

【翻译】第十二章节：光滑的镜面高光（关于每像素光照）

2014-12-04 08:34:00 1181 人阅读 [Unity3D](#) [cg](#) [光滑的镜面高光](#)

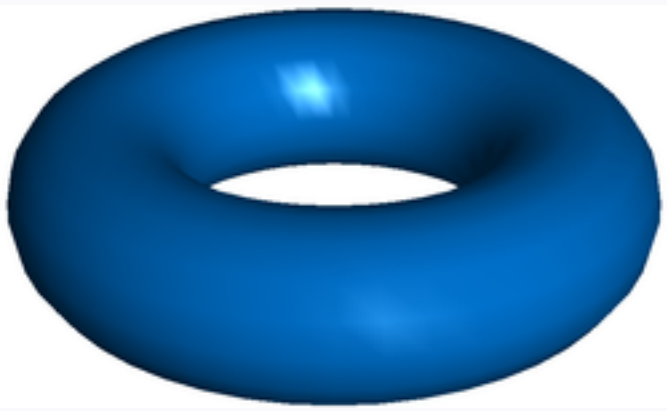
A- A+

文章内容	例子源码	网友评论	最后编辑：2014-12-21 18:22:19
本文永久地址： http://www.omuying.com/article/101.aspx ，【 文章转载请注明出处！ 】			

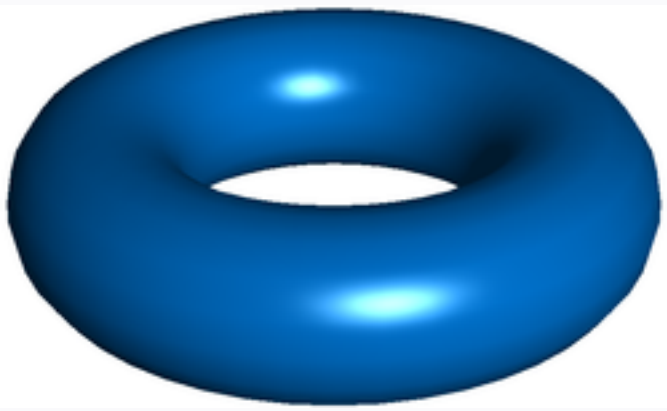
原文链接：http://en.wikibooks.org/wiki/Cg_Programming/Unity/Smooth_Specular_Highlights

本教程介绍 per-pixel lighting。

这篇文章基于《[镜面高光](#)》章节，所以你应该先去阅读它，per-vertex lighting 照明的缺点（计算每个顶点的表面光照然后插值顶点颜色）是表现效果不好，我们可以看看下图的 per-vertex lighting 镜面高光展示：



补救的办法是使用 per-pixel lighting，它的原理是基于法线向量的插值来计算每个片段的光照，这样光照的表现效果会比较好，但是性能方面是要值得注意的，下面是 per-pixel lighting 镜面高光展示：



每像素光照

Per-pixel lighting 也称为 Phong shading（与 per-vertex lighting 一起称为 Gouraud shading），但它不属于 Phong reflection model，Phong reflection model 由环境光照、漫反射、镜面反射组成，详情可以查看《[镜面高光](#)》章节。

Per-pixel lighting 的核心思想是很容易理解的：首先在每个片段上插值法线向量和位置，然后在片段着色器中计算光照。

着色器代码

除了优化，基于 per-vertex lighting 的着色器代码来编写 per-pixel lighting 着色器很简单：光照计算从顶点着色器移到了片段着色器，但是在顶点着色器中需要根据顶点输入参数把光照计算所需要的数据写到顶点输出参数中，然后片段着色器根据这些参数来计算光照。

在这个教程中，我们修改《[镜面高光](#)》章节中的着色器代码来实现 per-pixel lighting，代码如下：

```
001 Shader "Cg per-pixel lighting"
002 {
003     Properties
004     {
005         _Color ("Diffuse Material Color", Color) = (1,1,1,1)
```



最新文章

[【原创】C# 基础之 Lambda 表达式](#) - 907 次阅读

[【原创】C#基础之 IEnumerable 和 IEnumerator](#) - 792 次阅读

[【原创】C#基础之事件](#) - 886 次阅读

[【原创】C#基础之委托](#) - 912 次阅读

[【原创】C#基础之委托的使用](#) - 856 次阅读



随机阅读

```
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" }
014         // pass for ambient light and first light source
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(
053                 mul(float4(input.normal, 0.0), modelMatrixInverse).xyz);
054             output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
055             return output;
056         }
057
058         float4 frag(vertexOutput input) : COLOR
059         {
060             float3 normalDirection = normalize(input.normalDir);
061
062             float3 viewDirection = normalize(_WorldSpaceCameraPos -
input.posWorld.xyz);
063             float3 lightDirection;
064             float attenuation;
065
066             if (0.0 == _WorldSpaceLightPos0.w) // directional light?
067             {
068                 attenuation = 1.0; // no attenuation
069                 lightDirection = normalize(_WorldSpaceLightPos0.xyz);
070             }
071             else // point or spot light
072             {
073                 float3 vertexToLightSource = _WorldSpaceLightPos0.xyz -
input.posWorld.xyz;
074                 float distance = length(vertexToLightSource);
075                 attenuation = 1.0 / distance; // linear attenuation
076                 lightDirection = normalize(vertexToLightSource);
077             }
078
079             float3 ambientLighting = UNITY_LIGHTMODEL_AMBIENT.rgb *
_Color.rgb;
080
081             float3 diffuseReflection = attenuation * _LightColor0.rgb *
_Color.rgb * max(0.0, dot(normalDirection, lightDirection));
082
083             float3 specularReflection;
084             if (dot(normalDirection, lightDirection) < 0.0) // light source on
the wrong side?
085             {
086                 specularReflection = float3(0.0, 0.0, 0.0);
087                 // no specular reflection
088             }
089             else // light source on the right side
090             {
091                 specularReflection = attenuation * _LightColor0.rgb *
_SpecColor.rgb * pow(max(0.0, dot(reflect(-lightDirection, normalDirection),
viewDirection))), _Shininess);

```



【原创】Shader 内置
Shader 之 Parallax
Diffuse 学习 - 1257 次
阅读



【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



【原创】Shader 内置
Shader 之 Diffuse Detail
学习 - 1460 次阅读



【原创】Shader 内置
Shader 之 Specular 学
习 - 2024 次阅读



【翻译】第十一章：双面
表面 (关于双面每顶点光
照) - 1186 次阅读

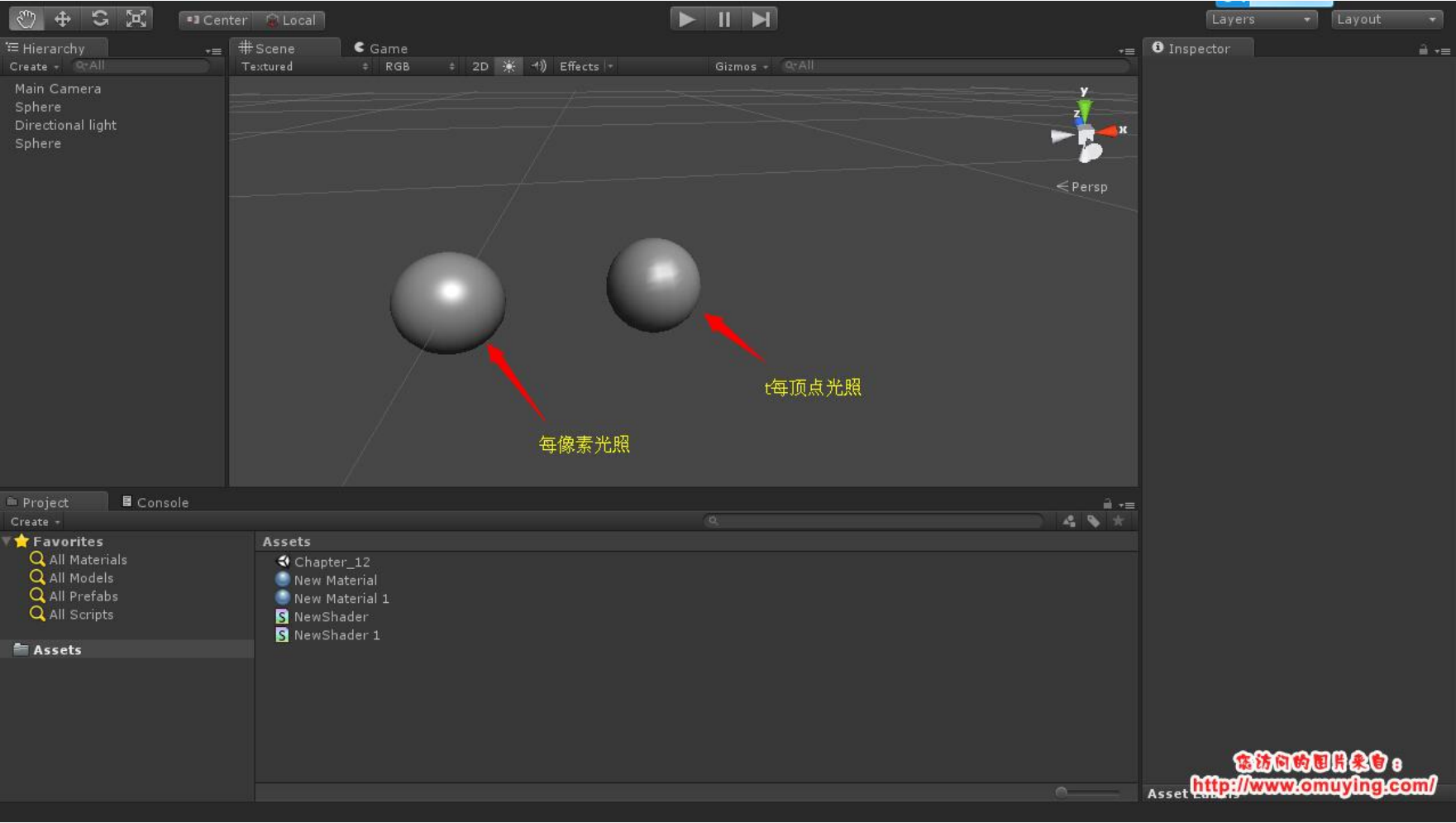

```

092     }
093
094     return float4(ambientLighting + diffuseReflection +
specularReflection, 1.0);
095     }
096     ENDCG
097 }
098
099 Pass
100 {
101     Tags { "LightMode" = "ForwardAdd" }
102     // pass for additional light sources
103     Blend One One // additive blending
104
105     CGPROGRAM
106
107     #pragma vertex vert
108     #pragma fragment frag
109
110     #include "UnityCG.cginc"
111     uniform float4 _LightColor0;
112     // color of light source (from "Lighting.cginc")
113
114     // User-specified properties
115     uniform float4 _Color;
116     uniform float4 _SpecColor;
117     uniform float _Shininess;
118
119     struct vertexInput
120     {
121         float4 vertex : POSITION;
122         float3 normal : NORMAL;
123     };
124     struct vertexOutput
125     {
126         float4 pos : SV_POSITION;
127         float4 posWorld : TEXCOORD0;
128         float3 normalDir : TEXCOORD1;
129     };
130
131     vertexOutput vert(vertexInput input)
132     {
133         vertexOutput output;
134
135         float4x4 modelMatrix = _Object2World;
136         float4x4 modelMatrixInverse = _World2Object;
137         // multiplication with unity_Scale.w is unnecessary
138         // because we normalize transformed vectors
139
140         output.posWorld = mul(modelMatrix, input.vertex);
141         output.normalDir = normalize(mul(float4(input.normal, 0.0),
modelMatrixInverse).xyz);
142         output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
143         return output;
144     }
145
146     float4 frag(vertexOutput input) : COLOR
147     {
148         float3 normalDirection = normalize(input.normalDir);
149
150         float3 viewDirection = normalize(_WorldSpaceCameraPos -
input.posWorld.xyz);
151         float3 lightDirection;
152         float attenuation;
153
154         if (0.0 == _WorldSpaceLightPos0.w) // directional light?
155         {
156             attenuation = 1.0; // no attenuation
157             lightDirection = normalize(_WorldSpaceLightPos0.xyz);
158         }
159         else // point or spot light
160         {
161             float3 vertexToLightSource = _WorldSpaceLightPos0.xyz -
input.posWorld.xyz;
162             float distance = length(vertexToLightSource);
163             attenuation = 1.0 / distance; // linear attenuation
164             lightDirection = normalize(vertexToLightSource);
165         }
166
167         float3 diffuseReflection = attenuation * _LightColor0.rgb *
_Color.rgb * max(0.0, dot(normalDirection, lightDirection));
168
169         float3 specularReflection;
170         if (dot(normalDirection, lightDirection) < 0.0) // light source on
the wrong side?
171         {
172             specularReflection = float3(0.0, 0.0, 0.0);
173             // no specular reflection
174         }
175         else // light source on the right side
176         {
177             specularReflection = attenuation * _LightColor0.rgb *
_SpecColor.rgb * pow(max(0.0, dot(reflect(-lightDirection, normalDirection),

```

```
viewDirection)), _Shininess));
178         }
179     }
180     return float4(diffuseReflection + specularReflection, 1.0);
181     // no ambient lighting in this pass
182 }
183 ENDCG
184 }
185 }
186 // The definition of a fallback shader should be commented out
187 // during development:
188 // Fallback "Specular"
189 }
```

请注意，顶点着色器写入一个规范化的 `output.normalDir` 是为了确保所有方向上的插值权重相同，片段着色器规范化的作用是因为方向被插值之后已经不再规范化了，在场景中，我们添加两个球体，一个球体应用 `per-vertex lighting`，另一个球体应用 `per-pixel lighting`，然后比较这两个着色器的效果，如图：



恭喜你，学习完本章节你应该了解：

- 1、为什么 `per-vertex lighting` 的效果有时候会比较差。
- 2、`per-pixel lighting` 的实现原理，以及如何基于 `per-vertex lighting` 来实现 `per-pixel lighting`。

资源下载地址：[点击下载](#)，共下载 13 次。

前一篇：[第十一章：双面表面（关于双面每顶点光照）](#)

后一篇：[第十三章：双面平滑表面（关于双面每像素光照）](#)



赞

4 人



打酱油

0 人



呵呵

0 人



鄙视

0 人



正能量

0 人



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