

【翻译】第十一章节：双面表面（关于双面每顶点光照）

2014-12-03 08:54:00

1187 人阅读

Unity3D

cg

双面表面

A- A+

文章内容	例子源码	网友评论	最后编辑：2014-12-21 18:20:21
本文永久地址： http://www.omuying.com/article/100.aspx ，【 文章转载请注明出处！ 】			

原文链接：http://en.wikibooks.org/wiki/Cg_Programming/Unity/Two-Sided_Surfaces

本篇教程介绍 two-sided per-vertex lighting。

这是 Unity 基本光照系列教程的一部分，在这个教程中，我们扩展《[镜面高光](#)》章节中的着色器代码来渲染双面表面（two-sided surfaces），如果你还不熟悉《[镜面高光](#)》章节，你应该先去阅读它。

双面光照

如下图所示，这有时候很有用，因为我们可以将不同的颜色应用于两个表面上：



在章节《[剖面模型](#)》章节中，我们已经看到如何在一个着色器中使用两个 pass 来分别剔除一个网格的前脸（外）和后脸（内），这里我们将采用相同的策略。

在《[剖面模型](#)》章节中提到，在 Cg 中我们可以使用片段着色器的输入参数 FACE、VFACE 或者 SV_IsFrontFacing 来区分这两个面（前脸和后脸），但是这个方法在 Unity 中是行不通的。

着色器代码

two-sided per-vertex lighting 着色器的代码是《[镜面高光](#)》章节中着色器代码的简单扩展，它需要设置两个材质参数，并且还需要复制所有的 pass 拷贝（一个拷贝用于前脸剔除，另一个拷贝用于后脸剔除），这两个拷贝的着色器代码不同之处在于后脸着色器中使用了负面（negated surface）法线向量以及使用背面材质属性。

着色器的代码如下：

```
001 Shader "Cg two-sided per-vertex lighting"
002 {
003     Properties
004     {
005         _Color ("Front Material Diffuse Color", Color) = (1,1,1,1)
006         _SpecColor ("Front Material Specular Color", Color) = (1,1,1,1)
007         _Shininess ("Front Material Shininess", Float) = 10
008         _BackColor ("Back Material Diffuse Color", Color) = (1,1,1,1)
009         _BackSpecColor ("Back Material Specular Color", Color) = (1,1,1,1)
010         _BackShininess ("Back Material Shininess", Float) = 10
011     }
012     SubShader
013     {
```



猎头公司排名



外企猎头公司



猎头公司

企业名录

室内设计师

猎头企业

找保姆照顾老人

猎头咨询公司


带车求职

广告

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



上海猎头公司

西安猎头公司

企业名录

找保姆照顾老人

猎头公司

房地产猎头

室内设计师

广告

随机阅读



【翻译】第十五章节：纹理球（关于纹理球面） - 1907 次阅读



【翻译】第十九章节：纹理层（关于多重纹理） - 1661 次阅读



【翻译】第二十一章节：凹凸表面投影（关于视差贴图） - 1462 次阅读



【转载】Shader 变灰效果 - 2552 次阅读



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

```
014 Pass
015 {
016     Tags { "LightMode" = "ForwardBase" }
017     // pass for ambient light and first light source
018     Cull Back // render only front faces
019
020     CGPROGRAM
021
022     #pragma vertex vert
023     #pragma fragment frag
024
025     #include "UnityCG.cginc"
026     uniform float4 _LightColor0;
027     // color of light source (from "Lighting.cginc")
028
029     // User-specified properties
030     uniform float4 _Color;
031     uniform float4 _SpecColor;
032     uniform float _Shininess;
033     uniform float4 _BackColor;
034     uniform float4 _BackSpecColor;
035     uniform float _BackShininess;
036
037     struct vertexInput
038     {
039         float4 vertex : POSITION;
040         float3 normal : NORMAL;
041     };
042     struct vertexOutput
043     {
044         float4 pos : SV_POSITION;
045         float4 col : COLOR;
046         float4 posInObjectCoords : TEXCOORD0; //测试使用, 可以删除
047     };
048
049     vertexOutput vert(vertexInput input)
050     {
051         vertexOutput output;
052
053         float4x4 modelMatrix = _Object2World;
054         float4x4 modelMatrixInverse = _World2Object;
055         // multiplication with unity_Scale.w is unnecessary
056         // because we normalize transformed vectors
057
058         float3 normalDirection = normalize(mul(float4(input.normal, 0.0),
059 modelMatrixInverse).xyz);
060         float3 viewDirection = normalize(_WorldSpaceCameraPos -
061 mul(modelMatrix, input.vertex).xyz);
062         float3 lightDirection;
063         float attenuation;
064
065         if (0.0 == _WorldSpaceLightPos0.w) // directional light?
066         {
067             attenuation = 1.0; // no attenuation
068             lightDirection = normalize(_WorldSpaceLightPos0.xyz);
069         }
070         else // point or spot light
071         {
072             float3 vertexToLightSource = _WorldSpaceLightPos0.xyz -
073 mul(modelMatrix, input.vertex).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 *
080 _Color.rgb;
081
082         float3 diffuseReflection = attenuation * _LightColor0.rgb *
083 _Color.rgb * max(0.0, dot(normalDirection, lightDirection));
084
085         float3 specularReflection;
086         if (dot(normalDirection, lightDirection) < 0.0) // light source on
087 the wrong side?
088         {
089             specularReflection = float3(0.0, 0.0, 0.0);
090             // no specular reflection
091         }
092         else // light source on the right side
093         {
094             specularReflection = attenuation * _LightColor0.rgb *
095 _SpecColor.rgb * pow(max(0.0, dot(reflect(-lightDirection, normalDirection),
096 viewDirection)), _Shininess);
097         }
098
099         output.col = float4(ambientLighting + diffuseReflection +
100 specularReflection, 1.0);
101         output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
102         output.posInObjectCoords = input.vertex; //测试使用, 可以删除
103         return output;
104     }
105
106     float4 frag(vertexOutput input) : COLOR
```

```

098     {
099         if (input.posInObjectCoords.y > 0.0) //测试使用, 可以删除
100         {
101             discard; // drop the fragment if y coordinate > 0
102         }
103         return input.col;
104     }
105     ENDCG
106 }
107
108 Pass
109 {
110     Tags { "LightMode" = "ForwardAdd" }
111     // pass for additional light sources
112     Blend One One // additive blending
113     Cull Back // render only front faces
114
115     CGPROGRAM
116
117     #pragma vertex vert
118     #pragma fragment frag
119
120     #include "UnityCG.cginc"
121     uniform float4 _LightColor0;
122     // color of light source (from "Lighting.cginc")
123
124     // User-specified properties
125     uniform float4 _Color;
126     uniform float4 _SpecColor;
127     uniform float _Shininess;
128     uniform float4 _BackColor;
129     uniform float4 _BackSpecColor;
130     uniform float _BackShininess;
131
132     struct vertexInput
133     {
134         float4 vertex : POSITION;
135         float3 normal : NORMAL;
136     };
137     struct vertexOutput
138     {
139         float4 pos : SV_POSITION;
140         float4 col : COLOR;
141         float4 posInObjectCoords : TEXCOORD0; //测试使用, 可以删除
142     };
143
144     vertexOutput vert(vertexInput input)
145     {
146         vertexOutput output;
147
148         float4x4 modelMatrix = _Object2World;
149         float4x4 modelMatrixInverse = _World2Object;
150         // multiplication with unity_Scale.w is unnecessary
151         // because we normalize transformed vectors
152
153         float3 normalDirection = normalize(mul(float4(input.normal, 0.0),
154 modelMatrixInverse).xyz);
155         float3 viewDirection = normalize(_WorldSpaceCameraPos -
156 mul(modelMatrix, input.vertex).xyz);
157         float3 lightDirection;
158         float attenuation;
159
160         if (0.0 == _WorldSpaceLightPos0.w) // directional light?
161         {
162             attenuation = 1.0; // no attenuation
163             lightDirection = normalize(_WorldSpaceLightPos0.xyz);
164         }
165         else // point or spot light
166         {
167             float3 vertexToLightSource = _WorldSpaceLightPos0.xyz -
168 mul(modelMatrix, input.vertex).xyz;
169             float distance = length(vertexToLightSource);
170             attenuation = 1.0 / distance; // linear attenuation
171             lightDirection = normalize(vertexToLightSource);
172         }
173
174         float3 diffuseReflection = attenuation * _LightColor0.rgb *
175 _Color.rgb * max(0.0, dot(normalDirection, lightDirection));
176
177         float3 specularReflection;
178         if (dot(normalDirection, lightDirection) < 0.0) // light source on
179 the wrong side?
180         {
181             specularReflection = float3(0.0, 0.0, 0.0);
182             // no specular reflection
183         }
184         else // light source on the right side
185         {
186             specularReflection = attenuation * _LightColor0.rgb *
187 _SpecColor.rgb * pow(max(0.0, dot(reflect(-lightDirection, normalDirection),
188 viewDirection)), _Shininess);
189         }
190     }
191 }

```



```

184         output.col = float4(diffuseReflection + specularReflection, 1.0);
185         // no ambient contribution in this pass
186         output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
187         output.posInObjectCoords = input.vertex; //测试使用, 可以删除
188         return output;
189     }
190
191     float4 frag(vertexOutput input) : COLOR
192     {
193         if (input.posInObjectCoords.y > 0.0) //测试使用, 可以删除
194         {
195             discard; // drop the fragment if y coordinate > 0
196         }
197         return input.col;
198     }
199
200     ENDCG
201 }
202
203 Pass
204 {
205     Tags { "LightMode" = "ForwardBase" }
206     // pass for ambient light and first light source
207     Cull Front// render only back faces
208
209     CGPROGRAM
210
211     #pragma vertex vert
212     #pragma fragment frag
213
214     #include "UnityCG.cginc"
215     uniform float4 _LightColor0;
216     // color of light source (from "Lighting.cginc")
217
218     // User-specified properties
219     uniform float4 _Color;
220     uniform float4 _SpecColor;
221     uniform float _Shininess;
222     uniform float4 _BackColor;
223     uniform float4 _BackSpecColor;
224     uniform float _BackShininess;
225
226     struct vertexInput
227     {
228         float4 vertex : POSITION;
229         float3 normal : NORMAL;
230     };
231     struct vertexOutput
232     {
233         float4 pos : SV_POSITION;
234         float4 col : COLOR;
235         float4 posInObjectCoords : TEXCOORD0; //测试使用, 可以删除
236     };
237
238     vertexOutput vert(vertexInput input)
239     {
240         vertexOutput output;
241
242         float4x4 modelMatrix = _Object2World;
243         float4x4 modelMatrixInverse = _World2Object;
244         // multiplication with unity_Scale.w is unnecessary
245         // because we normalize transformed vectors
246
247         float3 normalDirection = normalize(mul(float4(-input.normal, 0.0),
modelMatrixInverse).xyz);
248         float3 viewDirection = normalize(_WorldSpaceCameraPos -
mul(modelMatrix, input.vertex).xyz);
249         float3 lightDirection;
250         float attenuation;
251
252         if (0.0 == _WorldSpaceLightPos0.w) // directional light?
253         {
254             attenuation = 1.0; // no attenuation
255             lightDirection = normalize(_WorldSpaceLightPos0.xyz);
256         }
257         else // point or spot light
258         {
259             float3 vertexToLightSource = _WorldSpaceLightPos0.xyz -
mul(modelMatrix, input.vertex).xyz;
260             float distance = length(vertexToLightSource);
261             attenuation = 1.0 / distance; // linear attenuation
262             lightDirection = normalize(vertexToLightSource);
263         }
264
265         float3 ambientLighting = UNITY_LIGHTMODEL_AMBIENT.rgb *
_BackColor.rgb;
266
267         float3 diffuseReflection = attenuation * _LightColor0.rgb *
_BackColor.rgb * max(0.0, dot(normalDirection, lightDirection));
268
269         float3 specularReflection;
270         if (dot(normalDirection, lightDirection) < 0.0) // light source on
the wrong side?

```

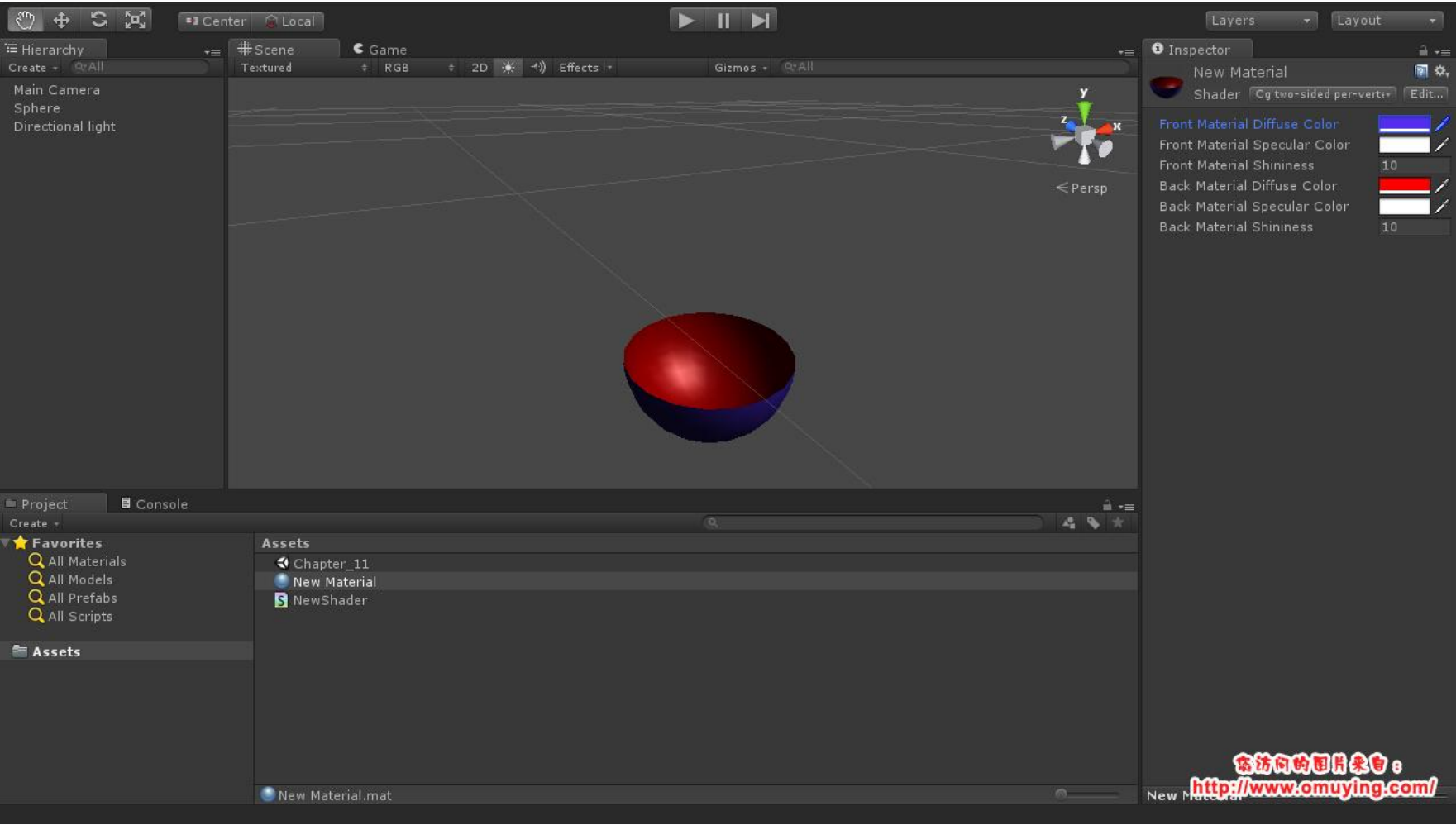
```

271     {
272         specularReflection = float3(0.0, 0.0, 0.0);
273         // no specular reflection
274     }
275     else // light source on the right side
276     {
277         specularReflection = attenuation * _LightColor0.rgb *
_BackSpecColor.rgb * pow(max(0.0, dot(reflect(-lightDirection,
normalDirection), viewDirection)), _BackShininess);
278     }
279
280     output.col = float4(ambientLighting + diffuseReflection +
specularReflection, 1.0);
281     output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
282     output.posInObjectCoords = input.vertex; //测试使用, 可以删除
283     return output;
284 }
285
286 float4 frag(vertexOutput input) : COLOR
287 {
288     if (input.posInObjectCoords.y > 0.0) //测试使用, 可以删除
289     {
290         discard; // drop the fragment if y coordinate > 0
291     }
292     return input.col;
293 }
294
295 ENDCG
296 }
297
298 Pass
299 {
300     Tags { "LightMode" = "ForwardAdd" }
301     // pass for additional light sources
302     Blend One One // additive blending
303     Cull Front // render only back faces
304
305     CGPROGRAM
306
307     #pragma vertex vert
308     #pragma fragment frag
309
310     #include "UnityCG.cginc"
311     uniform float4 _LightColor0;
312     // color of light source (from "Lighting.cginc")
313
314     // User-specified properties
315     uniform float4 _Color;
316     uniform float4 _SpecColor;
317     uniform float _Shininess;
318     uniform float4 _BackColor;
319     uniform float4 _BackSpecColor;
320     uniform float _BackShininess;
321
322     struct vertexInput
323     {
324         float4 vertex : POSITION;
325         float3 normal : NORMAL;
326     };
327     struct vertexOutput
328     {
329         float4 pos : SV_POSITION;
330         float4 col : COLOR;
331         float4 posInObjectCoords : TEXCOORD0; //测试使用, 可以删除
332     };
333
334     vertexOutput vert(vertexInput input)
335     {
336         vertexOutput output;
337
338         float4x4 modelMatrix = _Object2World;
339         float4x4 modelMatrixInverse = _World2Object;
340         // multiplication with unity_Scale.w is unnecessary
341         // because we normalize transformed vectors
342
343         float3 normalDirection = normalize(mul(float4(-input.normal, 0.0),
modelMatrixInverse).xyz);
344         float3 viewDirection = normalize(_WorldSpaceCameraPos -
mul(modelMatrix, input.vertex).xyz);
345         float3 lightDirection;
346         float attenuation;
347
348         if (0.0 == _WorldSpaceLightPos0.w) // directional light?
349         {
350             attenuation = 1.0; // no attenuation
351             lightDirection = normalize(_WorldSpaceLightPos0.xyz);
352         }
353         else // point or spot light
354         {
355             float3 vertexToLightSource = _WorldSpaceLightPos0.xyz -
mul(modelMatrix, input.vertex).xyz;
356             float distance = length(vertexToLightSource);
357             attenuation = 1.0 / distance; // linear attenuation

```

```
358         lightDirection = normalize(vertexToLightSource);
359     }
360
361     float3 diffuseReflection = attenuation * _LightColor0.rgb *
_BackColor.rgb * max(0.0, dot(normalDirection, lightDirection));
362
363     float3 specularReflection;
364     if (dot(normalDirection, lightDirection) < 0.0) // light source on
the wrong side?
365     {
366         specularReflection = float3(0.0, 0.0, 0.0);
367         // no specular reflection
368     }
369     else // light source on the right side
370     {
371         specularReflection = attenuation * _LightColor0.rgb *
_BackSpecColor.rgb * pow(max(0.0, dot(reflect(-lightDirection,
normalDirection), viewDirection)), _BackShininess);
372     }
373
374     output.col = float4(diffuseReflection + specularReflection, 1.0);
375     // no ambient contribution in this pass
376     output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
377     output.posInObjectCoords = input.vertex; //测试使用，可以删除
378     return output;
379 }
380
381 float4 frag(vertexOutput input) : COLOR
382 {
383     if (input.posInObjectCoords.y > 0.0) //测试使用，可以删除
384     {
385         discard; // drop the fragment if y coordinate > 0
386     }
387     return input.col;
388 }
389
390 ENDCG
391 }
392
393 }
394 // The definition of a fallback shader should be commented out
395 // during development:
396 // Fallback "Specular"
397 }
```

这个代码包含了四个 pass，第一对 pass 用于渲染前脸，第二对 pass 使用负（negated）法线向量以及使用背面材质属性来渲染后脸，每对的第二个 Pass 与 第一个 Pass 不同之处在于使用 additive 混合和缺少环境颜色，我们使用一个半球体来查看着色器的效果，如图：



恭喜你，在本教程中你应该了解：

- 1、在一个网格的两个面上使用两个不同的着色器来做前脸剔除和后脸剔除。
- 2、如何为背脸三角形改变 Phone 光照计算。

资源下载地址：[点击下载](#)，共下载 18 次。

前一篇：[Unity3D 使用 A 星寻路（摄像机移动、缩放优化）](#)



赞

6 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