【翻译】第十三章节：双面平滑表面（关于双面每像素光照）

2014-12-05 08:35:00 1101 人阅读 Unity3D cg 双面平滑表面 A- A+

文章内容 例子源码 网友评论 最后编辑：2014-12-21 18:23:38

本文永久地址：http://www.omuying.com/article/102.aspx，【文章转载请注明出处！】

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

本篇教程介绍 two-sided per-pixel lighting。

本教程合并了《平滑的镜面高光》章节中的 per-pixel lighting 和《双面表面》章节中的 two-sided lighting 着色器代码。

着色器代码

根据《平滑的镜面高光》章节，我们需要做以下修改：添加背面材质属性，添加两个 pass ，一个用来做前脸剔除，一个用来做后脸剔除，渲染背脸时使用负表面法线向量，所以着色器的代码如下：

```
Shader "Cg two-sided per-pixel lighting"
{
   Properties
   {
      _Color ("Diffuse Material Color", Color) = (1,1,1,1)
      _SpecColor ("Specular Material Color", Color) = (1,1,1,1)
      _Shininess ("Shininess", Float) = 10
      _BackColor ("Back Material Diffuse Color", Color) = (1,1,1,1)
      _BackSpecColor ("Back Material Specular Color", Color) = (1,1,1,1)
      _BackShininess ("Back Material Shininess", Float) = 10
   }
   SubShader
   {
      Pass {
         Tags { "LightMode" = "ForwardBase" }
         // pass for ambient light and first light source
         Cull Back // render only front faces

         CGPROGRAM

         #pragma vertex vert
         #pragma fragment frag

         #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;
         uniform float4 _BackColor;
         uniform float4 _BackSpecColor;
         uniform float _BackShininess;

         struct vertexInput
         {
            float4 vertex : POSITION;
            float3 normal : NORMAL;
         };
         struct vertexOutput
         {
            float4 pos : SV_POSITION;
            float4 posWorld : TEXCOORD0;
            float3 normalDir : TEXCOORD1;
            float4 posInObjectCoords : TEXCOORD2; //测试用，可以删除
         };

         vertexOutput vert(vertexInput input)
         {
            vertexOutput output;

            float4x4 modelMatrix = _Object2World;
```

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

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

            output.posWorld = mul(modelMatrix, input.vertex);
            output.normalDir = normalize(mul(float4(input.normal, 0.0), modelMatrixInverse).xyz);
            output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
            output.posInObjectCoords = input.vertex; //测试用，可以删除
            return output;
         }

         float4 frag(vertexOutput input) : COLOR
         {
            if (input.posInObjectCoords.y > 0.0) //测试用，可以删除
            {
               discard; // drop the fragment if y coordinate > 0
            }
            float3 normalDirection = normalize(input.normalDir);

            float3 viewDirection = normalize(_WorldSpaceCameraPos - input.posWorld.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 - input.posWorld.xyz;
               float distance = length(vertexToLightSource);
               attenuation = 1.0 / distance; // linear attenuation
               lightDirection = normalize(vertexToLightSource);
            }

            float3 ambientLighting = UNITY_LIGHTMODEL_AMBIENT.rgb * _Color.rgb;

            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);
            }

            return float4(ambientLighting + diffuseReflection + specularReflection, 1.0);
         }
         ENDCG
      }

      Pass
      {
         Tags { "LightMode" = "ForwardAdd" }
         // pass for additional light sources
         Blend One One // additive blending
         Cull Back // render only front faces

         CGPROGRAM

         #pragma vertex vert
         #pragma fragment frag

         #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;
         uniform float4 _BackColor;
         uniform float4 _BackSpecColor;
         uniform float _BackShininess;

         struct vertexInput
         {
            float4 vertex : POSITION;
            float3 normal : NORMAL;
```

【翻译】第九章节：漫反射（关于每顶点漫反射和多光源漫反射） - 1946 次阅读

【翻译】第十九章节：纹理层（关于多重纹理） - 1661 次阅读

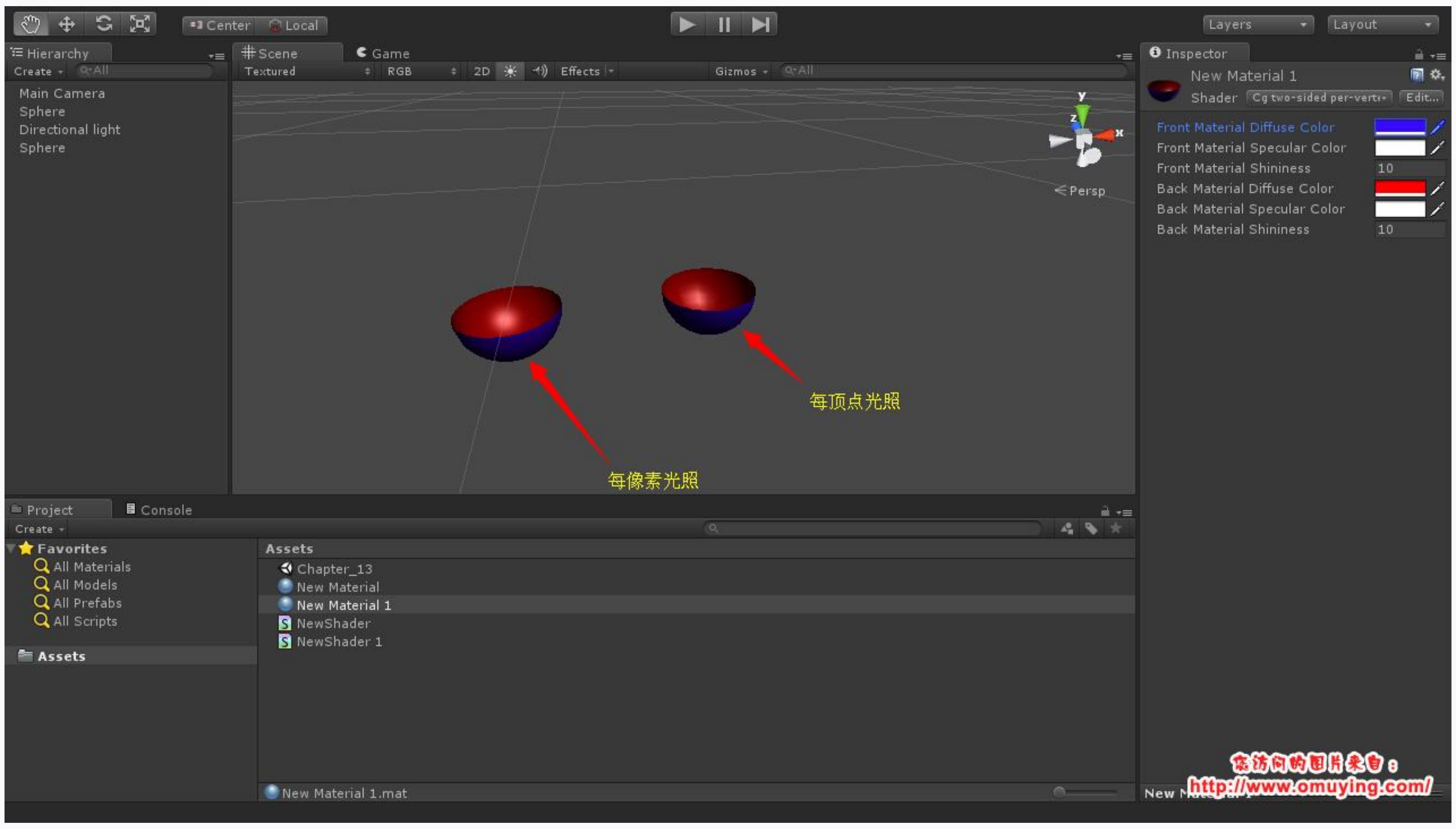
【原创】Shader 内置 Shader 之 Parallax Diffuse 学习 - 1257 次阅读

【翻译】第十七章节：光泽纹理（关于光泽贴图） - 1201 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

```
         };
         struct vertexOutput
         {
            float4 pos : SV_POSITION;
            float4 posWorld : TEXCOORD0;
            float3 normalDir : TEXCOORD1;
            float4 posInObjectCoords : TEXCOORD2; //测试用，可以删除
         };

         vertexOutput vert(vertexInput input)
         {
            vertexOutput output;

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

            output.posWorld = mul(modelMatrix, input.vertex);
            output.normalDir = normalize(mul(float4(input.normal, 0.0), modelMatrixInverse).xyz);
            output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
            output.posInObjectCoords = input.vertex; //测试用，可以删除
            return output;
         }

         float4 frag(vertexOutput input) : COLOR
         {
            if (input.posInObjectCoords.y > 0.0) //测试用，可以删除
            {
               discard; // drop the fragment if y coordinate > 0
            }
            float3 normalDirection = normalize(input.normalDir);

            float3 viewDirection = normalize(_WorldSpaceCameraPos - input.posWorld.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 - input.posWorld.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);
            }

            return float4(diffuseReflection + specularReflection, 1.0);
            // no ambient lighting in this pass
         }
         ENDCG
      }

      Pass
      {
         Tags { "LightMode" = "ForwardBase" }
         // pass for ambient light and first light source
         Cull Front // render only back faces

         CGPROGRAM

         #pragma vertex vert
         #pragma fragment frag

         #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;
         uniform float4 _BackColor;
         uniform float4 _BackSpecColor;
         uniform float _BackShininess;

         struct vertexInput
         {
            float4 vertex : POSITION;
            float3 normal : NORMAL;
         };
         struct vertexOutput
         {
            float4 pos : SV_POSITION;
            float4 posWorld : TEXCOORD0;
            float3 normalDir : TEXCOORD1;
            float4 posInObjectCoords : TEXCOORD2; //测试用，可以删除
         };

         vertexOutput vert(vertexInput input)
         {
            vertexOutput output;

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

            output.posWorld = mul(modelMatrix, input.vertex);
            output.normalDir = normalize(mul(float4(-input.normal, 0.0), modelMatrixInverse).xyz);
            output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
            output.posInObjectCoords = input.vertex; //测试用，可以删除
            return output;
         }

         float4 frag(vertexOutput input) : COLOR
         {
            if (input.posInObjectCoords.y > 0.0) //测试用，可以删除
            {
               discard; // drop the fragment if y coordinate > 0
            }
            float3 normalDirection = normalize(input.normalDir);

            float3 viewDirection = normalize(_WorldSpaceCameraPos - input.posWorld.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 - input.posWorld.xyz;
               float distance = length(vertexToLightSource);
               attenuation = 1.0 / distance; // linear attenuation
               lightDirection = normalize(vertexToLightSource);
            }

            float3 ambientLighting = UNITY_LIGHTMODEL_AMBIENT.rgb * _BackColor.rgb;

            float3 diffuseReflection = attenuation * _LightColor0.rgb * _BackColor.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 * _BackSpecColor.rgb * pow(max(0.0, dot(reflect(-lightDirection, normalDirection), viewDirection)), _BackShininess);
            }

            return float4(ambientLighting + diffuseReflection + specularReflection, 1.0);
         }
         ENDCG
      }

      Pass
      {
         Tags { "LightMode" = "ForwardAdd" }
         // pass for additional light sources
         Blend One One // additive blending
         Cull Front // render only back faces

         CGPROGRAM

         #pragma vertex vert
         #pragma fragment frag

         #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;
         uniform float4 _BackColor;
         uniform float4 _BackSpecColor;
         uniform float _BackShininess;

         struct vertexInput
         {
            float4 vertex : POSITION;
            float3 normal : NORMAL;
         };
         struct vertexOutput
         {
            float4 pos : SV_POSITION;
            float4 posWorld : TEXCOORD0;
            float3 normalDir : TEXCOORD1;
            float4 posInObjectCoords : TEXCOORD2; //测试用，可以删除
         };

         vertexOutput vert(vertexInput input)
         {
            vertexOutput output;

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

            output.posWorld = mul(modelMatrix, input.vertex);
            output.normalDir = normalize(mul(float4(-input.normal, 0.0), modelMatrixInverse).xyz);
            output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
            output.posInObjectCoords = input.vertex; //测试用，可以删除
            return output;
         }

         float4 frag(vertexOutput input) : COLOR
         {
            if (input.posInObjectCoords.y > 0.0) //测试用，可以删除
            {
               discard; // drop the fragment if y coordinate > 0
            }
            float3 normalDirection = normalize(input.normalDir);

            float3 viewDirection = normalize(_WorldSpaceCameraPos - input.posWorld.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 - input.posWorld.xyz;
               float distance = length(vertexToLightSource);
               attenuation = 1.0 / distance; // linear attenuation
               lightDirection = normalize(vertexToLightSource);
            }

            float3 diffuseReflection = attenuation * _LightColor0.rgb * _BackColor.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 * _BackSpecColor.rgb * pow(max(0.0, dot(reflect(-lightDirection, normalDirection), viewDirection)), _BackShininess);
            }

            return float4(diffuseReflection + specularReflection, 1.0);
               // no ambient lighting in this pass
         }
         ENDCG
      }
   }
   // The definition of a fallback shader should be commented out
   // during development:
   // Fallback "Specular"
}
```

在场景中添加两个球体，一个使用 two-sided per-vertex lighting， 另一个使用 two-sided per-pixel lighting，并观察他们的不同，效果如下：


New Material 1
Shader Cg two-sided per-verte

Front Material Diffuse Color
Front Material Specular Color
Front Material Shininess 10
Back Material Diffuse Color
Back Material Specular Color
Back Material Shininess 10

恭喜你，在本章节中你应该了解：

> 1、如何用 per-pixel lighting 渲染 two-sided surfaces。

资源下载地址：点击下载，共下载 12 次。

前一篇：第十二章节：光滑的镜面高光（关于每像素光照）

后一篇：第十四章节：多个灯（关于在一个 pass 中遍历处理多个光源）

赞	打酱油	呵呵	鄙视	正能量
2 人	0 人	0 人	0 人	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