【翻译】第十五章节：纹理球（关于纹理球面）

2014-12-07 08:08:00 1907 人阅读 Unity3D cg 纹理球

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

本教程介绍纹理贴图。

这是在系列文章中第一篇介绍 Unity 着色器纹理的教程，在这个教程中，我们开始在一个球体上面使用单一纹理贴图，在这个例子中，我们把地球图像映射到一个球体表面。在本教程的基础上，后面的教程会包括纹理表面光照，透明纹理、多纹理、光泽贴图等等。

纹理映射

“纹理映射”的基本思路是映射一个图像到三角形网格，换句话说，就是把一个平面图像平铺到三维物体表面。

“纹理坐标”的意思简单的说就是指在纹理中的位置，水平坐标用 S 表示，垂直坐标用 T 表示，然而更常见的还是把他们称为 x 和 y，在动画和建模工作中，纹理坐标通常称为 U 和 V。

为了把纹理图像映射到网格中，网格中的每个顶点都被赋予一对纹理坐标（这个过程（或结果）有时候也称为“UV 映射”，因为每个顶点都对应 UV 空间中的一个点）。因此，每个顶点都对应纹理图像中的一个点，然后顶点的纹理坐标在三角形的每个点上进行插值，这样网格中的每个三角形上的每个点都有一对（插值）的纹理坐标，网格中的每个纹理坐标都对应纹理贴图中的一个位置，然后根据这个位置来获取颜色数据，因此在一个网格上渲染纹理贴图包括两步：

> 1、插值纹理坐标。
>
> 2、根据插值的纹理坐标在纹理图像相应的位置查找颜色值。

一般情况下，任何一个有效的浮点数都是一个有效的纹理坐标，然而当 GPU 在纹理图像（tex2D 指令）上查找一个像素（纹理像素）时，如果在导入纹理时指定“Wrap Mode”模式，它将会内部映射纹理坐标的范围在 0 至 1 之间，当 Warp Mode 为“repeat”时，基本上使用纹理坐标的小数部分来确定纹理坐标的值在 0 至 1范围内，另一方面当 Warp Mode 为"clamp"时，则限制纹理坐标在 0 至 1 之间，纹理坐标的范围在 0 至 1 之间是为了确定在纹理图像的位置：（0，0）表示纹理图像的左下角，（1，0）表示纹理图像的右下角，（0，1）表示纹理图像的左上角等等。

Unity 中在球上设置纹理

在 Unity 中要把地球图像映射到一个球体上，你必须先把图像导入到 Unity 中，从这儿下载并保存图片到你的计算机中，保存之后切换到 Unity 窗口中，选择 Assets -> Import New Asset... 菜单，在文件选择框中选中刚才保存的图片，然后单击导入，这样新导入的纹理图片会显示在 Project 视图中，接着选中新导入的纹理图片，这时你会在 Inspector 视图看到导入图片的详细信息，如图：

最新文章

暂无图片 【原创】C# 基础之Lambda表达式 - 907 次阅读

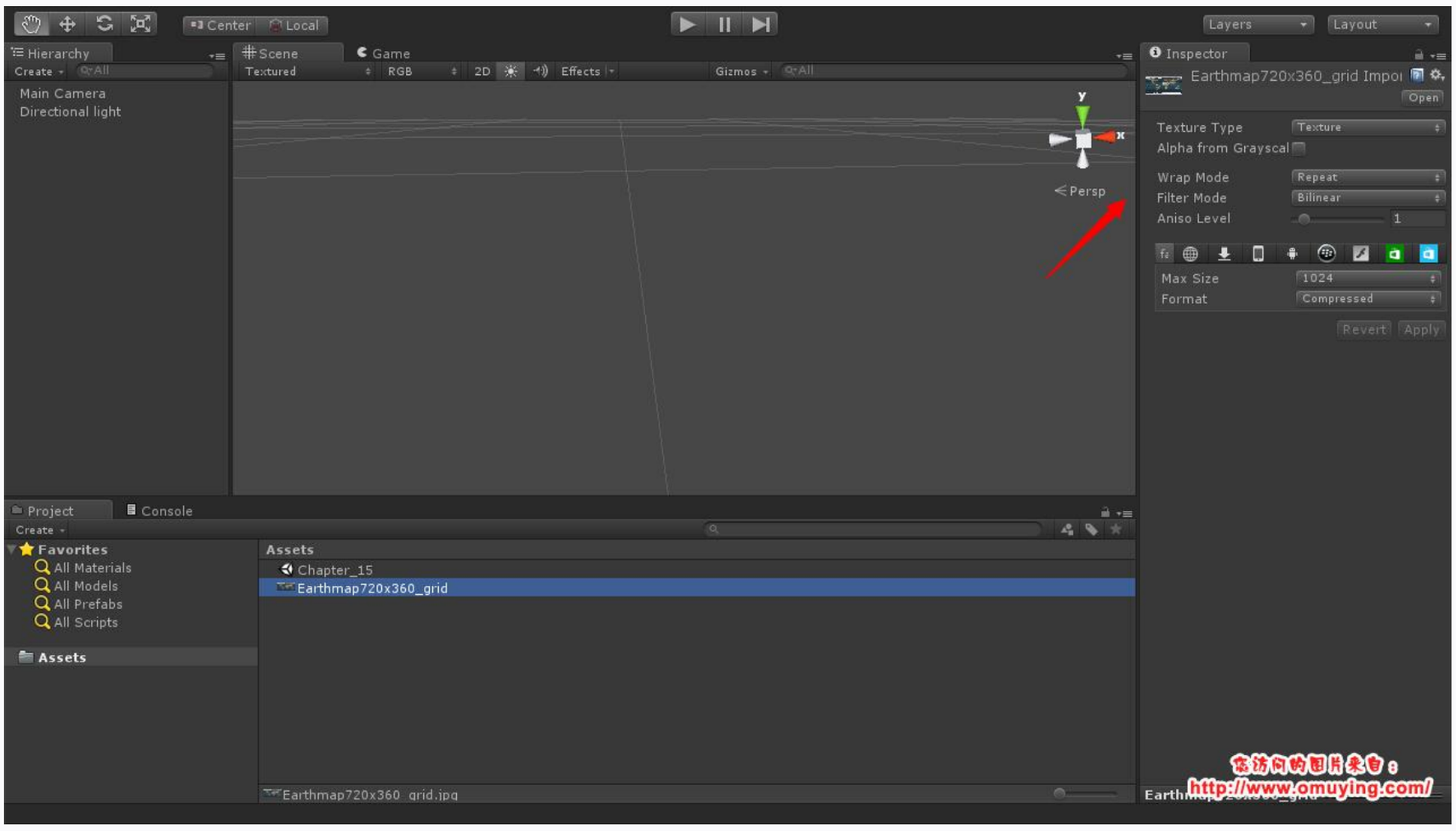
暂无图片 【原创】C#基础之IEnumerable和IEnumerator - 792 次阅读

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



创建球体、Shader、Material 可以查看《最简单的着色器》章节，然后使用下面的着色器的代码：

```
Shader "Cg shader with single texture"
{
    Properties
    {
        _MainTex ("Texture Image", 2D) = "white" {}
        // a 2D texture property that we call "_MainTex", which should
        // be labeled "Texture Image" in Unity's user interface.
        // By default we use the built-in texture "white"
        // (alternatives: "black", "gray" and "bump").
    }
    SubShader
    {
        Pass
        {
            CGPROGRAM

            #pragma vertex vert
            #pragma fragment frag

            uniform sampler2D _MainTex;
            // a uniform variable refering to the property above
            // (in fact, this is just a small integer specifying a
            // "texture unit", which has the texture image "bound"
            // to it; Unity takes care of this).

            struct vertexInput
            {
                float4 vertex : POSITION;
                float4 texcoord : TEXCOORD0;
            };
            struct vertexOutput
            {
                float4 pos : SV_POSITION;
                float4 tex : TEXCOORD0;
            };

            vertexOutput vert(vertexInput input)
            {
                vertexOutput output;

                output.tex = input.texcoord;
                // Unity provides default longitude-latitude-like
                // texture coordinates at all vertices of a
                // sphere mesh as the input parameter
                // "input.texcoord" with semantic "TEXCOORD0".
                output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
                return output;
            }
            float4 frag(vertexOutput input) : COLOR
            {
                return tex2D(_MainTex, input.tex.xy);
                //return tex2D(_MainTex, input.tex.xy);
                // look up the color of the texture image specified by
                // the uniform "_MainTex" at the position specified by
                // "input.tex.x" and "input.tex.y" and return it

            }

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

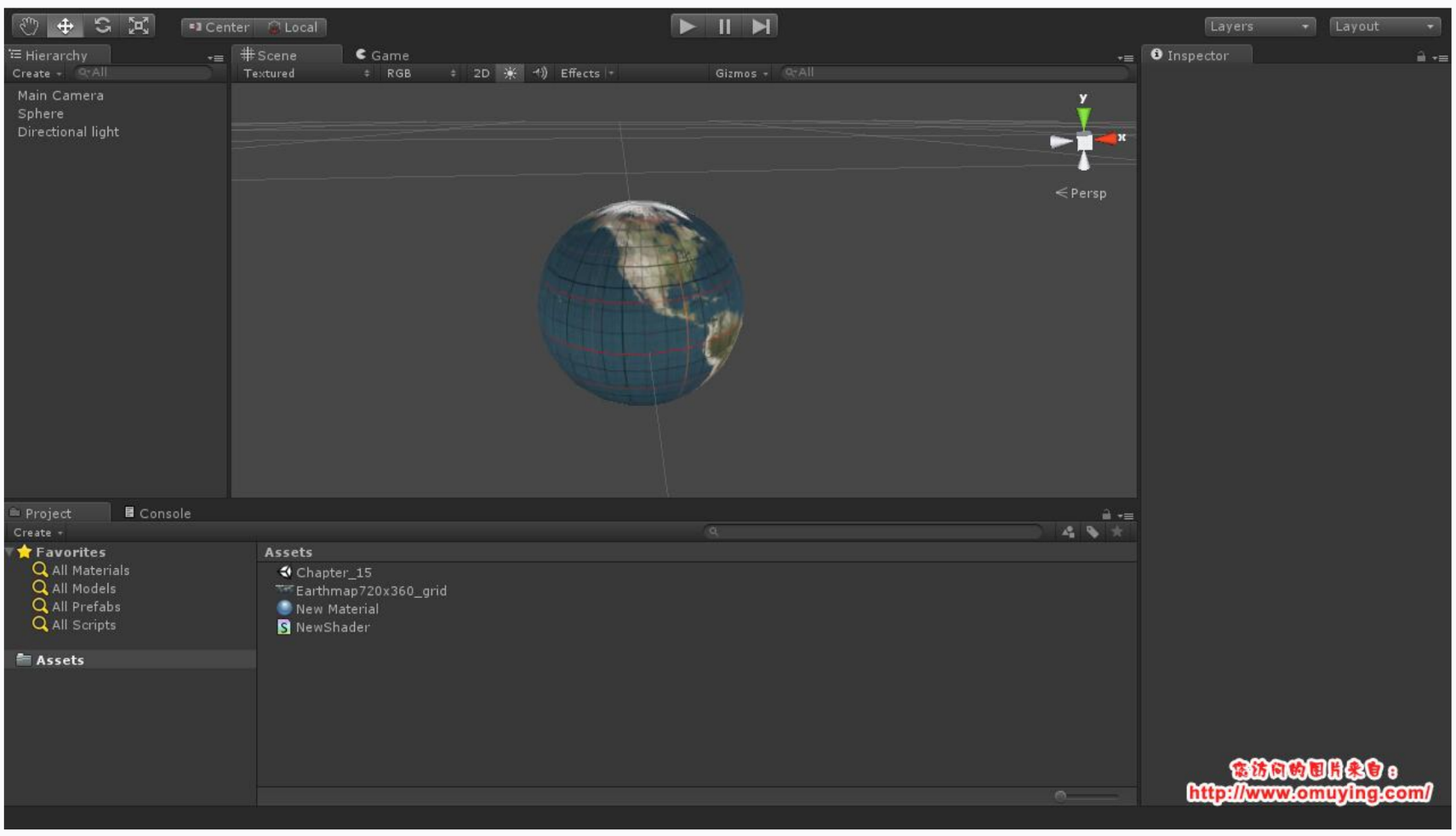
```
64      // Fallback "Unlit/Texture"
65  }
```

需要注意的是我们需要在着色器中指定 _MainTex 属性，因为如果当着色器编译失败时，程序会使用 fallback 调用 Unlit/Texture 着色器。

球现在应该是白色的，如果是灰色，你应该检查是否给球体应用了正确的 Material，以及是否给 Material 应用了正确的 Shader，此外还应该检查着色器的代码是否存在错误。

在 Hierarchy 视图或者 Scene 视图选中球体，然后会在 Inspector 显示球体的详细信息，点击 Mesh Renderer 选项下的 material，这时会在下面显示 Texture Image 标签，最后把导入的图片拖放到 _MainTex 属性上，或者你也可以通过点击 _MainTex 属性右下角的按钮，然后从弹出的窗口中选择导入的图片。

如果一切顺利，纹理图像应该显示在球体上，效果如图：



它是如何工作的？

由于许多技术会使用纹理映射，回顾我们的着色器代码，我们看看这个着色器工作的原理：

Unity 球体对象上的每个顶点的纹理坐标由顶点输入参数中的 texcoord（语义词：TEXCOORD0）指定，这个坐标类似于经度和纬度（但是范围是 0 至 1），另外它还与顶点输入参数中的语义词 POSITION 相似，POSITION 指定对象空间中的位置，而 texcoord 则指定纹理图像中的纹理坐标。

顶点着色器把每个顶点的纹理坐标写到顶点输出参数 output.tex 中，对于每个三角形片段，首先在三角形的顶点之间插值，然后把这些插值的纹理坐标作为顶点着色器的输出参数传递给片段着色器的输入参数，接着片段着色器使用这些已插值的纹理坐标在 uniform _MainTex 指定的纹理图像上查找颜色，最后这个颜色作为片段输出参数被写入到帧缓冲区以及被显示在屏幕上。

以上这些，将对你理解其他教程中更为复杂的纹理映射是至关重要的。

重复和移动纹理

在 Unity 的着色器界面，你可能已经注意到 Tiling 和 Offset 参数以及他们 x 和 y 属性，在内置的着色器中，这些参数允许你重复纹理（在纹理坐标空间缩放纹理图像）和在表面上移动纹理图像（在纹理坐标空间偏移纹理图像），为此我们还需要定义其他的 uniform 属性：

```
1  uniform float4 _MainTex_ST;
2  // tiling and offset parameters of property "_MainTex"
```

对于每个纹理属性，Unity 通常使用一个以“_ST”结尾的 float4 uniform（记住：“S”和“T”是纹理坐标的正式名称，不过我们通常称呼“U”和“V”或者“x”和“y”），Tiling 参数的 x 和 y 属性可以通过 _MainTex_ST.x 和 _MainTex_ST.y 来获得，而 Offset 参数的 x 和 y 属性需要通过 _MainTex_ST.w 和

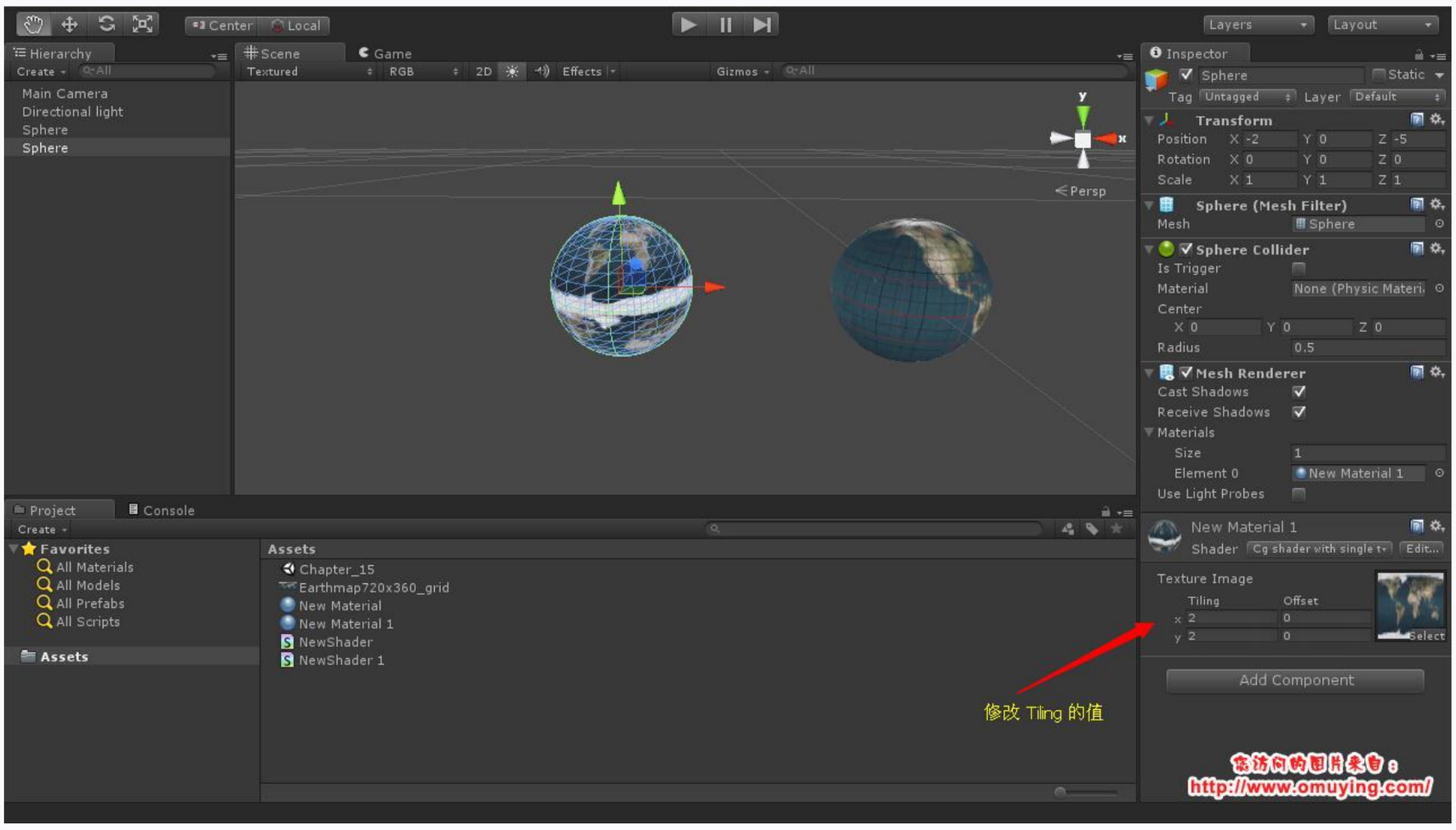
_MainTex_ST.z 来获得，所以使用代码看起来像这样：

```
return tex2D(_MainTex, _MainTex_ST.xy * input.tex.xy + _MainTex_ST.zw);
```

这样着色器可以像内置着色器那样使用 Tiling 和 Offset，但是在其他的教程中，为了保持代码的清洁这个功能通常我们不会实现，最终的着色器代码如下：

```
Shader "Cg shader with single texture 1"
{
   Properties
   {
      _MainTex ("Texture Image", 2D) = "white" {}
      // a 2D texture property that we call "_MainTex", which should
      // be labeled "Texture Image" in Unity's user interface.
      // By default we use the built-in texture "white"
      // (alternatives: "black", "gray" and "bump").
   }
   SubShader
   {
      Pass
      {
         CGPROGRAM

         #pragma vertex vert
         #pragma fragment frag

         uniform sampler2D _MainTex;
         // a uniform variable refering to the property above
         // (in fact, this is just a small integer specifying a
         // "texture unit", which has the texture image "bound"
         // to it; Unity takes care of this).
         uniform float4 _MainTex_ST;
         // tiling and offset parameters of property "_MainTex"

         struct vertexInput
         {
            float4 vertex : POSITION;
            float4 texcoord : TEXCOORD0;
         };
         struct vertexOutput
         {
            float4 pos : SV_POSITION;
            float4 tex : TEXCOORD0;
         };

         vertexOutput vert(vertexInput input)
         {
            vertexOutput output;

            output.tex = input.texcoord;
            // Unity provides default longitude-latitude-like
            // texture coordinates at all vertices of a
            // sphere mesh as the input parameter
            // "input.texcoord" with semantic "TEXCOORD0".
            output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
            return output;
         }
         float4 frag(vertexOutput input) : COLOR
         {
            return tex2D(_MainTex, _MainTex_ST.xy * input.tex.xy +
_MainTex_ST.zw);
         }

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

然后我们把这个着色器应用到一个球体对象上，然后修改 Tiling 和 Offset 的值，效果如图：



恭喜你，在本章节中你应该了解：

> 1、如何导入纹理图像以及如何给着色器的纹理属性附加纹理图像。
>
> 2、顶点着色器与片段着色器如何协同把纹理图像映射到网格。
>
> 3、Unity 内置着色器的 tiling 与 offset 作用，以及如何实现它们。

资源下载地址：点击下载，共下载 26 次。

前一篇：第十四章节：多个灯（关于在一个 pass 中遍历处理多个光源）

后一篇：第十六章节：表面纹理光照（关于纹理漫射光照）

赞	打酱油	呵呵	鄙视	正能量
6 人	0 人	1 人	0 人	0 人

清茶一盏

非常好的网站~支持

2015年5月26日 回复 顶 转发

.最终幻想.

回复 清茶一盏: 谢谢回复！

2015年6月1日 回复 顶 转发

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