法是在除以 posLight.w 之前先加上 0.5 \* input.posLight.w，这样做等同于除以 posLight.w 之后再加上 0.5，代码如下：


```
1 float3 textureCoords = float3( input.posLight.x + 0.5 * input.posLight.w,
input.posLight.y + 0.5 * input.posLight.w, input.posLight.w);
2 cookieAttenuation = tex2Dproj(_LightTexture0, textureCoords).a;
```

注意，在定向光源中，纹理查找也可以用 tex2Dproj 来实现，我们可以设置 textureCoords 的值等于 float3(input.posLight.xy, 1.0)，这样做的好处就是我们可以在定向光源与聚光灯中只使用一种纹理查找方式，并且这样做在一些图形处理器中更加高效。


### 完成着色器代码

完成着色器代码，我们使用《平滑镜面高光》章节中 ForwardBase pass 的简化版本，因为 Unity 在 ForwardBase pass 中只使用没有 Cookie 的定向光源，而带有 Cookie 的光源在 ForwardAdd pass 中处理，对于点光源，我们忽略 Cookies，为此我们设置 \_LightMatrix0[3][3] 的值是 1.0，聚光灯总是有一个 Cookie 纹理：如果用户没有指定 Cookie 纹理，Unity 提供了一个 Cookie 纹理来生成聚光灯的形状，因此它总能确保有 Cookie 纹理。定向光源并不总是有 Cookie，然而如果一个定向光源不带 Cookie，那么它会在 ForwardBase pass 处理，因此除非有多个不带 Cookie 的定向光源，否则我们假设在 ForwardAdd pass 中的定向光源都带有 Cookie，所以完整的着色器代码如下：


```
001 Shader "Cg per-pixel lighting with cookies"
```

暂无图片


【翻译】第二十一章节：凹凸表面投影（关于视差贴图） - 1460 次阅读

暂无图片


【翻译】第十章节：镜面高光（关于每顶点光照） - 2007 次阅读

暂无图片

【翻译】第十四章节：多个灯（关于在一个 pass 中遍历处理多个光源） - 1932 次阅读

暂无图片

【翻译】第七章节：顺序无关的透明度（关于顺序无关的混合） - 1978 次阅读

暂无图片

【转载】Shader 变灰效果 - 2552 次阅读



```

002 {
003     Properties
004     {
005         _Color ("Diffuse Material Color", Color) = (1,1,1,1)
006         _SpecColor ("Specular Material Color", Color) = (1,1,1,1)
007         _Shininess ("Shininess", Float) = 10
008     }
009     SubShader
010     {
011         Pass
012         {
013             Tags { "LightMode" = "ForwardBase" } // pass for ambient light
014             // and first directional light source without cookie
015
016             CGPROGRAM
017
018             #pragma vertex vert
019             #pragma fragment frag
020
021             #include "UnityCG.cginc"
022             uniform float4 _LightColor0;
023             // color of light source (from "Lighting.cginc")
024
025             // User-specified properties
026             uniform float4 _Color;
027             uniform float4 _SpecColor;
028             uniform float _Shininess;
029
030             struct vertexInput
031             {
032                 float4 vertex : POSITION;
033                 float3 normal : NORMAL;
034             };
035             struct vertexOutput
036             {
037                 float4 pos : SV_POSITION;
038                 float4 posWorld : TEXCOORD0;
039                 float3 normalDir : TEXCOORD1;
040             };
041
042             vertexOutput vert(vertexInput input)
043             {
044                 vertexOutput output;
045
046                 float4x4 modelMatrix = _Object2World;
047                 float4x4 modelMatrixInverse = _World2Object;
048                 // multiplication with unity_Scale.w is unnecessary
049                 // because we normalize transformed vectors
050
051                 output.posWorld = mul(modelMatrix, input.vertex);
052                 output.normalDir = normalize(mul(float4(input.normal, 0.0),
modelMatrixInverse).xyz);
053                 output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
054                 return output;
055             }
056
057             float4 frag(vertexOutput input) : COLOR
058             {
059                 float3 normalDirection = normalize(input.normalDir);
060
061                 float3 viewDirection = normalize(_WorldSpaceCameraPos -
input.posWorld.xyz);
062                 float3 lightDirection = normalize(_WorldSpaceLightPos0.xyz);
063
064                 float3 ambientLighting = UNITY_LIGHTMODEL_AMBIENT.rgb *
_Color.rgb;
065
066                 float3 diffuseReflection = _LightColor0.rgb * _Color.rgb *
max(0.0, dot(normalDirection, lightDirection));
067
068                 float3 specularReflection;
069                 if (dot(normalDirection, lightDirection) < 0.0) // light source on
the wrong side?
070                 {
071                     specularReflection = float3(0.0, 0.0, 0.0);
072                     // no specular reflection
073                 }
074                 else // light source on the right side
075                 {
076                     specularReflection = _LightColor0.rgb * _SpecColor.rgb *
pow(max(0.0, dot(reflect(-lightDirection, normalDirection), viewDirection)),
_Shininess);
077                 }
078
079                 return float4(ambientLighting + diffuseReflection +
specularReflection, 1.0);
080             }
081
082             ENDCG
083         }
084
085         Pass
086         {

```

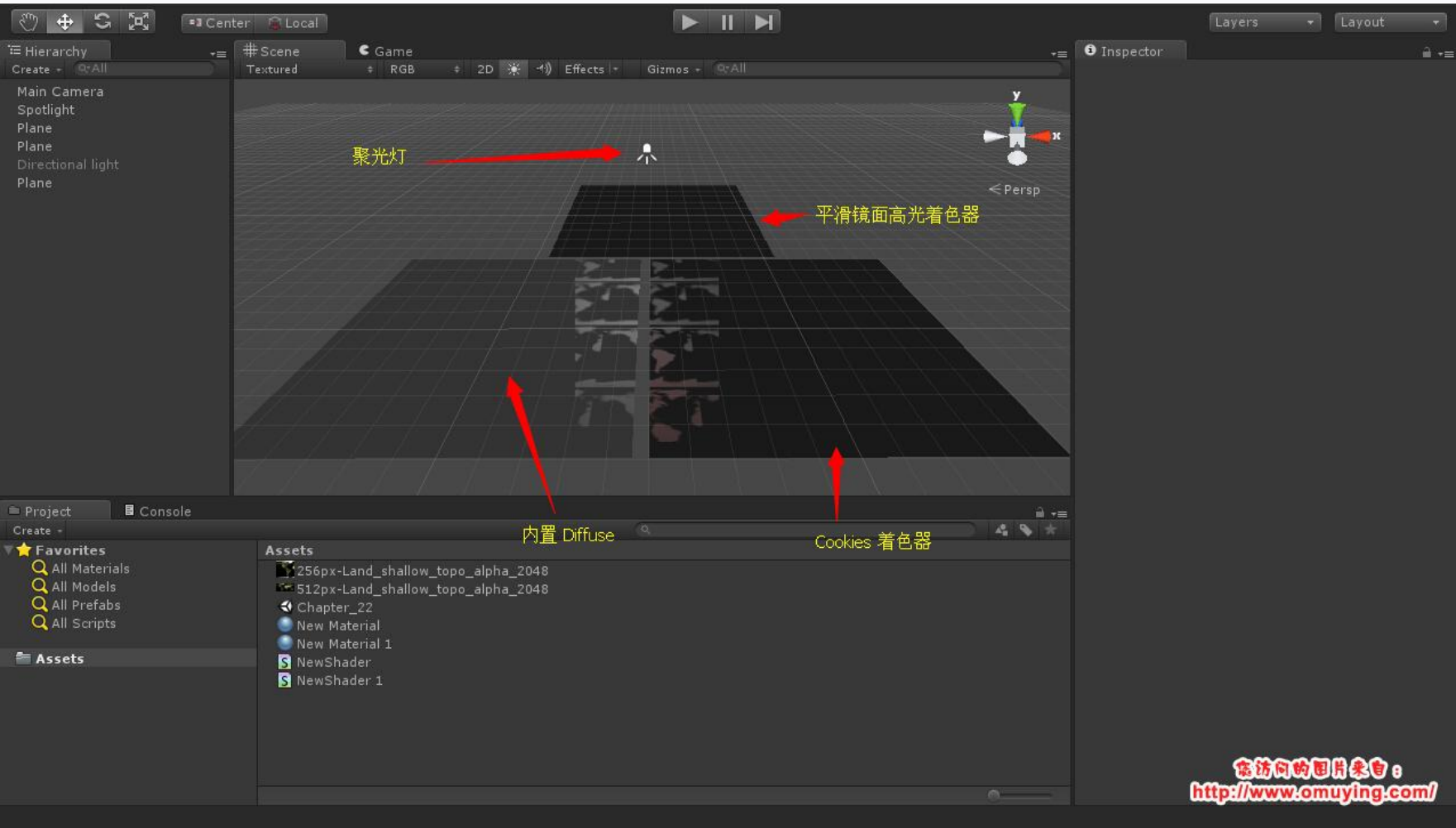
```

087     Tags { "LightMode" = "ForwardAdd" }
088     // pass for additional light sources
089     Blend One One // additive blending
090
091     CGPROGRAM
092
093     #pragma vertex vert
094     #pragma fragment frag
095
096     #include "UnityCG.cginc"
097     uniform float4 _LightColor0;
098     // color of light source (from "Lighting.cginc")
099     uniform float4x4 _LightMatrix0; // transformation
100     // from world to light space (from Autolight.cginc)
101     uniform sampler2D _LightTexture0;
102     // cookie alpha texture map (from Autolight.cginc)
103
104     // User-specified properties
105     uniform float4 _Color;
106     uniform float4 _SpecColor;
107     uniform float _Shininess;
108
109     struct vertexInput
110     {
111         float4 vertex : POSITION;
112         float3 normal : NORMAL;
113     };
114     struct vertexOutput
115     {
116         float4 pos : SV_POSITION;
117         float4 posWorld : TEXCOORD0;
118         // position of the vertex (and fragment) in world space
119         float4 posLight : TEXCOORD1;
120         // position of the vertex (and fragment) in light space
121         float3 normalDir : TEXCOORD2;
122         // surface normal vector in world space
123     };
124
125     vertexOutput vert(vertexInput input)
126     {
127         vertexOutput output;
128
129         float4x4 modelMatrix = _Object2World;
130         float4x4 modelMatrixInverse = _World2Object;
131         // multiplication with unity_Scale.w is unnecessary
132         // because we normalize transformed vectors
133
134         output.posWorld = mul(modelMatrix, input.vertex);
135         output.posLight = mul(_LightMatrix0, output.posWorld);
136         output.normalDir = normalize(mul(float4(input.normal, 0.0),
modelMatrixInverse).xyz);
137         output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
138         return output;
139     }
140
141     float4 frag(vertexOutput input) : COLOR
142     {
143         float3 normalDirection = normalize(input.normalDir);
144
145         float3 viewDirection = normalize(_WorldSpaceCameraPos -
input.posWorld.xyz);
146         float3 lightDirection;
147         float attenuation;
148
149         if (0.0 == _WorldSpaceLightPos0.w) // directional light?
150         {
151             attenuation = 1.0; // no attenuation
152             lightDirection = normalize(_WorldSpaceLightPos0.xyz);
153         }
154         else // point or spot light
155         {
156             float3 vertexToLightSource = _WorldSpaceLightPos0.xyz -
input.posWorld.xyz;
157             float distance = length(vertexToLightSource);
158             attenuation = 1.0 / distance; // linear attenuation
159             lightDirection = normalize(vertexToLightSource);
160         }
161
162         float3 diffuseReflection = attenuation * _LightColor0.rgb *
_Color.rgb * max(0.0, dot(normalDirection, lightDirection));
163
164         float3 specularReflection;
165         if (dot(normalDirection, lightDirection) < 0.0) // light source on
the wrong side?
166         {
167             specularReflection = float3(0.0, 0.0, 0.0);
168             // no specular reflection
169         }
170         else // light source on the right side
171         {
172             specularReflection = attenuation * _LightColor0.rgb *
_SpecColor.rgb * pow(max(0.0, dot(reflect(-lightDirection, normalDirection),
viewDirection)), _Shininess);

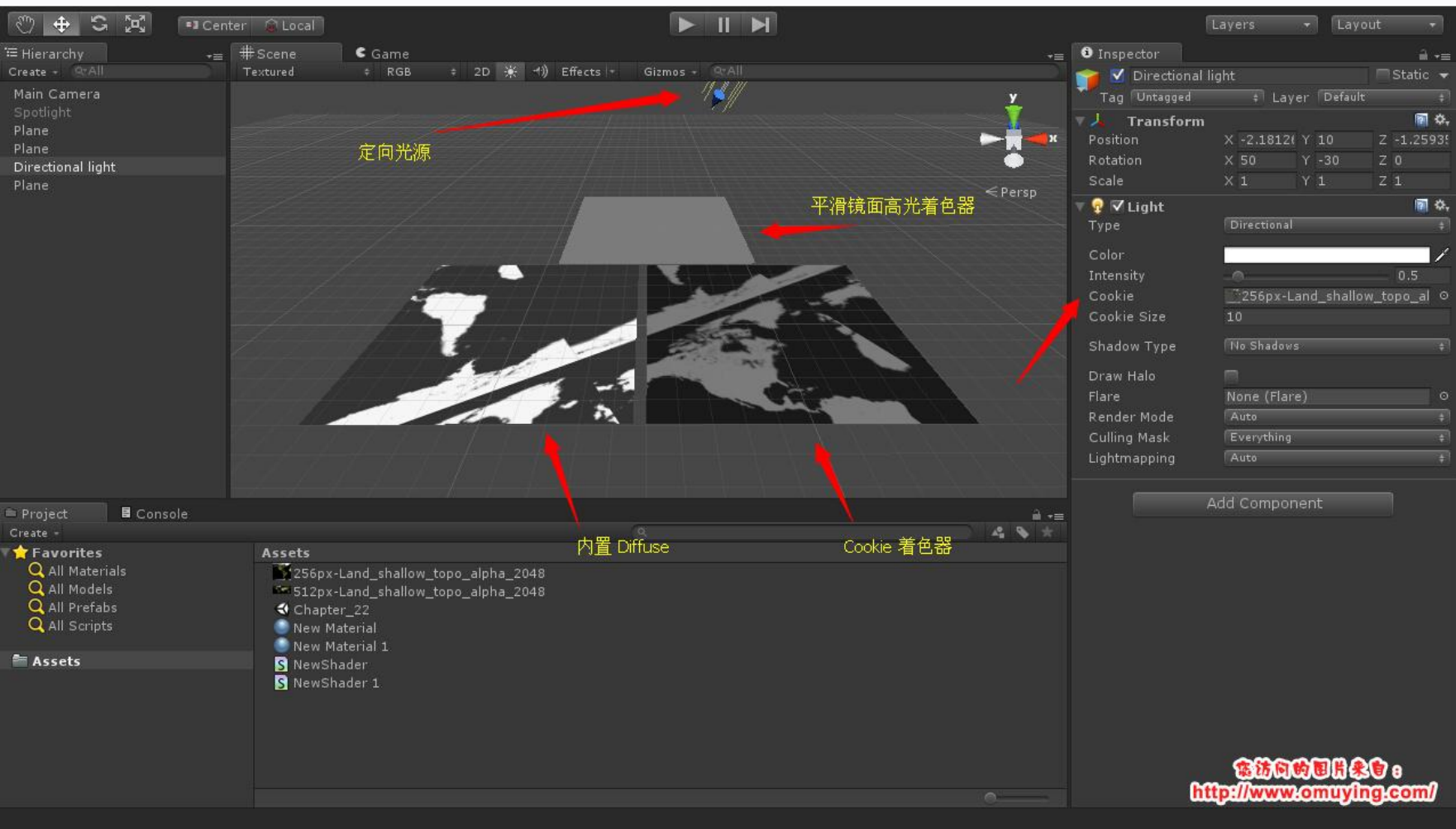
```

```
173 }
174
175 float cookieAttenuation = 1.0;
176 if (0.0 == _WorldSpaceLightPos0.w) // directional light?
177 {
178     cookieAttenuation = tex2D(_LightTexture0, input.posLight.xy).a;
179 }
180 else if (1.0 != _LightMatrix0[3][3])
181     // spotlight (i.e. not a point light)?
182 {
183     cookieAttenuation = tex2D(_LightTexture0, input.posLight.xy /
input.posLight.w + float2(0.5, 0.5)).a;
184 }
185
186 return float4(cookieAttenuation * (diffuseReflection +
specularReflection), 1.0);
187 }
188
189 ENDCG
190 }
191 }
192 // The definition of a fallback shader should be commented out
193 // during development:
194 // Fallback "Specular"
195 }
```

注意点光源中的 Cookie 使用的是一个立方体贴图，这个纹理贴图在《Reflecting Surfaces》章节中讨论，我们在场景中建立三个平面，并且这三个平面的材质分别使用内置 Diffuse 着色器，本章节的 Cookie 着色器以及平面镜面高光中的着色器，聚光灯下的效果如图：



定向光源中的效果如图：



恭喜你，在本章节中你应该了解：



- 1、如何在定义光源中实现 Cookie。
- 2、如何在聚光灯中实现 Cookie。
- 3、如何针对不同光源实现不同的着色器。

资源下载地址：[点击下载](#)，共下载 24 次。

前一篇：[第二十一章：凹凸表面投影（关于视差贴图）](#)

后一篇：[第二十三章：投影（关于使用投影纹理贴图实现投影）](#)



赞

3 人



打酱油

0 人



呵呵

0 人



鄙视

0 人



正能量

0 人



0

0条评论

最新 最早 最热

还没有评论，沙发等你来抢

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

0条评论

最新 最早 最热

还没有评论，沙发等你来抢

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