

```
01 // Diffuse reflection by four "vertex lights"
02 float3 vertexLighting = float3(0.0, 0.0, 0.0);
03 #ifdef VERTEXLIGHT_ON
04 for (int index = 0; index < 4; index++)
05 {
06     float4 lightPosition = float4(unity_4LightPosX0[index],
07         unity_4LightPosY0[index],
08         unity_4LightPosZ0[index], 1.0);
09
10     float3 vertexToLightSource =
11         lightPosition.xyz - output.posWorld.xyz;
12     float3 lightDirection = normalize(vertexToLightSource);
```



```
13     float squaredDistance =
14         dot(vertexToLightSource, vertexToLightSource);
15     float attenuation = 1.0 / (1.0 +
16         unity_4LightAtten0[index] * squaredDistance);
17     float3 diffuseReflection = attenuation
18         * unity_LightColor[index].rgb * _Color.rgb
19         * max(0.0, dot(output.normalDir, lightDirection));
20
21     vertexLighting = vertexLighting + diffuseReflection;
22 }
23 #endif
```

总的漫反射 vertexLighting 值初始化时为黑色，然后在 for 循环的结尾通过添加前一个顶点光照的漫反射值来积累，这儿需要说明一点，任何 C/C++/Java/JavaScript 的程序员都熟悉 for 循环，但是在着色器中使用 for 循环有时候会有一些限制，需要特别说明的是，在 Unity 的着色器中 for 循环的起始值和结束值必须是一个常数（这儿是 0 和 4），另外你也不可以使用 uniform 作为 for 循环的起始和结束值。

这儿或多或少的介绍了 Unity 内置着色器的 vertex lights 如何计算，但是请记住没有什么可以阻止你用这些“vertex lights”来计算镜面反射或者 per-pixel lighting。

完成着色器代码

修改《平滑的镜面高光》章节中的着色器代码来实现多个灯，代码如下：

```
001 Shader "Cg per-pixel lighting with vertex lights"
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
014             // 4 vertex lights, ambient light & first pixel light
015
016             CGPROGRAM
017             #pragma multi_compile_fwdbase
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                 float3 vertexLighting : TEXCOORD2;
041             };
042
043             vertexOutput vert(vertexInput input)
044             {
045                 vertexOutput output;
046
047                 float4x4 modelMatrix = _Object2World;
048                 float4x4 modelMatrixInverse = _World2Object;
049                 // unity_Scale.w is unnecessary here
050
051                 output.posWorld = mul(modelMatrix, input.vertex);
052                 output.normalDir = normalize(mul(float4(input.normal, 0.0),
053 modelMatrixInverse).xyz);
054                 output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
055
056                 // Diffuse reflection by four "vertex lights"
057                 output.vertexLighting = float3(0.0, 0.0, 0.0);
058                 #ifdef VERTEXLIGHT_ON
059                 for (int index = 0; index < 4; index++)
060                 {
061                     float4 lightPosition = float4(unity_4LightPosX0[index],
```

 暂无图片	【原创】Shader 表面着色器语法 - 2660 次阅读
 暂无图片	【翻译】第三章：在着色器中调试（关于顶点输入参数） - 2499 次阅读
 暂无图片	【原创】Shader 内置 Shader 之 Bumped Specular 学习 - 1785 次阅读
 暂无图片	【翻译】第二十二章：Cookies（关于投影纹理贴图塑造光的形状） - 1392 次阅读
 暂无图片	【翻译】第十三章：双面平滑表面（关于双面每像素光照） - 1100 次阅读


```

unity_4LightPosY0[index], unity_4LightPosZ0[index], 1.0);
061
062         float3 vertexToLightSource = lightPosition.xyz -
output.posWorld.xyz;
063         float3 lightDirection = normalize(vertexToLightSource);
064         float squaredDistance = dot(vertexToLightSource,
vertexToLightSource);
065         float attenuation = 1.0 / (1.0 + unity_4LightAtten0[index] *
squaredDistance);
066         float3 diffuseReflection = attenuation *
unity_LightColor[index].rgb * _Color.rgb * max(0.0, dot(output.normalDir,
lightDirection));
067
068         output.vertexLighting = output.vertexLighting +
diffuseReflection;
069     }
070     #endif
071     return output;
072 }
073
074 float4 frag(vertexOutput input) : COLOR
075 {
076     float3 normalDirection = normalize(input.normalDir);
077     float3 viewDirection = normalize(_WorldSpaceCameraPos -
input.posWorld.xyz);
078     float3 lightDirection;
079     float attenuation;
080
081     if (0.0 == _WorldSpaceLightPos0.w) // directional light?
082     {
083         attenuation = 1.0; // no attenuation
084         lightDirection = normalize(_WorldSpaceLightPos0.xyz);
085     }
086     else // point or spot light
087     {
088         float3 vertexToLightSource = _WorldSpaceLightPos0.xyz -
input.posWorld.xyz;
089         float distance = length(vertexToLightSource);
090         attenuation = 1.0 / distance; // linear attenuation
091         lightDirection = normalize(vertexToLightSource);
092     }
093
094     float3 ambientLighting = UNITY_LIGHTMODEL_AMBIENT.rgb *
_Color.rgb;
095
096     float3 diffuseReflection = attenuation * _LightColor0.rgb *
_Color.rgb * max(0.0, dot(normalDirection, lightDirection));
097
098     float3 specularReflection;
099     if (dot(normalDirection, lightDirection) < 0.0) // light source on
the wrong side?
100     {
101         specularReflection = float3(0.0, 0.0, 0.0);
102         // no specular reflection
103     }
104     else // light source on the right side
105     {
106         specularReflection = attenuation * _LightColor0.rgb *
_SpecColor.rgb * pow(max(0.0, dot(reflect(-lightDirection, normalDirection),
viewDirection)), _Shininess);
107     }
108
109     return float4(input.vertexLighting + ambientLighting +
diffuseReflection + specularReflection, 1.0);
110 }
111 ENDCG
112 }
113
114 Pass
115 {
116     Tags { "LightMode" = "ForwardAdd" }
117     // pass for additional light sources
118     Blend One One // additive blending
119
120     CGPROGRAM
121
122     #pragma vertex vert
123     #pragma fragment frag
124
125     #include "UnityCG.cginc"
126     uniform float4 _LightColor0;
127     // color of light source (from "Lighting.cginc")
128
129     // User-specified properties
130     uniform float4 _Color;
131     uniform float4 _SpecColor;
132     uniform float _Shininess;
133
134     struct vertexInput
135     {
136         float4 vertex : POSITION;
137         float3 normal : NORMAL;
138     };

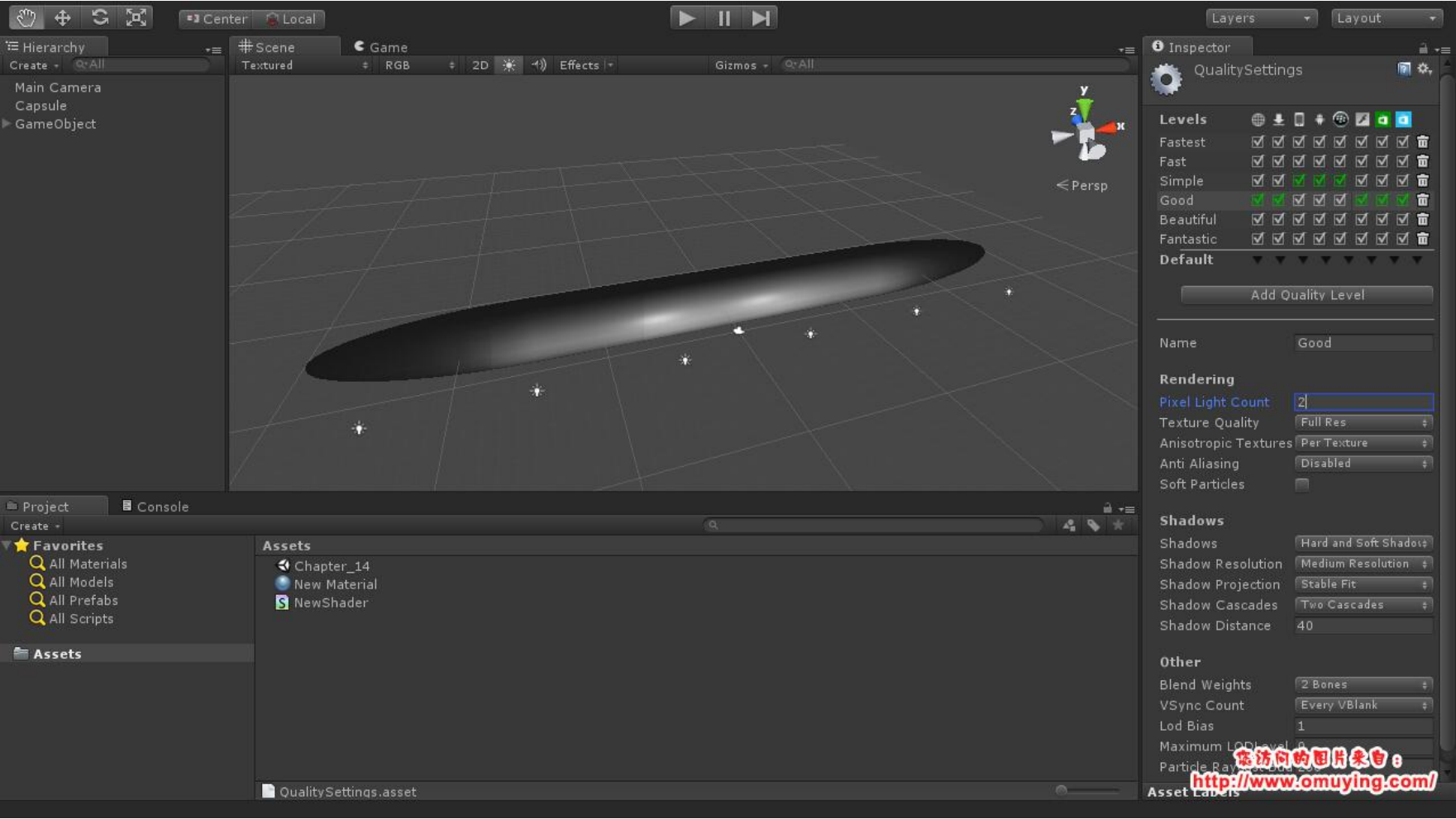
```

```

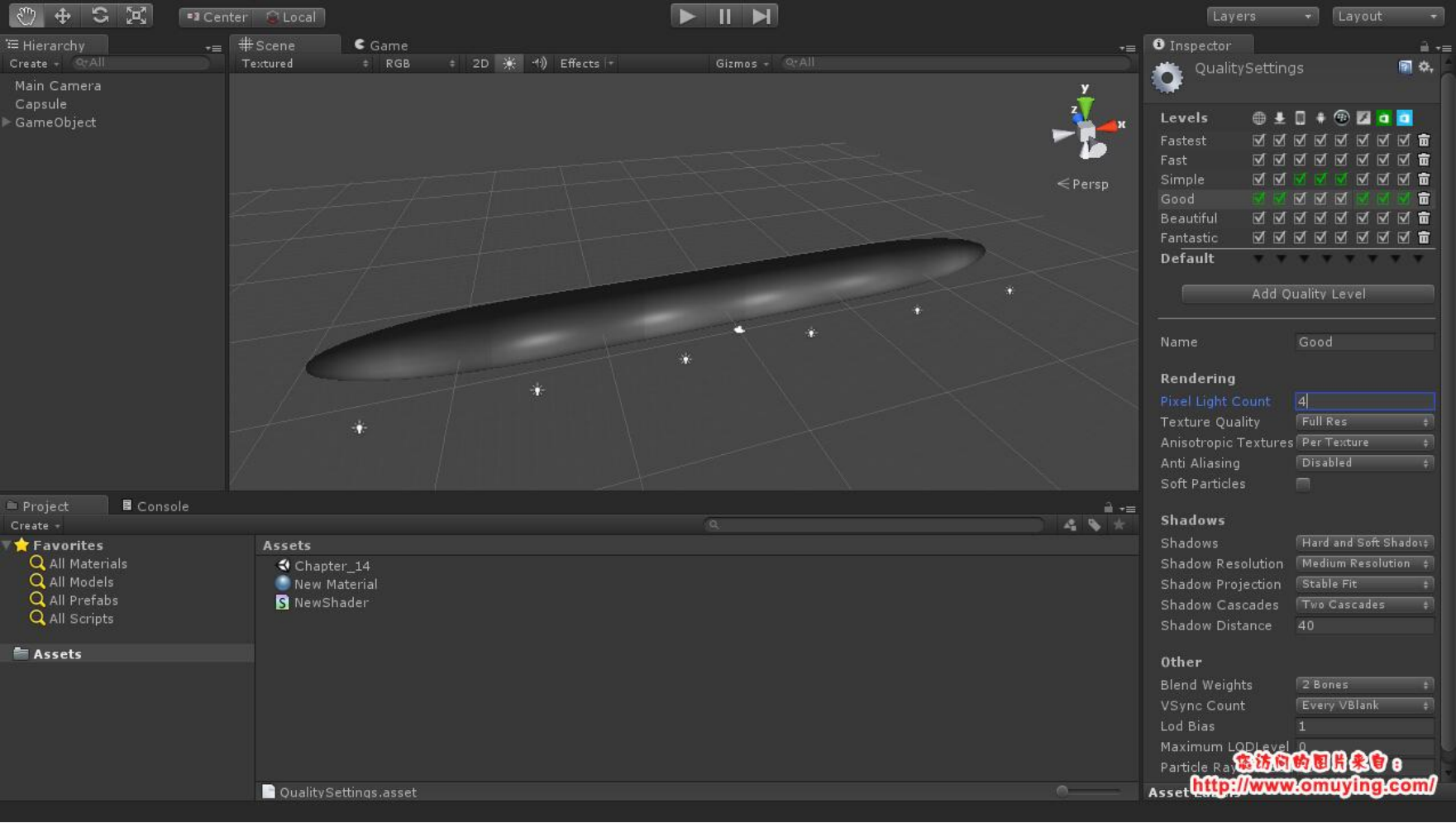
139 struct vertexOutput
140 {
141     float4 pos : SV_POSITION;
142     float4 posWorld : TEXCOORD0;
143     float3 normalDir : TEXCOORD1;
144 };
145
146 vertexOutput vert(vertexInput input)
147 {
148     vertexOutput output;
149
150     float4x4 modelMatrix = _Object2World;
151     float4x4 modelMatrixInverse = _World2Object;
152     // multiplication with unity_Scale.w is unnecessary
153     // because we normalize transformed vectors
154
155     output.posWorld = mul(modelMatrix, input.vertex);
156     output.normalDir = normalize(mul(float4(input.normal, 0.0),
modelMatrixInverse).xyz);
157     output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
158     return output;
159 }
160
161 float4 frag(vertexOutput input) : COLOR
162 {
163     float3 normalDirection = normalize(input.normalDir);
164
165     float3 viewDirection = normalize(_WorldSpaceCameraPos.xyz -
input.posWorld.xyz);
166     float3 lightDirection;
167     float attenuation;
168
169     if (0.0 == _WorldSpaceLightPos0.w) // directional light?
170     {
171         attenuation = 1.0; // no attenuation
172         lightDirection = normalize(_WorldSpaceLightPos0.xyz);
173     }
174     else // point or spot light
175     {
176         float3 vertexToLightSource = _WorldSpaceLightPos0.xyz -
input.posWorld.xyz;
177         float distance = length(vertexToLightSource);
178         attenuation = 1.0 / distance; // linear attenuation
179         lightDirection = normalize(vertexToLightSource);
180     }
181
182     float3 diffuseReflection = attenuation * _LightColor0.rgb *
_Color.rgb * max(0.0, dot(normalDirection, lightDirection));
183
184     float3 specularReflection;
185     if (dot(normalDirection, lightDirection) < 0.0) // light source on
the wrong side?
186     {
187         specularReflection = float3(0.0, 0.0, 0.0);
188         // no specular reflection
189     }
190     else // light source on the right side
191     {
192         specularReflection = attenuation * _LightColor0.rgb *
_SpecColor.rgb * pow(max(0.0, dot(reflect(-lightDirection, normalDirection),
viewDirection)), _Shininess);
193     }
194
195     return float4(diffuseReflection + specularReflection, 1.0);
196     // no ambient lighting in this pass
197 }
198
199 ENDCG
200 }
201
202 }
203 // The definition of a fallback shader should be commented out
204 // during development:
205 // Fallback "Specular"
206 }

```

在着色器代码中使用 `#pragma multi_compile_fwdbase` 和 `#ifdef VERTEXLIGHT_ON ... #endif` 是必要的，这样做是为了确保当 Unity 未提供灯光数据时，不计算顶点光照。我们可以修改 Pixel Light Count 的值并观察效果，当 Pixel Light Count 为 2 时效果如下：



当 Pixel Light Count 为 4 时效果如下：



恭喜你，在本章节中你应该了解：

- 1、如何在 Unity 中指定 vertex lights。
- 2、如何在一个 pass 中遍历计算多光源的光照。

资源下载地址：[点击下载](#)，共下载 22 次。

前一篇：[第十三章节：双面平滑表面（关于双面每像素光照）](#)

后一篇：[第十五章节：纹理球（关于纹理球面）](#)



赞

1 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