【翻译】第十九章节：纹理层（关于多重纹理）

2014-12-11 08:31:00 1661 人阅读 Unity3D cg 多重纹理 A- A+

文章内容 例子源码 网友评论 最后编辑：2014-12-21 18:36:09

本文永久地址：http://www.omuying.com/article/108.aspx，【文章转载请注明出处！】

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

本教程介绍多重纹理（multitexturing），即在一个着色器中使用多个纹理图像。

本章节是对《纹理球》章节着色器代码的扩展，如果你还没有看过这个教程，你应该先阅读它。

表面层

许多真实的表面（比如人类的皮肤）由不同颜色、透明度、反射率等等的层组成，如果最上面的层是不透明的并且它也不会发出任何光，那么渲染这样的表面一点也不困难，然而在许多情况下最上面的层是（半）透明的，所以我们要精确的在表面渲染就需要考虑多个层。

事实上，我们可以用 Phong reflection model 中的镜面反射来表现一个透明层的光的反射，例如出汗的皮肤、果蜡（wax on fruits）、透明塑料中的颜料颗粒等等，另一方面，顶部透明层下面的层我们可以使用漫反射来表现。

表层的光照不要求层是几何模型，它们是单一、薄的多边形网格就可以了，但是光照计算必须计算不同层之间的反射，同时还要考虑光在层之间的透射（光在层中的进出），Nvidia 演示“Dawn”和“Human Head”例子时使用了这种方法。

具体的细节描述可能超出了本教程的范围，但是这儿需要说明的是，层往往是通过纹理图像来指定他们的表面的特征，在这个教程中，我们只展示如何使用两个纹理，并通过指定的方式来合并这两个纹理，这个例子其实与层无关，但是也表明了多重纹理应用的广泛。

亮或者不亮的地球

如下图所示，由于人类的活动，人工照明使黑夜中的地球（另一边是白天）也并没有全黑：

白天的地球：

最新文章

暂无图片 【原创】C# 基础之Lambda表达式 - 907 次阅读

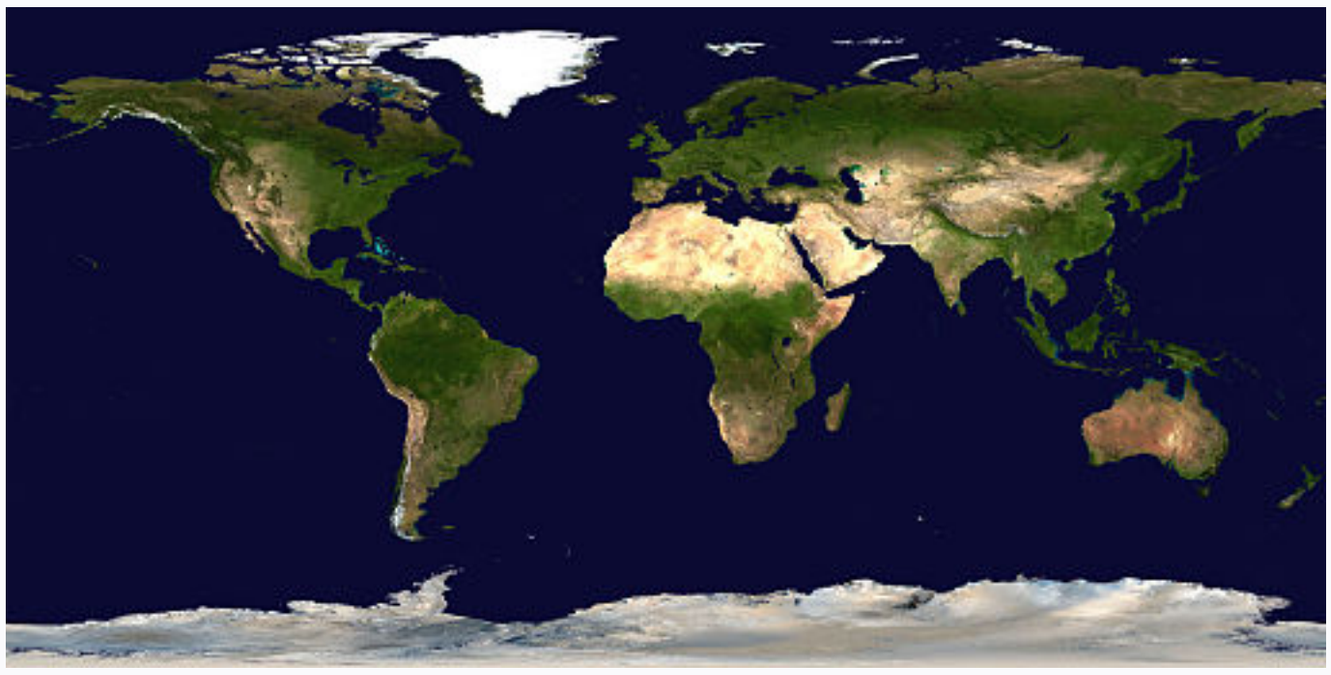
暂无图片 【原创】C#基础之IEnumerable和IEnumerator - 792 次阅读

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

随机阅读



白天地球的漫射光照应该不只是暗淡的纹理图像，它实际上也是混合了无光照的纹理图像，需要注意的是白天的地球远比黑夜中的地球明亮，但是为了表现夜间的纹理我们需要减少这种对比度。

我们需要扩展《纹理球》章节中的着色器来支持两个纹理图像，同时我们还需要使用《漫反射》章节中的漫反射公式：

$$I_{\text{diffuse}} = I_{\text{incoming}}\, k_{\text{diffuse}}\, \max(0, \mathbf{N} \cdot \mathbf{L})$$

根据这个公式，max(0, N·L) 是漫射光的等级（level）levelOfLighting，然后我们基于 levelOfLighting 值来混合白天和夜间的纹理，我们可以在设置片段颜色之前先把 levelOfLighting 乘以白天的纹理颜色与 1.0 - levelOfLighting 乘以黑夜的纹理颜色相加，不过我们可以使用 Cg 内置的函数 lerp（lerp(a, b, w) = b*w + a*(1.0-w)），这样可能效果会更好，因此片段着色器的代码应该是这样：

```
float4 frag(vertexOutput input) : COLOR
{
    float4 nighttimeColor = tex2D(_MainTex, input.tex.xy);
    float4 daytimeColor = tex2D(_DecalTex, input.tex.xy);
    return lerp(nighttimeColor, daytimeColor, input.levelOfLighting);
    // = daytimeColor * levelOfLighting
    // + nighttimeColor * (1.0 - levelOfLighting)
}
```

注意，这个混合与《透明度》章节中的 alpha 混合非常相似，不同的是我们在片段着色器中执行混合时是使用 levelOfLighting 来代替纹理的 alpha 分量（即不透明度），因为纹理应该被混合在其他纹理上面（blended “over”），事实上，如果 _DecalTex 指定 alpha 分量，那么我们可以使用这个 alpha 分量在混合时让 _DecalTex 超过（over）_MainTex，这实际上与 Unity 内置的 Decal 着色器相同，它对应一个有部分透明的不透明层，然后透明部分下面是可见的。

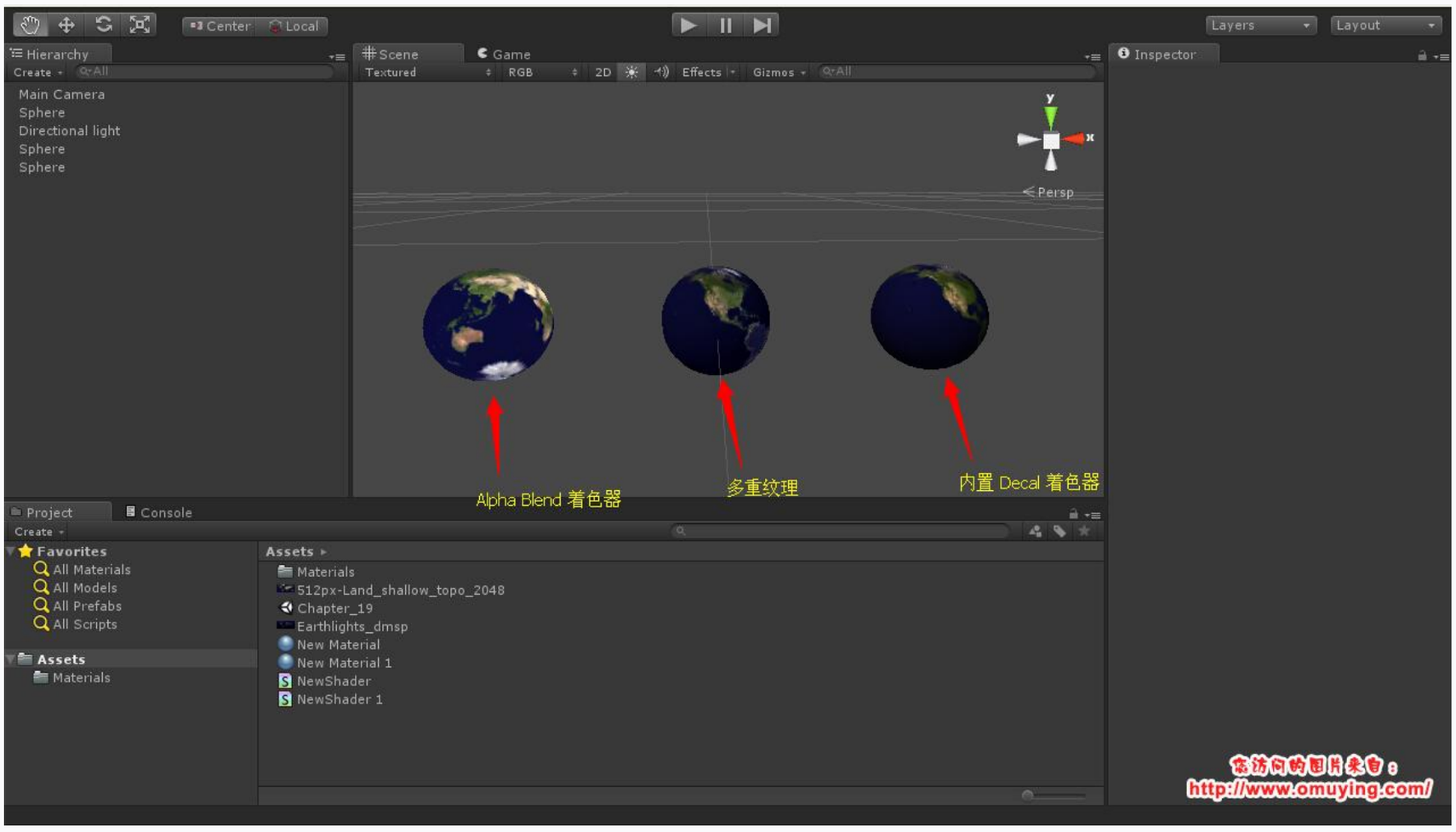
完成着色器代码

着色器的属性名称与 Decal 着色器的属性名称相同，目的是 fallback 可以指定为 Decal，另外额外添加的属性 _Color 用来与夜间的纹理颜色相乘，目的是控制总体的亮度，此外 _LightColor0 与白天的纹理颜色相乘，目的是考虑彩色光源，着色器的代码如下：

```
Shader "Cg multitexturing of Earth"
{
   Properties
   {
      _DecalTex ("Daytime Earth", 2D) = "white" {}
      _MainTex ("Nighttime Earth", 2D) = "white" {}
      _Color ("Nighttime Color Filter", Color) = (1,1,1,1)
   }
   SubShader
   {
      Pass
      {
         Tags { "LightMode" = "ForwardBase" }
         // pass for the first, directional light

         CGPROGRAM

         #pragma vertex vert
         #pragma fragment frag

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

         uniform sampler2D _MainTex;
         uniform sampler2D _DecalTex;
         uniform float4 _Color;
```

```
            struct vertexInput
            {
                float4 vertex : POSITION;
                float3 normal : NORMAL;
                float4 texcoord : TEXCOORD0;
            };
            struct vertexOutput
            {
                float4 pos : SV_POSITION;
                float4 tex : TEXCOORD0;
                float levelOfLighting : TEXCOORD1;
                // level of diffuse lighting computed in vertex shader
            };

            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 lightDirection = normalize(_WorldSpaceLightPos0.xyz);

                output.levelOfLighting = max(0.0, dot(normalDirection,
lightDirection));
                output.tex = input.texcoord;
                output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
                return output;
            }

            float4 frag(vertexOutput input) : COLOR
            {
                float4 nighttimeColor = tex2D(_MainTex, input.tex.xy);
                float4 daytimeColor = tex2D(_DecalTex, input.tex.xy);
                return lerp(nighttimeColor, daytimeColor, input.levelOfLighting);
                // = daytimeColor * levelOfLighting
                // + nighttimeColor * (1.0 - levelOfLighting)
            }
            ENDCG
        }
    }
    // The definition of a fallback shader should be commented out
    // during development:
    // Fallback "Decal"
}
```

当你运行这个着色器时，请确保你的场景中有一个激活的方向光，另外我们使用多重纹理着色器与 alpha 混合、内置 Decal 着色器进行比较，查看效果如图：



恭喜你，在本章节中你应该了解：

> 1、表层如何影响材质外观。
>
> 2、在地球纹理的球体中黑夜面考虑人工照明。
>
> 3、如果在着色器中实现球体上的地球纹理。
>
> 4、alpha 纹理与不透明纹理混合（over）。

资源下载地址：点击下载，共下载 24 次。

前一篇：第十八章节：透明纹理（关于在片段擦除、混合中使用 alpha 纹理）

后一篇：第二十章节：凹凸表面光照（关于法线贴图）

赞 2 人　打酱油 0 人　呵呵 0 人　鄙视 0 人　正能量 0 人