

## 【翻译】第九章节：漫反射（关于每顶点漫反射和多光源漫反射）

2014-11-30 08:43:00    1947 人阅读    [Unity3D](#)   [cg](#)   [漫反射](#)

A<sup>-</sup>   A<sup>+</sup>

文章内容	例子源码	网友评论	最后编辑：2014-12-21 18:15:30
本文永久地址： <a href="http://www.omuying.com/article/97.aspx">http://www.omuying.com/article/97.aspx</a> ，【 <a href="#">文章转载请注明出处！</a> 】			

原文链接：[http://en.wikibooks.org/wiki/Cg\\_Programming/Unity/Diffuse\\_Reflection](http://en.wikibooks.org/wiki/Cg_Programming/Unity/Diffuse_Reflection)

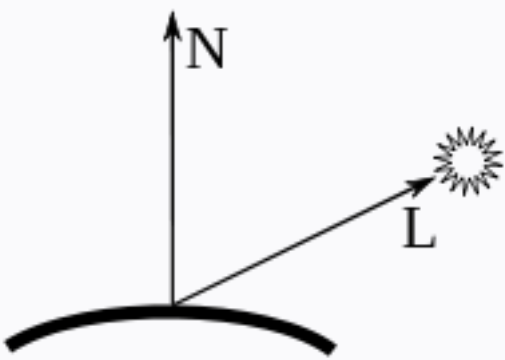
这个教程介绍每顶点漫反射（per-vertex diffuse reflection）。

本篇是系列教程中第一篇介绍 Unity 基本照明的教程，本教程我们开始在方向光、点光源、多光源（多个 pass）中的使用漫反射。另外教程还涵盖了镜面反射，每像素光照和双面光。

### 漫反射

月球表面就属于漫反射（也叫 Lambertian 反射），就是所有方向的光被反射但没有高光（镜面），这有点像白垩土（chalk）或者亚光纸（matte paper），他们的表面都比较暗淡并且无光泽。

完美条件下的漫反射，观察到的反射光强度取决于表面法线向量和入射光射线之间角度的余弦值，如下图所示：



根据表面点的向量（normalized）来计算光照：规范化的表面法线向量 N 是正交于表面的，L 是规范化的点光源（表面点到光源）方向。为观察漫反射光  $I_{\text{diffuse}}$ ，我们可以根据表面法线向量 N（normalized）和光线方向 L（normalized）之间角度的余弦值来求得 N·L 的点积，因为任何 a 和 b 向量的点积（a·b）是：

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| |\mathbf{b}| \cos \angle(\mathbf{a}, \mathbf{b})$$

因为 a 和 b 都是规范化的向量，所以 |a| 和 |b| 的值都是 1。

如果点积（N·L）的结果是负值时，光在表面的另一边，因此我们应该把反射设置为 0。通过 max(0, N·L)，我们能保证点积的结果始终大于等于 0，此外，漫反射的反射光取决于入射光强度  $I_{\text{incoming}}$  和材质常数  $k_{\text{diffuse}}$ ，对于黑色表面，材质常数  $k_{\text{diffuse}}$  的值是 0，对于白色表面，材质常数  $k_{\text{diffuse}}$  的值是 1。所以漫反射强度的公式为：

$$I_{\text{diffuse}} = I_{\text{incoming}} k_{\text{diffuse}} \max(0, \mathbf{N} \cdot \mathbf{L})$$

同样，如果变量  $I_{\text{diffuse}}$   $I_{\text{incoming}}$  和  $k_{\text{diffuse}}$  表示颜色向量的各个分量（红、绿、蓝）分别相乘，那么这个公式也就适用于彩色光，实际上我们也在着色器使用他们。

### 单方向光着色器代码

如果我们只有一个方向光，那么在着色器中计算  $I_{\text{diffuse}}$  值的代码会相对较少，另外我们还需要解决几个问题：



游戏开发学习



比基尼除毛

伪装微型摄像机

unity3d摄像机

微型摄像机

无线监控摄像机

unity3d移动

室内设计师

针孔摄像头


比基尼除毛

最新文章



暂无图片

【原创】C# 基础之 Lambda表达式 - 907 次阅读



暂无图片

【原创】C#基础之 IEnumerable和 IEnumerator - 792 次阅读




暂无图片

【原创】C#基础之事件 - 886 次阅读



暂无图片

【原创】C#基础之委托 - 912 次阅读



暂无图片

【原创】C#基础之委托的使用 - 856 次阅读



猎头公司排名



高速摄像机

伪装微型摄像机

上海猎头公司

超高速摄像机

防爆摄像机

微型摄像机

猎头公司

随机阅读

- 1、这个公式是在顶点着色器还是片段着色器实现？在这里，我们会在顶点着色器中实现，在《[Smooth Specular Highlights](#)》章节中，我们会在片段着色器中实现。
- 2、在哪个坐标系中实现呢？在这里我们使用 Unity 的世界坐标系（因为 Unity 提供了光线在世界空间中的方向）。
- 3、我们从哪获得参数呢？这个问题的答案有些长。

我们在着色器中添加一个属性来让用户指定漫反射的材质颜色  $k_{\text{diffuse}}$ ，我们可以根据 Unity 指定的 uniform `_WorldSpaceLightPos0` 获得光源的方向以及 uniform `_LightColor0`，获得光的颜色  $I_{\text{incoming}}$ 。就像《[世界空间中的着色器](#)》章节中的那样，我们必须给着色器的 pass 添加标记 `{"LightMode" = "ForwardBase"}` 来确保 uniforms 有正确的值，我们在顶点输入参数中通过语义词 `NORMAL` 获得对象坐标系中的表面法线向量，在公式中，我们需要世界坐标，参考《[轮廓加强](#)》章节我们必须把法线向量从对象坐标系转到世界坐标系。

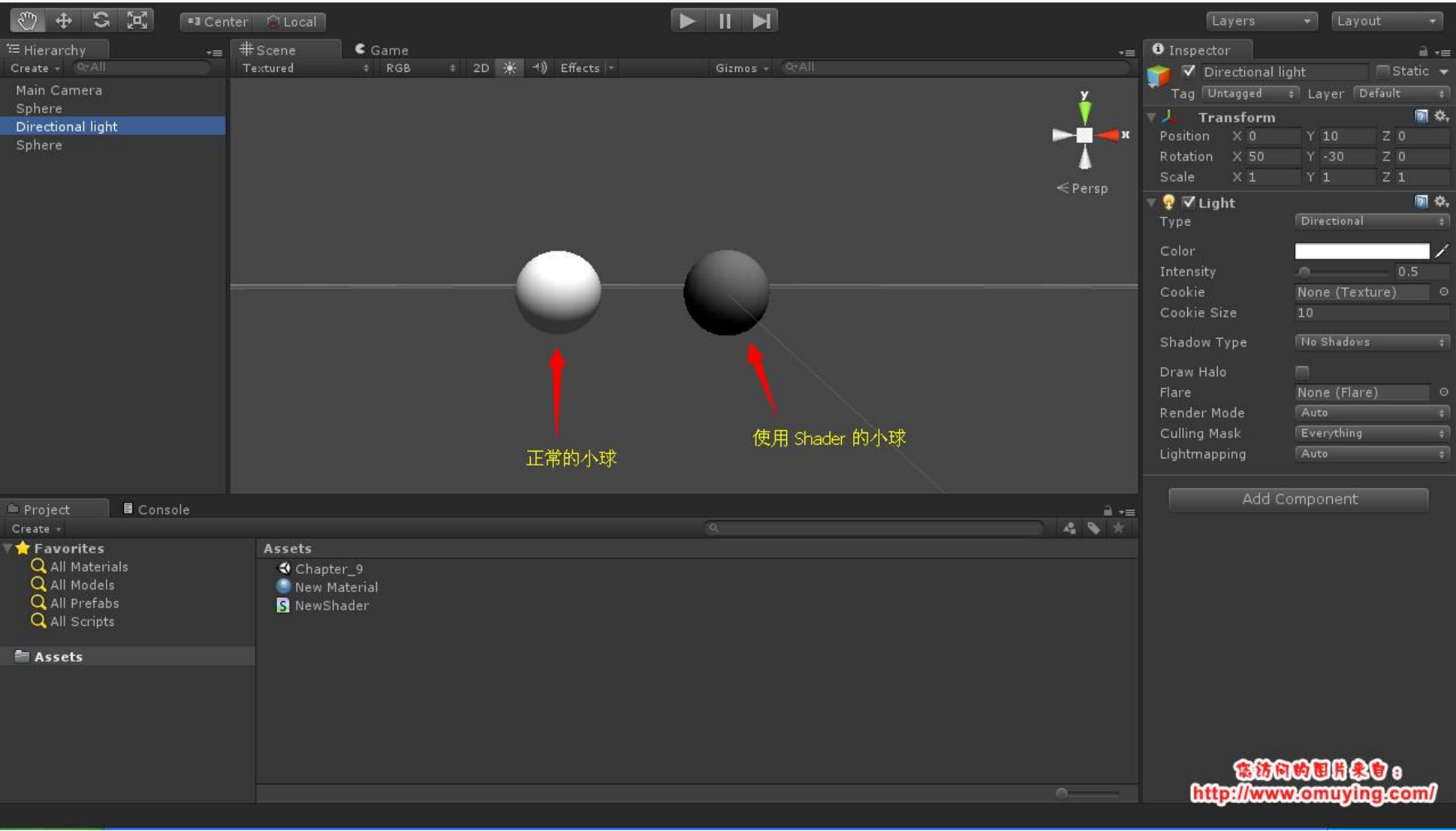
所以着色器的代码看起来是这样的：

```
01 Shader "Cg per-vertex diffuse lighting"
02 {
03     Properties
04     {
05         _Color ("Diffuse Material Color", Color) = (1,1,1,1)
06     }
07     SubShader
08     {
09         Pass
10         {
11             Tags { "LightMode" = "ForwardBase" }
12             // make sure that all uniforms are correctly set
13
14             CGPROGRAM
15
16             #pragma vertex vert
17             #pragma fragment frag
18
19             #include "UnityCG.cginc"
20
21             uniform float4 _LightColor0;
22             // color of light source (from "Lighting.cginc")
23
24             uniform float4 _Color; // define shader property for shaders
25
26             struct vertexInput
27             {
28                 float4 vertex : POSITION;
29                 float3 normal : NORMAL;
30             };
31             struct vertexOutput
32             {
33                 float4 pos : SV_POSITION;
34                 float4 col : COLOR;
35             };
36
37             vertexOutput vert(vertexInput input)
38             {
39                 vertexOutput output;
40
41                 float4x4 modelMatrix = _Object2World;
42                 float4x4 modelMatrixInverse = _World2Object;
43                 // multiplication with unity_Scale.w is unnecessary
44                 // because we normalize transformed vectors
45
46                 float3 normalDirection = normalize(mul(float4(input.normal, 0.0),
47 modelMatrixInverse).xyz);
48                 float3 lightDirection = normalize(_WorldSpaceLightPos0.xyz);
49                 float3 diffuseReflection = _LightColor0.rgb * _Color.rgb *
48 max(0.0, dot(normalDirection, lightDirection));
49
50                 output.col = float4(diffuseReflection, 1.0);
51                 output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
52                 return output;
53             }
54
55             float4 frag(vertexOutput input) : COLOR
56             {
57                 return input.col;
58             }
59         }
60     }
61 }
62
63 // The definition of a fallback shader should be commented out
64 // during development:
65 // Fallback "Diffuse"
```

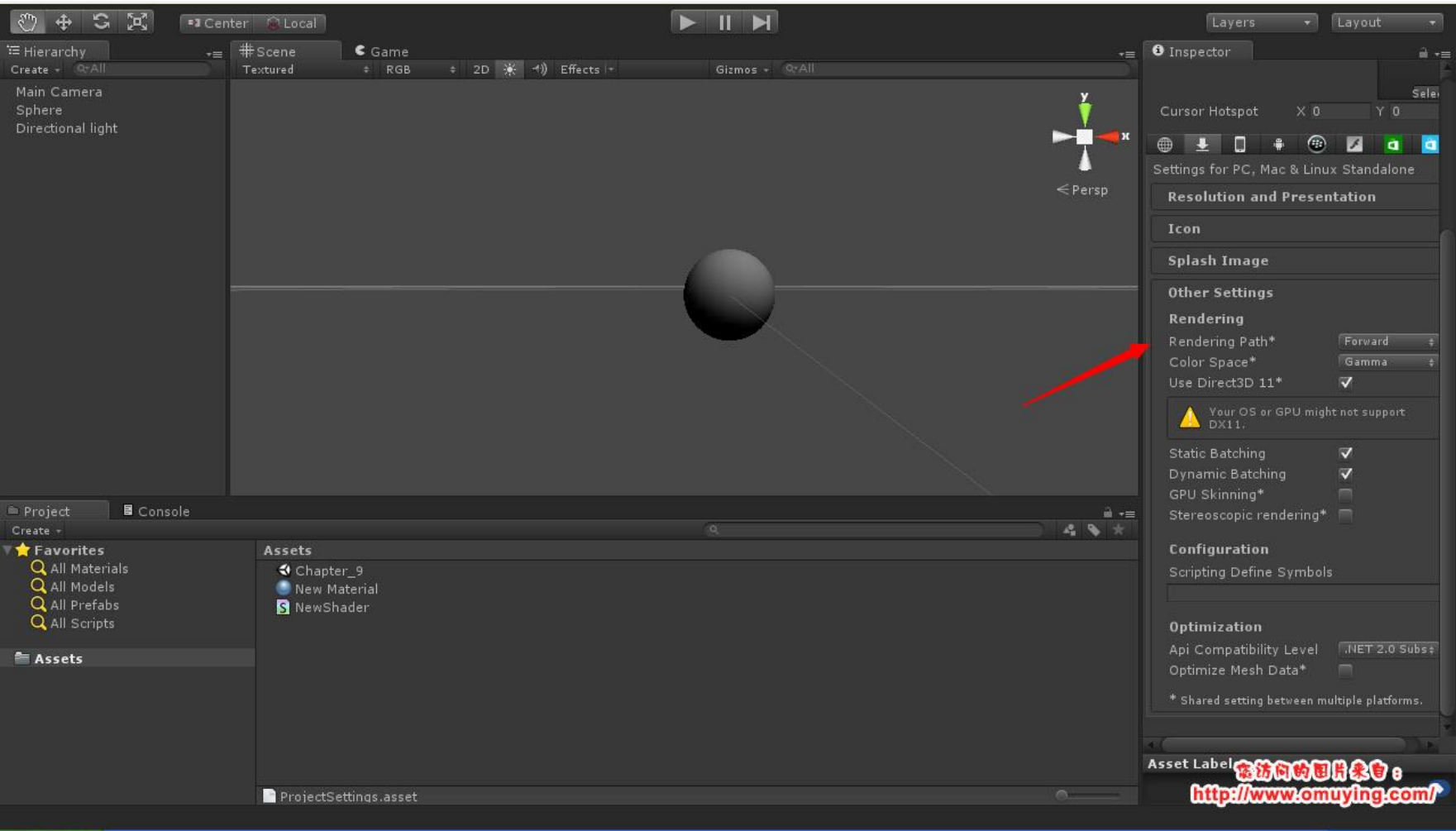
 暂无图片	<a href="#">【原创】Shader 表面着色器语法</a> - 2660 次阅读
 暂无图片	<a href="#">【原创】Shader 内置 Shader 之 Bumped Specular 学习</a> - 1785 次阅读
 暂无图片	<a href="#">【翻译】第四章节：世界空间中的着色器（关于 uniforms）</a> - 2328 次阅读
 暂无图片	<a href="#">【翻译】第二十二章节：Cookies（关于投影纹理贴图塑造光的形状）</a> - 1392 次阅读
 暂无图片	<a href="#">【翻译】第二十四章节：表面反射（关于反射贴图）</a> - 1441 次阅读



为了对比，我们在场景中添加两个球体，一个球体使用上面的着色器，另一个使用 Unity 自带的着色器，效果如图：



使用这个着色器，我们必须并且只有一个方向光，如果没有方向光，我们可以通过 Game Object -> Create Other -> Directional Light 菜单来创建一个方向光，此外还要确保“Forward Rendering Path”处于激活状态，选择 Edit -> Project Settings -> Player 菜单然后在 Inspector 视图中通过 Per-Platform Settings -> Other Settings -> Rendering -> Rendering Path，把 Rendering Path 设置为 Forward，如图：



### Fallback 着色器

代码 Fallback "Diffuse" 的意思是如果 unity 没有找到适合的 subshader，那么 unity 将使用“Diffuse”着色器，在我们的例子中，如果我们没有使用“forward rendering path”或者着色器的代码没有被正确编译，着色器将使用“Diffuse”着色器。我们为着色器指定了“\_Color”属性，这样可以确保 Fallback 的着色器也可以对它进行访问，内置着色器的源码可以在 Unity 网站上获取，选择适当 Fallback 着色器的唯一方法是查看内置着色器的源代码以及内置着色器的属性名称（保证属性一样）。

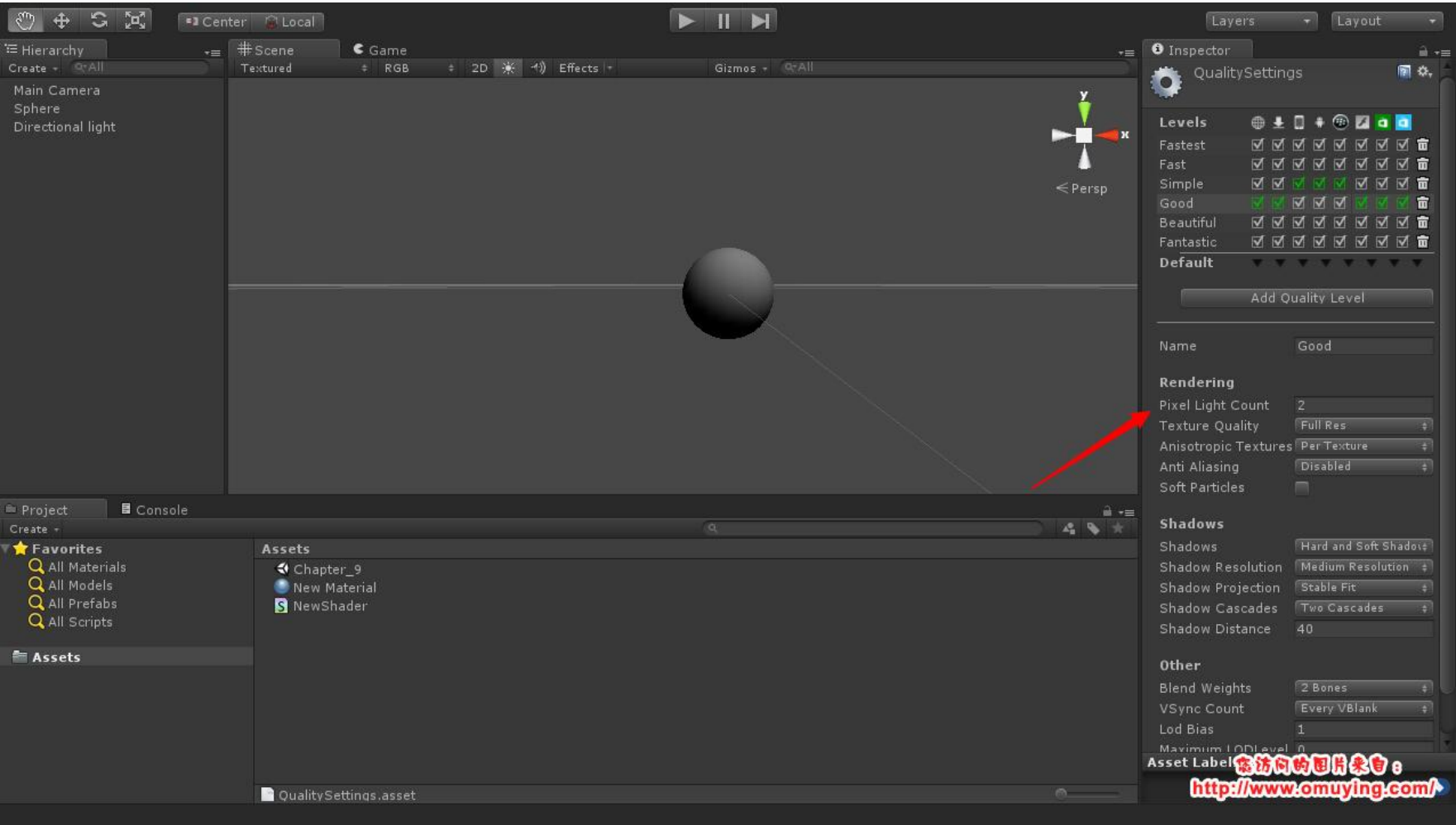
如前所述，如果着色器的代码存在编译错误，那么 Unity 也会使用备用着色器，但是在这种情况下，只有在 Inspector 视图才显示着色器中的错误，因此，如果使用了 fallback 着色器，这往往不利于我们发现错误，所以更好的方法是只在最终的版本中才加入 fallback 着色器。

### 多方向光着色器

到目前为止，我们只考虑了单一光源，在多个光源的处理中，Unity 通过 rendering 和 quality settings 选

项来决定使用什么样的技术，在这个教程中，我们只讨论 “Forward Rendering Path” 。选择 Edit -> Project Settings -> Player 菜单，然后在 Inspector 视图中通过 Per-Platform Settings -> Other Settings -> Rendering -> Rendering Path 设置 Rendering Path 的值为 Forward。

在本教程中，我们只考虑 Unity 中所谓的像素光，对于第一像素光，Unity 通过在 pass 中使用标记 Tags { "LightMode" = "ForwardBase" } 来调用，对于另外的像素光，Unity 使用 pass 标记 Tags { "LightMode" = "ForwardAdd" } 调用，为了确保所有的光都呈现为像素光，你必须确保 quality settings 有足够的像素光：选择 Edit -> Project Settings -> Quality 然后设置标签 Pixel Light Count 的值，如图：



如果在场景中 Unity 允许有多个像素光源，那么只有最重要的光源才会作为像素光来渲染，或者你也可以通过 Render Mode 把所有光源都设置为 Important 来让所有光源都可以作为像素光。

对于 ForwardBase pass，我们的代码到目前为止都还不错，对于 ForwardAdd pass，我们必须给已经存储在帧缓冲区中的光添加反射光，在《透明度》章节中我们已经知道 blend，所以我们可以通过如下公式来指定：

1 | Blend One One

Blending 的值被限制在 0 至 1，所以我们不需要担心颜色或者 alpha 的值大于 1。

所以，我们多方向光的着色器代码为：

```
001 Shader "Cg per-vertex diffuse lighting"
002 {
003     Properties
004     {
005         _Color ("Diffuse Material Color", Color) = (1,1,1,1)
006     }
007     SubShader
008     {
009         Pass
010         {
011             Tags { "LightMode" = "ForwardBase" }
012             // pass for first light source
013
014             CGPROGRAM
015
016             #pragma vertex vert
017             #pragma fragment frag
018
019             #include "UnityCG.cginc"
020
021             uniform float4 _LightColor0;
022             // color of light source (from "Lighting.cginc")
023
024             uniform float4 _Color; // define shader property for shaders
025
026             struct vertexInput
027             {
028                 float4 vertex : POSITION;
029                 float3 normal : NORMAL;
030             };
```

```

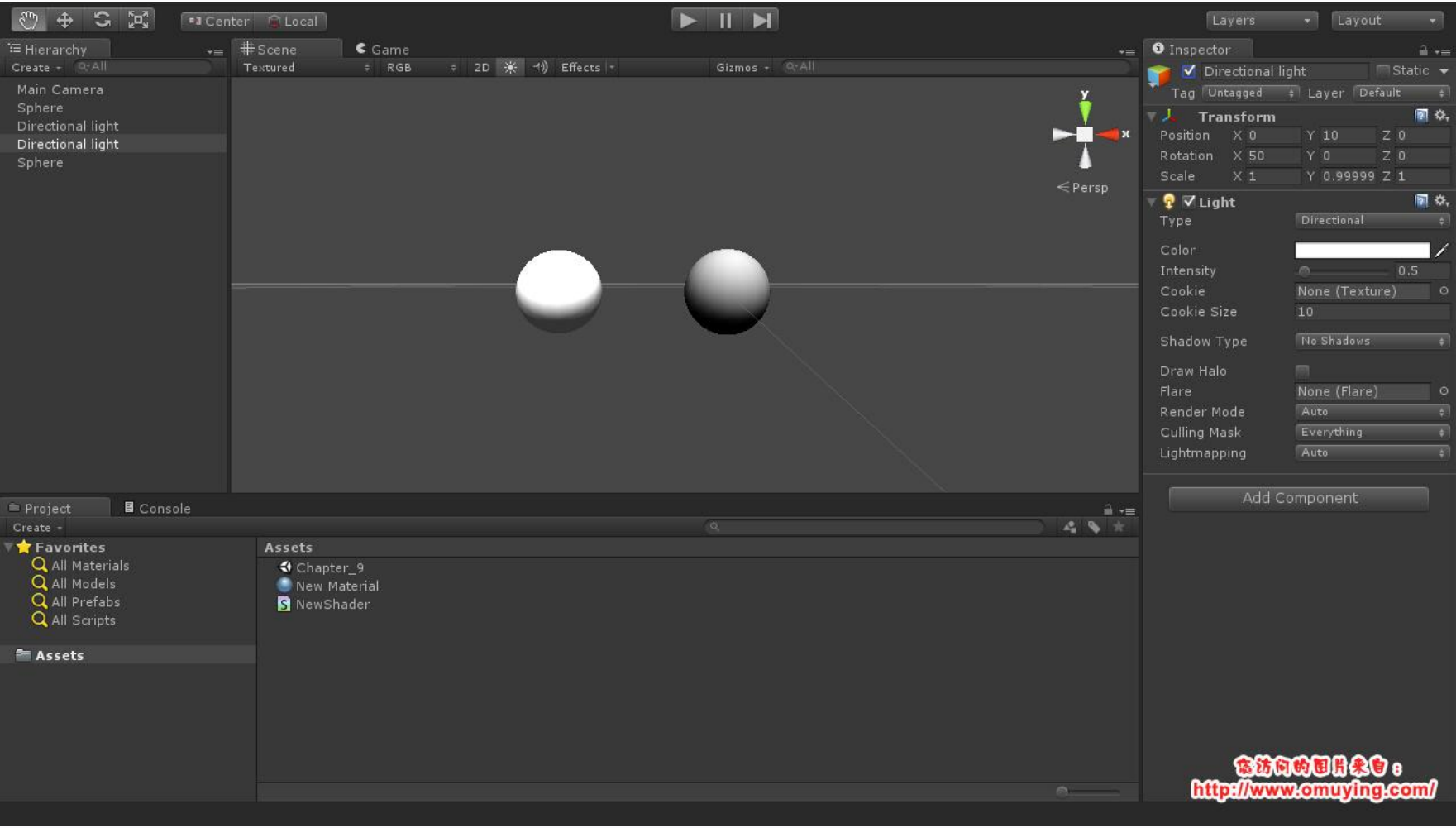
031 struct vertexOutput
032 {
033     float4 pos : SV_POSITION;
034     float4 col : COLOR;
035 };
036
037 vertexOutput vert(vertexInput input)
038 {
039     vertexOutput output;
040
041     float4x4 modelMatrix = _Object2World;
042     float4x4 modelMatrixInverse = _World2Object;
043     // multiplication with unity_Scale.w is unnecessary
044     // because we normalize transformed vectors
045
046     float3 normalDirection = normalize(mul(float4(input.normal, 0.0),
modelMatrixInverse).xyz);
047     float3 lightDirection = normalize(_WorldSpaceLightPos0.xyz);
048
049     float3 diffuseReflection = _LightColor0.rgb * _Color.rgb *
max(0.0, dot(normalDirection, lightDirection));
050
051     output.col = float4(diffuseReflection, 1.0);
052     output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
053     return output;
054 }
055
056 float4 frag(vertexOutput input) : COLOR
057 {
058     return input.col;
059 }
060 ENDCG
061 }
062
063 Pass
064 {
065     Tags { "LightMode" = "ForwardAdd" }
066     // pass for additional light sources
067     Blend One One // additive blending
068
069     CGPROGRAM
070
071     #pragma vertex vert
072     #pragma fragment frag
073
074     #include "UnityCG.cginc"
075
076     uniform float4 _LightColor0;
077     // color of light source (from "Lighting.cginc")
078
079     uniform float4 _Color; // define shader property for shaders
080
081     struct vertexInput
082     {
083         float4 vertex : POSITION;
084         float3 normal : NORMAL;
085     };
086     struct vertexOutput
087     {
088         float4 pos : SV_POSITION;
089         float4 col : COLOR;
090     };
091
092     vertexOutput vert(vertexInput input)
093     {
094         vertexOutput output;
095
096         float4x4 modelMatrix = _Object2World;
097         float4x4 modelMatrixInverse = _World2Object;
098         // multiplication with unity_Scale.w is unnecessary
099         // because we normalize transformed vectors
100
101         float3 normalDirection = normalize(mul(float4(input.normal, 0.0),
modelMatrixInverse).xyz);
102         float3 lightDirection = normalize(_WorldSpaceLightPos0.xyz);
103
104         float3 diffuseReflection = _LightColor0.rgb * _Color.rgb *
max(0.0, dot(normalDirection, lightDirection));
105
106         output.col = float4(diffuseReflection, 1.0);
107         output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
108         return output;
109     }
110
111     float4 frag(vertexOutput input) : COLOR
112     {
113         return input.col;
114     }
115     ENDCG
116 }
117 }
118 // The definition of a fallback shader should be commented out
119 // during development:

```



```
120 // Fallback "Diffuse"
121 }
```

这个着色器的代码有些长，两个 pass 的代码基本相同，我们只是把第二个 pass 的 LightMode 设置成 ForwardAdd 以及添加了 Blend 设置，效果如图：



### 点光源着色器

在方向光的情况下 \_WorldSpaceLightPos0 指定光传来的方向（入射），然而在点光源的情况下，\_WorldSpaceLightPos0 指定光源在世界空间的位置，所以我们必须根据世界空间的顶点的位置到光源的位置来计算不同顶点的光源方向，由于一个点的第四个坐标是 1 以及一个方向的第四个坐标是 0，所以我们可以很容易区别这两种情况：

```
01 float3 lightDirection;
02
03 if (0.0 == _WorldSpaceLightPos0.w) // directional light?
04 {
05     lightDirection = normalize(_WorldSpaceLightPos0.xyz);
06 }
07 else // point or spot light
08 {
09     lightDirection = normalize(_WorldSpaceLightPos0.xyz - mul(modelMatrix,
10 input.vertex).xyz);
10 }
```

虽然方向光不会衰减，但是点光源应该需要根据距离来衰减，光在一点向三个维度展开，它将覆盖更长的距离以及更大的虚拟球体（spheres）。由于每个虚拟球面的光总量是相同的，因此随着半径的增加，每单位面积光的数量会减小，然后我们可以通过光源强度除以光到顶点距离的平方来计算光的衰减。

因为平方衰减比较快，我们随距离进行线性衰减，即光源衰减使用距离来代替距离的平方，所以代码可以是：

```
01 float3 lightDirection;
02 float attenuation;
03
04 if (0.0 == _WorldSpaceLightPos0.w) // directional light?
05 {
06     attenuation = 1.0; // no attenuation
07     lightDirection = normalize(_WorldSpaceLightPos0.xyz);
08 }
09 else // point or spot light
10 {
11     float3 vertexToLightSource = _WorldSpaceLightPos0.xyz - mul(modelMatrix,
12 input.vertex).xyz;
13     float distance = length(vertexToLightSource);
14     attenuation = 1.0 / distance; // linear attenuation
15     lightDirection = normalize(vertexToLightSource);
16 }
```

另外请注意，上面这段代码性能方面比较差，因为任何 if 通常消耗都比较高，由于 \_WorldSpaceLightPos0.w 的值为 0 或者 1，所以我们可以优化代码：

```

1 float3 vertexToLightSource = _WorldSpaceLightPos0.xyz - mul(modelMatrix,
input.vertex * _WorldSpaceLightPos0.w).xyz;
2 float one_over_distance = 1.0 / length(vertexToLightSource);
3 float attenuation = lerp(1.0, one_over_distance, _WorldSpaceLightPos0.w);
4 float3 lightDirection = vertexToLightSource * one_over_distance;

```

衰减因子还应该乘以 `_LightColor0` 来计算入射光，可以查看下面的着色器。

然而，为了代码更加清晰，完整的多个方向和点光源的着色器代码是：

```

001 Shader "Cg per-vertex diffuse lighting"
002 {
003     Properties
004     {
005         _Color ("Diffuse Material Color", Color) = (1,1,1,1)
006     }
007     SubShader
008     {
009         Pass
010         {
011             Tags { "LightMode" = "ForwardBase" }
012             // pass for first light source
013
014             CGPROGRAM
015
016             #pragma vertex vert
017             #pragma fragment frag
018
019             #include "UnityCG.cginc"
020
021             uniform float4 _LightColor0;
022             // color of light source (from "Lighting.cginc")
023
024             uniform float4 _Color; // define shader property for shaders
025
026             struct vertexInput
027             {
028                 float4 vertex : POSITION;
029                 float3 normal : NORMAL;
030             };
031             struct vertexOutput
032             {
033                 float4 pos : SV_POSITION;
034                 float4 col : COLOR;
035             };
036
037             vertexOutput vert(vertexInput input)
038             {
039                 vertexOutput output;
040
041                 float4x4 modelMatrix = _Object2World;
042                 float4x4 modelMatrixInverse = _World2Object;
043                 // multiplication with unity_Scale.w is unnecessary
044                 // because we normalize transformed vectors
045
046                 float3 normalDirection = normalize(mul(float4(input.normal, 0.0),
modelMatrixInverse).xyz);
047                 float3 lightDirection;
048                 float attenuation;
049
050                 if (0.0 == _WorldSpaceLightPos0.w) // directional light?
051                 {
052                     attenuation = 1.0; // no attenuation
053                     lightDirection = normalize(_WorldSpaceLightPos0.xyz);
054                 }
055                 else // point or spot light
056                 {
057                     float3 vertexToLightSource = _WorldSpaceLightPos0.xyz -
mul(modelMatrix, input.vertex).xyz;
058                     float distance = length(vertexToLightSource);
059                     attenuation = 1.0 / distance; // linear attenuation
060                     lightDirection = normalize(vertexToLightSource);
061                 }
062
063                 float3 diffuseReflection = attenuation * _LightColor0.rgb *
_Color.rgb * max(0.0, dot(normalDirection, lightDirection));
064
065                 output.col = float4(diffuseReflection, 1.0);
066                 output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
067                 return output;
068             }
069
070             float4 frag(vertexOutput input) : COLOR
071             {
072                 return input.col;
073             }
074             ENDCG
075         }
076     }
077     Pass

```

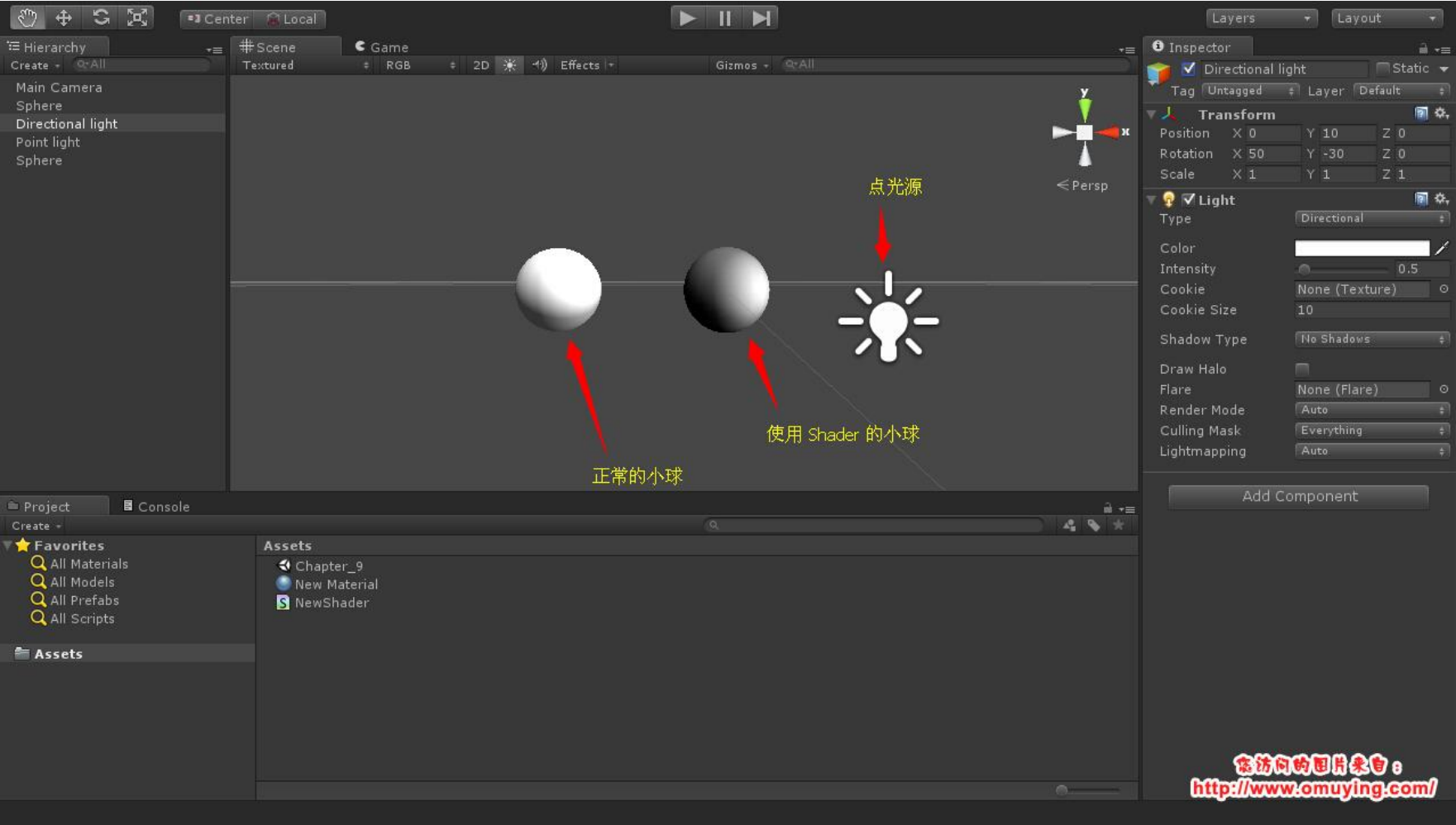
```

078     {
079         Tags { "LightMode" = "ForwardAdd" }
080         // pass for additional light sources
081         Blend One One // additive blending
082
083         CGPROGRAM
084
085         #pragma vertex vert
086         #pragma fragment frag
087
088         #include "UnityCG.cginc"
089
090         uniform float4 _LightColor0;
091         // color of light source (from "Lighting.cginc")
092
093         uniform float4 _Color; // define shader property for shaders
094
095         struct vertexInput
096         {
097             float4 vertex : POSITION;
098             float3 normal : NORMAL;
099         };
100         struct vertexOutput
101         {
102             float4 pos : SV_POSITION;
103             float4 col : COLOR;
104         };
105
106         vertexOutput vert(vertexInput input)
107         {
108             vertexOutput output;
109
110             float4x4 modelMatrix = _Object2World;
111             float4x4 modelMatrixInverse = _World2Object;
112             // multiplication with unity_Scale.w is unnecessary
113             // because we normalize transformed vectors
114
115             float3 normalDirection = normalize(mul(float4(input.normal, 0.0),
modelMatrixInverse).xyz);
116             float3 lightDirection;
117             float attenuation;
118
119             if (0.0 == _WorldSpaceLightPos0.w) // directional light?
120             {
121                 attenuation = 1.0; // no attenuation
122                 lightDirection = normalize(_WorldSpaceLightPos0.xyz);
123             }
124             else // point or spot light
125             {
126                 float3 vertexToLightSource = _WorldSpaceLightPos0.xyz -
mul(modelMatrix, input.vertex).xyz;
127                 float distance = length(vertexToLightSource);
128                 attenuation = 1.0 / distance; // linear attenuation
129                 lightDirection = normalize(vertexToLightSource);
130             }
131
132             float3 diffuseReflection = attenuation * _LightColor0.rgb *
_Color.rgb * max(0.0, dot(normalDirection, lightDirection));
133
134             output.col = float4(diffuseReflection, 1.0);
135             output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
136             return output;
137         }
138
139         float4 frag(vertexOutput input) : COLOR
140         {
141             return input.col;
142         }
143     ENDCG
144 }
145 }
146 // The definition of a fallback shader should be commented out
147 // during development:
148 // Fallback "Diffuse"
149 }

```

注意，光源在 pass ForwardBase 时传递的始终是方向光，因此，第一个 pass 代码可以被简化，效果如图：





如果着色器有问题，记得通过选择 Edit -> Project Settings -> Player 菜单并且在 Inspector 视图中通过 Per-Platform Settings -> Other Settings -> Rendering -> Rendering Path 设置 Rendering Path 的值为 “Forward” 。

### 聚光灯着色器

Unity 通过 cookie textures 实现聚光灯可以查看《[Cookies](#)》章节，然而这毕竟有些高级，这儿我们提到聚光灯，因为他们也是点光源。

恭喜你，在这个章节我们应该了解：

- 1、什么是漫反射。
- 2、单方向光实现漫反射。
- 3、点光源的线性衰减。
- 4、在着色器中使用多个像素灯。

资源下载地址：[点击下载](#)，共下载 15 次。

前一篇：[第八章：轮廓加强（关于转换法线向量）](#)

后一篇：[第十章：镜面高光（关于每顶点光照）](#)



赞

10 人



打酱油

0 人



呵呵

0 人



鄙视

0 人



正能量

0 人



2 条评论

最新 最早 最热



null

你好。貌似 vert 里面计算 normal 的时候有点错误：  
float3 normalDirection = normalize(mul(float4(input.normal, 0.0),  
modelMatrixInverse).xyz);  
应该是  
float3 normalDirection = normalize(mul(float4(input.normal, 0.0), modelMatrix).xyz);

否则法线向量和光源向量没有相同坐标系下。我初学，不知道说得对不对？

7月29日 回复 顶 转发



965

我也觉得modelMatrixInverse得改modelMatrix，看了原文也是楼主这样写的。我测试了下，感觉效果看不出啥区别

8月22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

2条评论

最新   最早   最热



null

你好。貌似 vert 里面计算 normal 的时候有点错误：  
float3 normalDirection = normalize(mul(float4(input.normal, 0.0),  
modelMatrixInverse).xyz);  
应该是  
float3 normalDirection = normalize(mul(float4(input.normal, 0.0), modelMatrix).xyz);

否则法线向量和光源向量没有相同坐标系下。我初学，不知道说得对不对？

7月29日   回复   顶   转发



965

我也觉得modelMatrixInverse得改modelMatrix，看了原文也是楼主这样写的。我测试了下，感觉效果看不出啥区别

8月22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