【翻译】第十章节：镜面高光（关于每顶点光照）

2014-12-01 08:22:00 2008 人阅读 Unity3D cg 镜面高光

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

本教程使用 Phong reflection model 来介绍每顶点光照。

环境光照、镜面反射与《漫反射》一起构成了 Phong reflection model。

环境光照

考虑下面的图片：

虽然大部分的白色衬衫在阴影里，但是任何一部分都不是全黑的，显然墙壁或者其他物体会反射一些光来照亮整个场景（至少一定程度上），在 Phong reflection model 中，环境光照的效果取决于一般环境光的强度 $I_{\text{ambient light}}$ 和漫反射的材质颜色 k_{diffuse}，环境光照的强度 I_{ambient} 公式为：

$$I_{\text{ambient}} = I_{\text{ambient light}}\, k_{\text{diffuse}}$$

与《漫反射》章节中的漫反射的公式类似，本公式也可以被理解为由红、绿、蓝成分构成的光的向量公式。

在 Unity 中，环境光通过选择 Edit -> Render Settings 菜单来指定，在 Unity 的着色器中，通过预定义的 uniform UNITY_LIGHTMODEL_AMBIENT 获得，uniform 详情可以查看《世界空间中的着色器》。

镜面高光

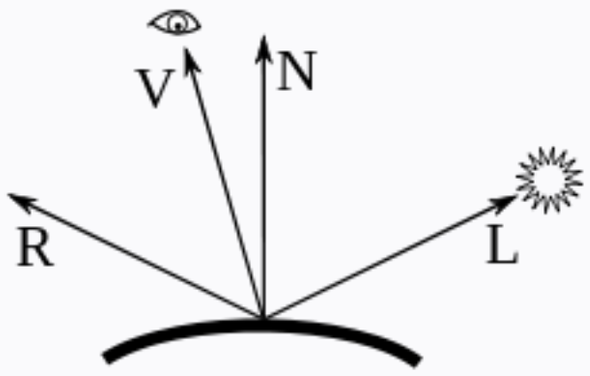
如果你仔细看过上面的图片，那么在人物的鼻子上、头发上、嘴唇上你会看到几处高光，Phong reflection model 中的镜面反射能模拟光滑表面的亮点，它包括一个指定材质光泽的参数 $n_{\text{shininess}}$，亮点光泽指定的原则是：亮点越小则光泽越亮。

假设一个全亮表面的光反射方向为 R（几何反射方向），那么暗淡表面光的反射方向会在 R 的周围（反光越小，方向越广），数学上标准的反射方向 R 的公式为：

$$\mathbf{R} = 2\mathbf{N}(\mathbf{N} \cdot \mathbf{L}) - \mathbf{L}$$

N 表示规范化的表面法线向量，L 表示规范化的光源方向，在 Cg 中，函数 float3 reflect(float3 I, float3 N)（或者 float4 reflect(float4 I, float4 N)）中的参数 I 表示光源到表面点的方向，但是这个函数总是计算出相同的反射向量，所以我们在着色器中不能使用这个函数。

如图所示：



镜面反射通过观察者方向 V 来计算镜面反射，如果 V 接近于 R，反射的强度也会变大（同时也受光泽度 $n_{\text{shininess}}$ 影响），在 Phong reflection model 中，根据 R 与 V 角度余弦值的 $n_{\text{shininess}}$-th 次方（pow 值）来生成不同光泽的亮点，与《漫反射》章节中的一样，我们需要限制负余弦的值为 0，此外镜面反射还要求一个材质颜色 k_{specular}（通常是白色），这样使得所有亮点都有入射光 I_{incoming} 的颜色，例如在上面的图中，人物身上的所有亮点都是白色。Phong reflection model 镜面公式为：

$$I_{\text{specular}} = I_{\text{incoming}} k_{\text{specular}} \max(0, \mathbf{R} \cdot \mathbf{V})^{n_{\text{shininess}}}$$

与《漫反射》章节相似，如果 N·L 的值为负数，镜面表面的反射的光则会被忽略。

着色器代码

在着色器中环境光照可以很简单的通过向量与向量的积计算出：

```
1  float3 ambientLighting = UNITY_LIGHTMODEL_AMBIENT.rgb * _Color.rgb;
```

实现镜面反射，我们可以通过世界空间中的顶点位置与世界空间中的摄像机位置计算出观察者方向，世界空间中的摄像机位置由 Unity 的 uniform _WorldSpaceCameraPos 提供，顶点位置可以像《漫反射》章节中的那样转换到世界空间中，世界空间中的镜面反射公式可以像下面代码那样实现：

```
01  float3 viewDirection = normalize(_WorldSpaceCameraPos - mul(modelMatrix,
    input.vertex).xyz);
02
03  float3 specularReflection;
04  if (dot(normalDirection, lightDirection) < 0.0) // light source on the wrong
    side?
05  {
06      specularReflection = float3(0.0, 0.0, 0.0);
07      // no specular reflection
08  }
09  else // light source on the right side
10  {
11      specularReflection = attenuation * _LightColor0.rgb * _SpecColor.rgb *
    pow(max(0.0, dot(reflect(-lightDirection, normalDirection), viewDirection)),
    _Shininess);
12  }
```

这段代码使用了与《漫反射》章节中着色器代码相同的变量，此外用户还指定了属性 _SpecColor 和 _Shininess，pow(a, b) 用于计算 a^b。

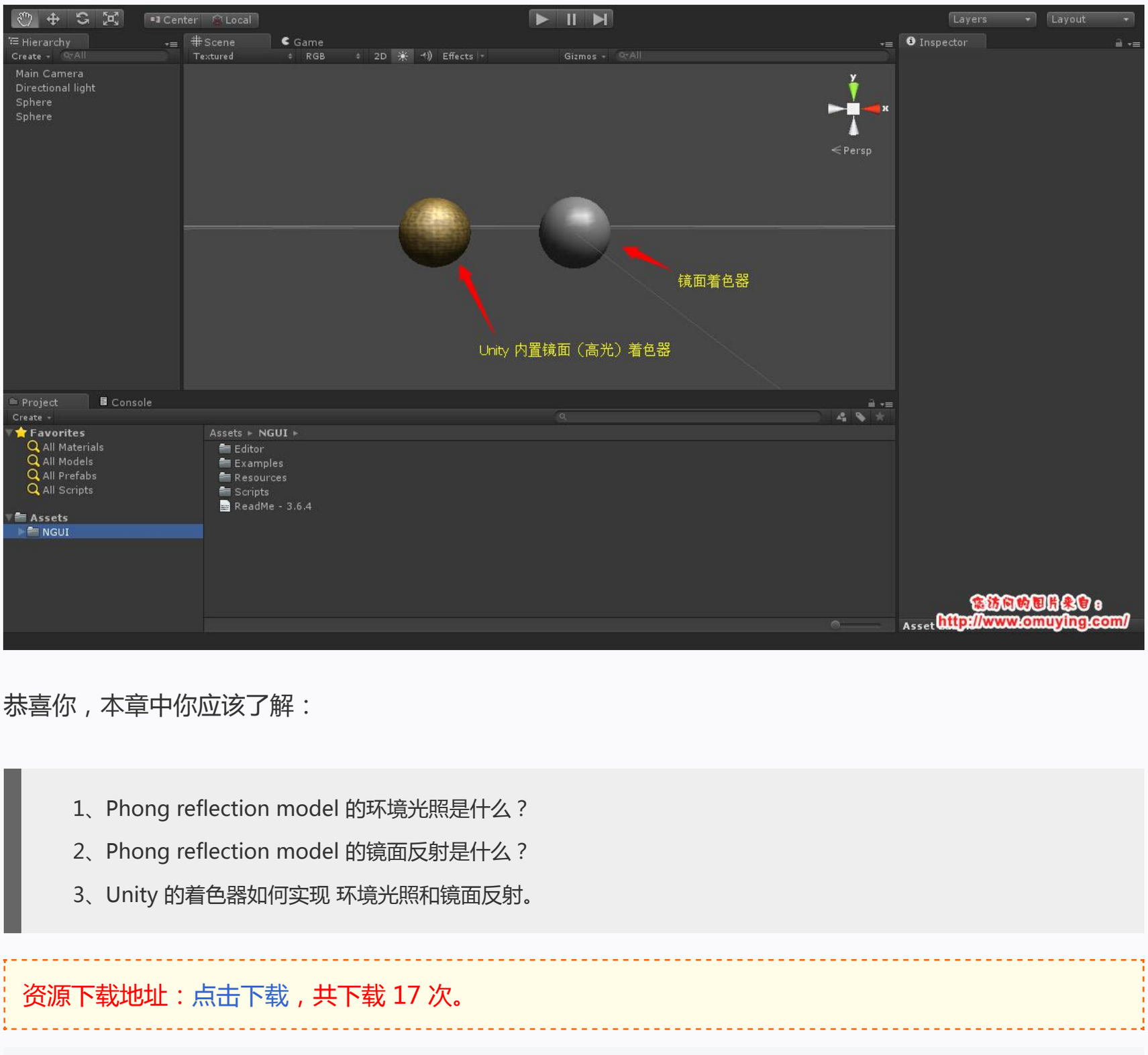
下面的着色器代码展示了环境照明与镜面反射，我们可以看到环境照明只在第一个 pass 被添加（只需要一次），镜面反射在两个 pass 中都有添加，着色器的代码如下：

```
001 Shader "Cg per-vertex lighting"
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" }
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 col : COLOR;
039         };
040
041         vertexOutput vert(vertexInput input)
042         {
043            vertexOutput output;
044
045            float4x4 modelMatrix = _Object2World;
046            float4x4 modelMatrixInverse = _World2Object;
047            // multiplication with unity_Scale.w is unnecessary
048            // because we normalize transformed vectors
049
050            float3 normalDirection = normalize(mul(float4(input.normal, 0.0),
modelMatrixInverse).xyz);
051            float3 viewDirection = normalize(_WorldSpaceCameraPos -
mul(modelMatrix, input.vertex).xyz);
052            float3 lightDirection;
053            float attenuation;
054
055            if (0.0 == _WorldSpaceLightPos0.w) // directional light?
056            {
057               attenuation = 1.0; // no attenuation
058               lightDirection = normalize(_WorldSpaceLightPos0.xyz);
059            }
060            else // point or spot light
061            {
062               float3 vertexToLightSource = _WorldSpaceLightPos0.xyz -
mul(modelMatrix, input.vertex).xyz;
063               float distance = length(vertexToLightSource);
064               attenuation = 1.0 / distance; // linear attenuation
065               lightDirection = normalize(vertexToLightSource);
066            }
067
068            float3 ambientLighting = UNITY_LIGHTMODEL_AMBIENT.rgb *
_Color.rgb;
069
070            float3 diffuseReflection = attenuation * _LightColor0.rgb *
_Color.rgb * max(0.0, dot(normalDirection, lightDirection));
071
072            float3 specularReflection;
073            if (dot(normalDirection, lightDirection) < 0.0) // light source on
the wrong side?
074            {
075               specularReflection = float3(0.0, 0.0, 0.0);
076               // no specular reflection
077            }
078            else // light source on the right side
079            {
080               specularReflection = attenuation * _LightColor0.rgb *
_SpecColor.rgb * pow(max(0.0, dot(reflect(-lightDirection, normalDirection),
viewDirection)), _Shininess);
081            }
082
083            output.col = float4(ambientLighting + diffuseReflection +
specularReflection, 1.0);
084            output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
085            return output;
086         }
087
088         float4 frag(vertexOutput input) : COLOR
089         {
090            return input.col;
091         }
092
093         ENDCG
094      }
095
096      Pass
097      {
098         Tags { "LightMode" = "ForwardAdd" }
099         // pass for additional light sources
100         Blend One One // additive blending
101
102         CGPROGRAM
103
104         #pragma vertex vert
105         #pragma fragment frag
106
107         #include "UnityCG.cginc"
```

```
            uniform float4 _LightColor0;
            // color of light source (from "Lighting.cginc")

            // User-specified properties
            uniform float4 _Color;
            uniform float4 _SpecColor;
            uniform float _Shininess;

            struct vertexInput
            {
               float4 vertex : POSITION;
               float3 normal : NORMAL;
            };
            struct vertexOutput
            {
               float4 pos : SV_POSITION;
               float4 col : COLOR;
            };

            vertexOutput vert(vertexInput input)
            {
               vertexOutput output;

               float4x4 modelMatrix = _Object2World;
               float4x4 modelMatrixInverse = _World2Object;
               // multiplication with unity_Scale.w is unnecessary
               // because we normalize transformed vectors

               float3 normalDirection = normalize(mul(float4(input.normal, 0.0), modelMatrixInverse).xyz);
               float3 viewDirection = normalize(_WorldSpaceCameraPos - mul(modelMatrix, input.vertex).xyz);
               float3 lightDirection;
               float attenuation;

               if (0.0 == _WorldSpaceLightPos0.w) // directional light?
               {
                  attenuation = 1.0; // no attenuation
                  lightDirection = normalize(_WorldSpaceLightPos0.xyz);
               }
               else // point or spot light
               {
                  float3 vertexToLightSource = _WorldSpaceLightPos0.xyz - mul(modelMatrix, input.vertex).xyz;
                  float distance = length(vertexToLightSource);
                  attenuation = 1.0 / distance; // linear attenuation
                  lightDirection = normalize(vertexToLightSource);
               }

               float3 diffuseReflection = attenuation * _LightColor0.rgb * _Color.rgb * max(0.0, dot(normalDirection, lightDirection));

               float3 specularReflection;
               if (dot(normalDirection, lightDirection) < 0.0) // light source on the wrong side?
               {
                  specularReflection = float3(0.0, 0.0, 0.0);
                  // no specular reflection
               }
               else // light source on the right side
               {
                  specularReflection = attenuation * _LightColor0.rgb * _SpecColor.rgb * pow(max(0.0, dot(reflect(-lightDirection, normalDirection), viewDirection)), _Shininess);
               }

               output.col = float4(diffuseReflection + specularReflection, 1.0);
               // no ambient contribution in this pass
               output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
               return output;
            }

            float4 frag(vertexOutput input) : COLOR
            {
               return input.col;
            }

            ENDCG
         }
      }
      // The definition of a fallback shader should be commented out
      // during development:
      // Fallback "Specular"
}
```

为了与 Unity 内置的镜面（高光）着色器产生对比，我们在场景中添加两个小球，左边的小球使用内置镜面着色器，右边的使用上面的着色器，效果如图：



恭喜你，本章中你应该了解：

> 1、Phong reflection model 的环境光照是什么？
> 2、Phong reflection model 的镜面反射是什么？
> 3、Unity 的着色器如何实现 环境光照和镜面反射。

资源下载地址：点击下载，共下载 17 次。

前一篇：第九章节：漫反射（关于每顶点漫反射和多光源漫反射）
后一篇：Unity3D 使用 A 星寻路（摄像机移动、缩放优化）

说点什么吧...

发布

最终幻想正在使用多说