用户登录

首页 Unity3D Shader

.Net(C#)

英语

其他 源码

# 【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)

2014-12-06 08:35:00 1933 人阅读 Unity3D cg 多个灯

 $A^ A^+$ 

文章内容

例子源码

网友评论

最后编辑: 2014-12-21 18:25:00

本文永久地址:http://www.omuying.com/article/103.aspx , 【文章转载请注明出处!】

原文链接:http://en.wikibooks.org/wiki/Cg\_Programming/Unity/Multiple\_Lights

本篇教程介绍在一个 pass 中使用多光源光照,具体来说,它在 ForwardBase pass 中介绍了 Unity 中所谓 的 "vertex lights"。本篇教程是《平面的镜面高光》章节的延伸,所以你应该先阅读一下它。

## 在一个 pass 中使用多个灯

在《漫反射》章节中讨论过, Unity 的正向渲染路径中重要的光源使用 pass 来分开, 这也被称为 "pixel lights",因为内置着色器使用 per-pixel lighting 来渲染他们,所有光源的 Render Mode 被设置为 Important 时才会作为 pixel lights 来渲染, 如果 Unity Quality 选项中的 Pixel Light Count 选项的值设 置得比较大,那么一些光源的 Render Mode 会被设置为 Auto,并且他们也会作为 pixel lights 渲染。那 其他的光源是什么情况呢?Unity 内置着色器在 ForwardBase pass 中用 vertex lights 来渲染四个额外的 光源,正如其名称所示, Unity 的内置着色器渲染这些光源使用 per-vertex lighting。

不幸的是,我们还不清楚如何访问这四个 vertex lights (即他们的位置和颜色),内置的 uniforms 声明如 下:

```
01 // Built-in uniforms for "vertex lights"
    uniform float4 unity_LightColor[4];
   // array of the colors of the 4 light sources
04 uniform float4 unity_4LightPosX0;
   // x coordinates of the 4 light sources in world space
06 uniform float4 unity_4LightPosY0;
   // y coordinates of the 4 light sources in world space
    uniform float4 unity_4LightPosZ0;
   // z coordinates of the 4 light sources in world space
uniform float4 unity_4LightAtten0;
    // scale factors for attenuation with squared distance
12 // uniform float4 unity_LightPosition[4] apparently is not
13 // always correctly set
14 // uniform float4 unity_LightAtten[4] apparently is not
15 // always correctly set
```

根据你使用 Unity 的平台和版本,你可能必须使用 unity\_LightPosition[4] 来代替 unity\_4LightPosX0、 unity\_4LightPosY0 和 unity\_4LightPosZ0。同样的,你可能必须使用 unity\_LightAtten[4]来代替 unity\_4LightAtten0。这儿说他们不可用是因为它们既没有任何 cookie texture 也不做光空间的转换 (英 文: the transformation to light space ),另外,第四个位置的光也是不可用的,因此现在还不清楚一个 vertex light 是方向光、点光源、还是聚光灯。

这儿,我们按照 Unity 的内置着色器,只用 per-vertex lighting 的 vertex lights 来计算漫反射,这个计算 可以通过在顶点着色器中使用 for 循环:

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



【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



【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



企业名录

室内设计师

找保姆照顾老人 猎头公司排名



猎头企业

猎头咨询公司

带车求职

随机阅读

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

```
Shader "Cg per-pixel lighting with vertex lights"
001
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
800
009
        SubShader
010
011
           Pass
012
013
              Tags { "LightMode" = "ForwardBase" } // pass for
014
              // 4 vertex lights, ambient light & first pixel light
015
              CGPROGRAM
016
              #pragma multi_compile_fwdbase
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
              struct vertexInput
030
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;
                 float4x4 modelMatrixInverse = World2Object;
048
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),
    modelMatrixInverse).xyz);
                 output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
053
054
055
                 // Diffuse reflection by four "vertex lights"
056
                 output.vertexLighting = float3(0.0, 0.0, 0.0);
057
                 #ifdef VERTEXLIGHT ON
058
                 for (int index = 0; index < 4; index++)</pre>
059
060
                    float4 lightPosition = float4(unity 4LightPosX0[index],
```

新无图片

【原创】Shader 表面着色 器语法 - 2660 次阅读

新工图片

【翻译】第三章节:在着色器中调试(关于顶点输入参数) - 2499次阅读

**新**无图片

【原创】Shader 内置 Shader 之 Bumped Specular 学 习 - 1785 次阅读

暂无图片

【翻译】第二十二章节:
Cookies(关于投影纹理
贴图塑造光的形
状) - 1392 次阅读

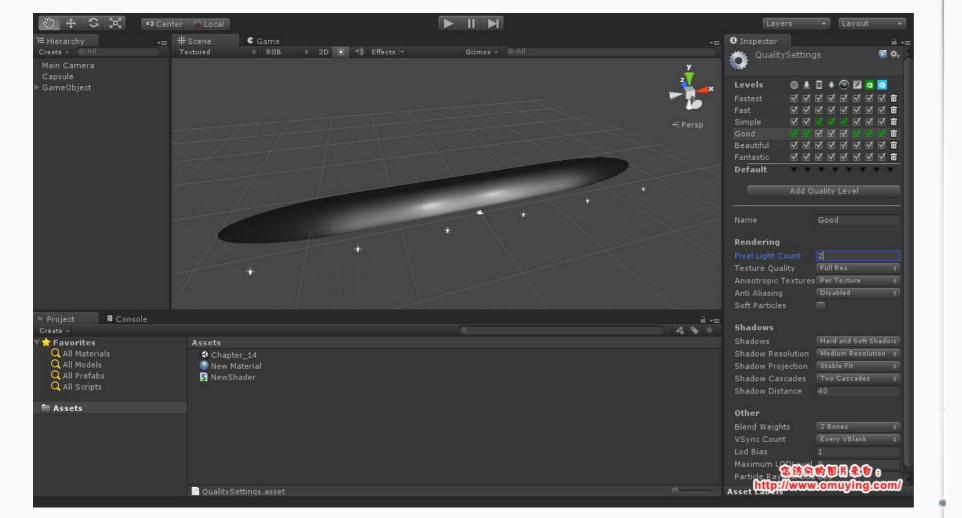
暂无图片

【翻译】第十三章节:双面平滑表面(关于双面每像素光照) - 1100次阅读

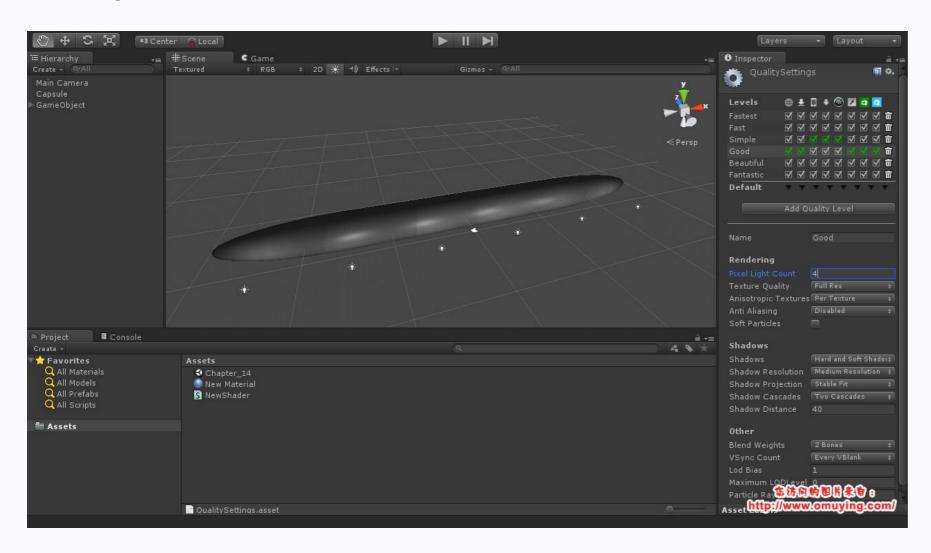
```
unity_4LightPosY0[index], unity_4LightPosZ0[index], 1.0);
061
062
                    float3 vertexToLightSource = lightPosition.xyz -
     output.posWorld.xyz;
                    float3 lightDirection = normalize(vertexToLightSource);
063
064
                    float squaredDistance = dot(vertexToLightSource,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

```
139
              struct vertexOutput
140
                 float4 pos : SV_POSITION;
141
142
                 float4 posWorld : TEXCOORD0;
                 float3 normalDir : TEXCOORD1;
143
144
              };
145
146
              vertexOutput vert(vertexInput input)
147
              {
148
                 vertexOutput output;
149
                 float4x4 modelMatrix = _Object2World;
150
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);
                 output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
157
158
                 return output;
159
              }
160
              float4 frag(vertexOutput input) : COLOR
161
162
163
                 float3 normalDirection = normalize(input.normalDir);
164
165
                 float3 viewDirection = normalize(_WorldSpaceCameraPos.xyz -
     input.posWorld.xyz);
                 float3 lightDirection;
166
167
                 float attenuation;
168
169
                 if (0.0 == _WorldSpaceLightPos0.w) // directional light?
170
                    attenuation = 1.0; // no attenuation
171
172
                    lightDirection = normalize(_WorldSpaceLightPos0.xyz);
173
174
                 else // point or spot light
175
176
                    float3 vertexToLightSource = _WorldSpaceLightPos0.xyz -
     input.posWorld.xyz;
                    float distance = length(vertexToLightSource);
177
                    attenuation = 1.0 / distance; // linear attenuation
178
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</pre>
     the wrong side?
186
                    specularReflection = float3(0.0, 0.0, 0.0);
187
                       // no specular reflection
188
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);
                 // no ambient lighting in this pass
196
197
198
              ENDCG
199
200
201
202
        // The definition of a fallback shader should be commented out
203
        // during development:
204
        // Fallback "Specular"
205
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。

前一篇:第十三章节:双面平滑表面(关于双面每像素光照)

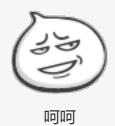
后一篇:第十五章节:纹理球(关于纹理球面)



1人



0人







0人

















0人







0人





0条评论

最新 最早 最热



最终幻想 - 个人博客 | 关于网站 | 联系我们 | 友情链接 | 网站声明 |

Copyright © 2012-2016 最终幻想 - 个人博客 苏ICP备09017906号-5