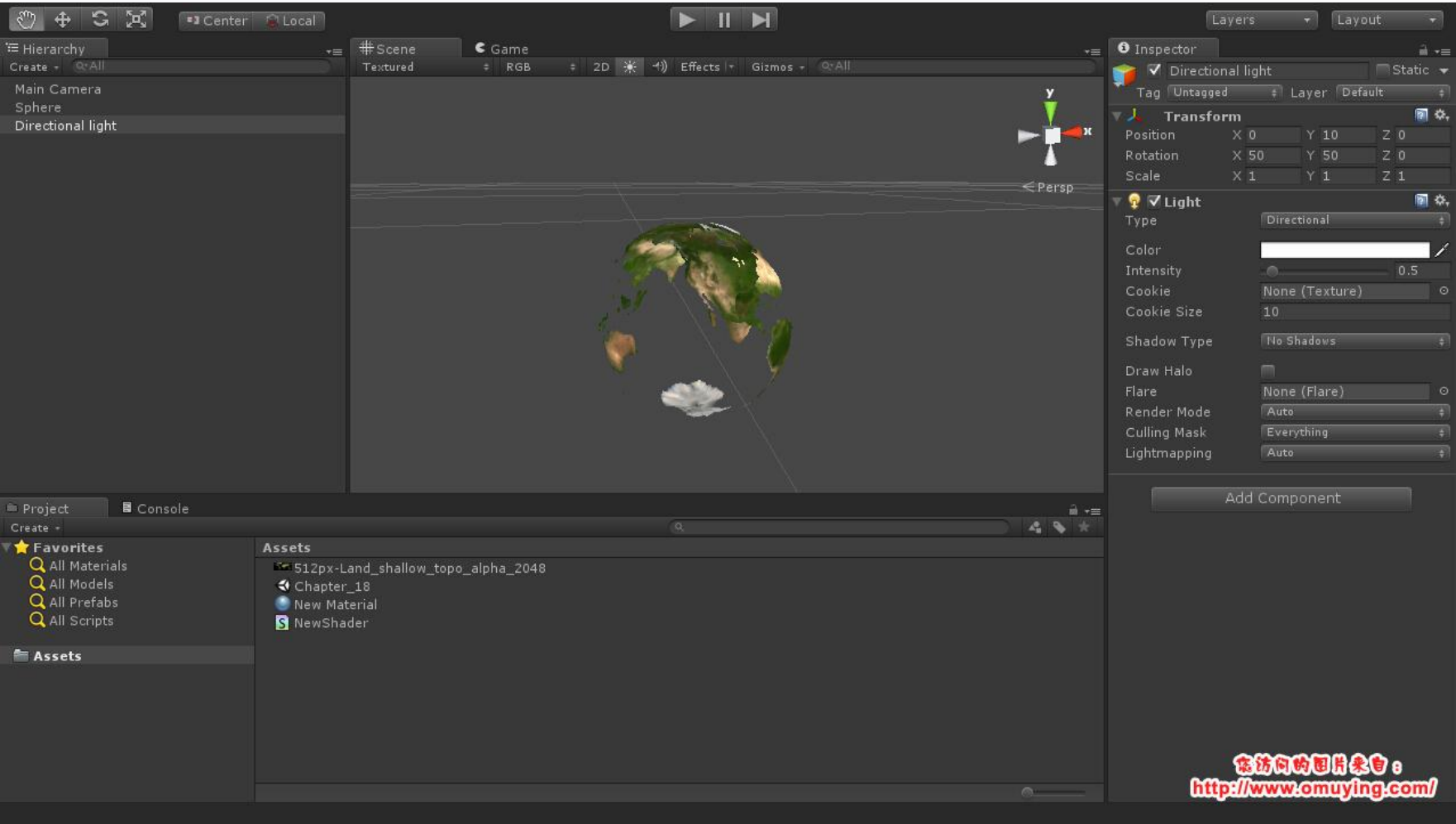


$$A^- \quad A^+$$

随机阅读

```
32     };
33
34     vertexOutput vert(vertexInput input)
35     {
36         vertexOutput output;
37
38         output.tex = input.texcoord;
39         output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
40         return output;
41     }
42
43     float4 frag(vertexOutput input) : COLOR
44     {
45         float4 textureColor = tex2D(_MainTex, input.tex.xy);
46         if (textureColor.a < _Cutoff) // alpha value less than user-
specified threshold?
47         {
48             discard; // yes: discard this fragment
49         }
50         return textureColor;
51     }
52     ENDCG
53 }
54 }
55 // The definition of a fallback shader should be commented out
56 // during development:
57 // Fallback "Unlit/Transparent Cutout"
58 }
```

首先片段着色器读取 RGBA 纹理，然后用纹理的 alpha 值与用户指定的阈值进行比较，如果 alpha 值低于阈值，那么就会把片段擦除，这样表面也会变得透明，效果如图：



注意，discard 指令在某些平台上相当消耗性能，尤其是在移动平台上，因此我们通常使用混合（blending）来代替。

混合

在《透明度》章节中，已经描述了如何使用 alpha 混合来渲染半透明的对象，这儿我们在着色器中使用 RGBA 纹理，代码如下：

```
001 Shader "Cg texturing with alpha blending"
002 {
003     Properties
004     {
005         _MainTex ("RGBA Texture Image", 2D) = "white" {}
006     }
007     SubShader
008     {
009         Tags {"Queue" = "Transparent"}
010
011         Pass
012         {
013             Cull Front // first render the back faces
014             ZWrite Off // don't write to depth buffer
015             // in order not to occlude other objects
016             Blend SrcAlpha OneMinusSrcAlpha
017             // blend based on the fragment's alpha value
018
019             CGPROGRAM
020 }
```

暂无图片

【翻译】第十五章节：纹理球（关于纹理球面） - 1906 次阅读

暂无图片

【翻译】第二十五章节：弧形玻璃（关于折射贴图） - 1722 次阅读

暂无图片

【翻译】第十二章节：光滑的镜面高光（关于每像素光照） - 1180 次阅读

暂无图片

【翻译】第二十一章节：凹凸表面投影（关于视差贴图） - 1462 次阅读

暂无图片

【翻译】第十一章节：双面表面（关于双面每顶点光照） - 1186 次阅读

```

021     #pragma vertex vert
022     #pragma fragment frag
023
024     uniform sampler2D _MainTex;
025     uniform float _Cutoff;
026
027     struct vertexInput
028     {
029         float4 vertex : POSITION;
030         float4 texcoord : TEXCOORD0;
031     };
032     struct vertexOutput
033     {
034         float4 pos : SV_POSITION;
035         float4 tex : TEXCOORD0;
036     };
037
038     vertexOutput vert(vertexInput input)
039     {
040         vertexOutput output;
041
042         output.tex = input.texcoord;
043         output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
044         return output;
045     }
046
047     float4 frag(vertexOutput input) : COLOR
048     {
049         return tex2D(_MainTex, input.tex.xy);
050     }
051
052     ENDCG
053 }
054
055 Pass
056 {
057     Cull Back // now render the front faces
058     ZWrite Off // don't write to depth buffer
059     // in order not to occlude other objects
060     Blend SrcAlpha OneMinusSrcAlpha
061     // blend based on the fragment's alpha value
062
063     CGPROGRAM
064
065     #pragma vertex vert
066     #pragma fragment frag
067
068     uniform sampler2D _MainTex;
069     uniform float _Cutoff;
070
071     struct vertexInput
072     {
073         float4 vertex : POSITION;
074         float4 texcoord : TEXCOORD0;
075     };
076     struct vertexOutput
077     {
078         float4 pos : SV_POSITION;
079         float4 tex : TEXCOORD0;
080     };
081
082     vertexOutput vert(vertexInput input)
083     {
084         vertexOutput output;
085
086         output.tex = input.texcoord;
087         output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
088         return output;
089     }
090
091     float4 frag(vertexOutput input) : COLOR
092     {
093         return tex2D(_MainTex, input.tex.xy);
094     }
095     ENDCG
096 }
097 }
098 // The definition of a fallback shader should be commented out
099 // during development:
100 // Fallback "Unlit/Transparent"
101 }

```

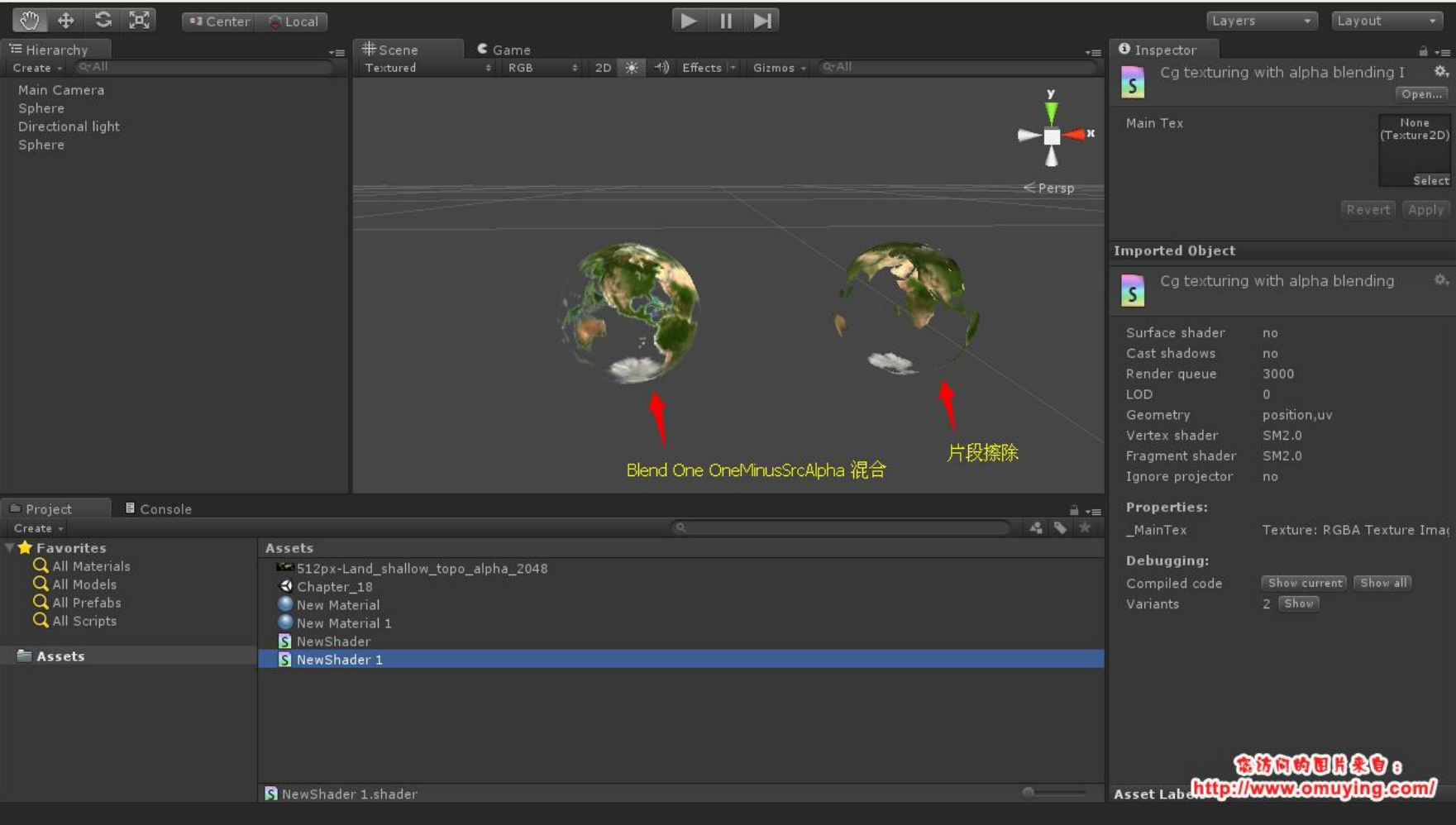
这个着色器的效果如图：



在这个特殊的纹理图像中，当纹理颜色的 alpha 值为 0 时，纹元（texels）是黑的，其实这个纹理图像中颜色都与它们 alpha 值预乘（有时候这种颜色称为不透明权值“opacity-weighted”），因此对于这个特殊的图像，为了避免在混合公式中再次与纹理颜色的 alpha 值相乘，我们应该在混合公式指定纹理颜色已被预乘过，不过我们可以使用下面的混合公式来改善上面的着色器（所有 pass 中的都要修改）：

```
1 | Blend One OneMinusSrcAlpha
```

改善混合的效果如图：



与自定义颜色混合

到这儿我们还不应该这么快就结束本教程，下面的地球图像中用的是半透明的蓝色海洋，如图：



我们可以使用下面的着色器来实现半透明的海洋，这个着色器的思想是忽略了纹理贴图的 RGB 颜色，然后基于纹理颜色的 alpha 值来指定用什么样的颜色值代替纹理贴图的 RGB 颜色，着色器的代码如下：

```
001 | Shader "Cg semitransparent colors based on alpha"
002 | {
003 |     Properties
```

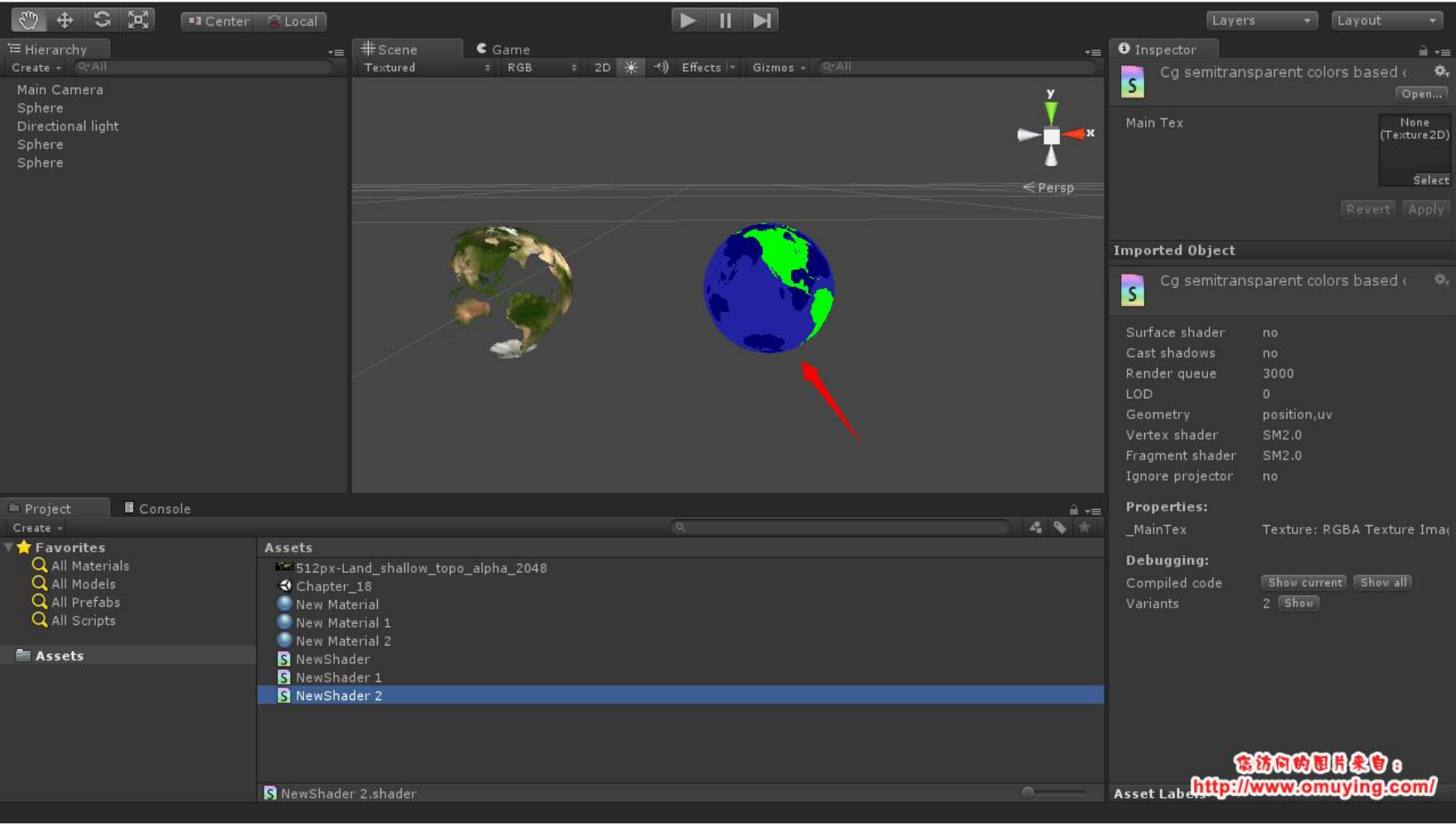
```

004 {
005     _MainTex ("RGBA Texture Image", 2D) = "white" {}
006 }
007 SubShader
008 {
009     Tags {"Queue" = "Transparent"}
010
011     Pass
012     {
013         Cull Front // first render the back faces
014         ZWrite Off // don't write to depth buffer
015         // in order not to occlude other objects
016         Blend SrcAlpha OneMinusSrcAlpha
017         // blend based on the fragment's alpha value
018
019         CGPROGRAM
020
021         #pragma vertex vert
022         #pragma fragment frag
023
024         uniform sampler2D _MainTex;
025         uniform float _Cutoff;
026
027         struct vertexInput
028         {
029             float4 vertex : POSITION;
030             float4 texcoord : TEXCOORD0;
031         };
032         struct vertexOutput
033         {
034             float4 pos : SV_POSITION;
035             float4 tex : TEXCOORD0;
036         };
037
038         vertexOutput vert(vertexInput input)
039         {
040             vertexOutput output;
041
042             output.tex = input.texcoord;
043             output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
044             return output;
045         }
046
047         float4 frag(vertexOutput input) : COLOR
048         {
049             float4 color = tex2D(_MainTex, input.tex.xy);
050             if (color.a > 0.5) // opaque back face?
051             {
052                 color = float4(0.0, 0.0, 0.2, 1.0);
053                 // opaque dark blue
054             }
055             else // transparent back face?
056             {
057                 color = float4(0.0, 0.0, 1.0, 0.3);
058                 // semitransparent green
059             }
060             return color;
061         }
062
063         ENDCG
064     }
065
066     Pass
067     {
068         Cull Back // now render the front faces
069         ZWrite Off // don't write to depth buffer
070         // in order not to occlude other objects
071         Blend SrcAlpha OneMinusSrcAlpha
072         // blend based on the fragment's alpha value
073
074         CGPROGRAM
075
076         #pragma vertex vert
077         #pragma fragment frag
078
079         uniform sampler2D _MainTex;
080         uniform float _Cutoff;
081
082         struct vertexInput
083         {
084             float4 vertex : POSITION;
085             float4 texcoord : TEXCOORD0;
086         };
087         struct vertexOutput {
088             float4 pos : SV_POSITION;
089             float4 tex : TEXCOORD0;
090         };
091
092         vertexOutput vert(vertexInput input)
093         {
094             vertexOutput output;
095
096             output.tex = input.texcoord;

```

```
097         output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
098         return output;
099     }
100
101     float4 frag(vertexOutput input) : COLOR
102     {
103         float4 color = tex2D(_MainTex, input.tex.xy);
104         if (color.a > 0.5) // opaque front face?
105         {
106             color = float4(0.0, 1.0, 0.0, 1.0);
107             // opaque green
108         }
109         else // transparent front face
110         {
111             color = float4(0.0, 0.0, 1.0, 0.3);
112             // semitransparent dark blue
113         }
114         return color;
115     }
116
117     ENDCG
118 }
119
120 // The definition of a fallback shader should be commented out
121 // during development:
122 // Fallback "Unlit/Transparent"
123 }
```

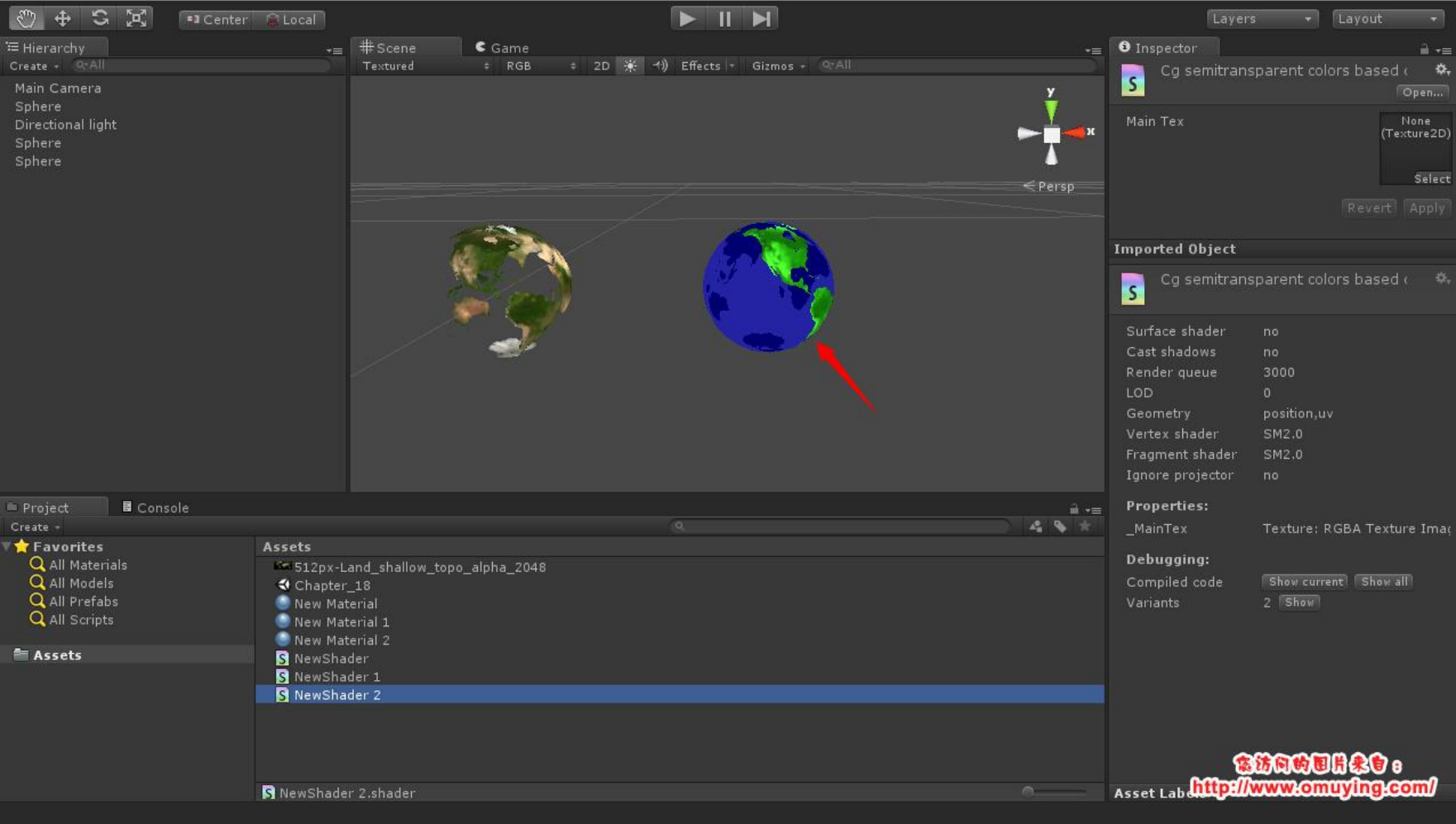
这段着色器的效果如图：



当然你还可以给这个着色器添加灯光以及轮廓加强，为了加深纹理颜色，你还可以修改不透明度、绿色值：

```
1 | color = float4(0.5 * color.r, 2.0 * color.g, 0.5 * color.b, 1.0);
```

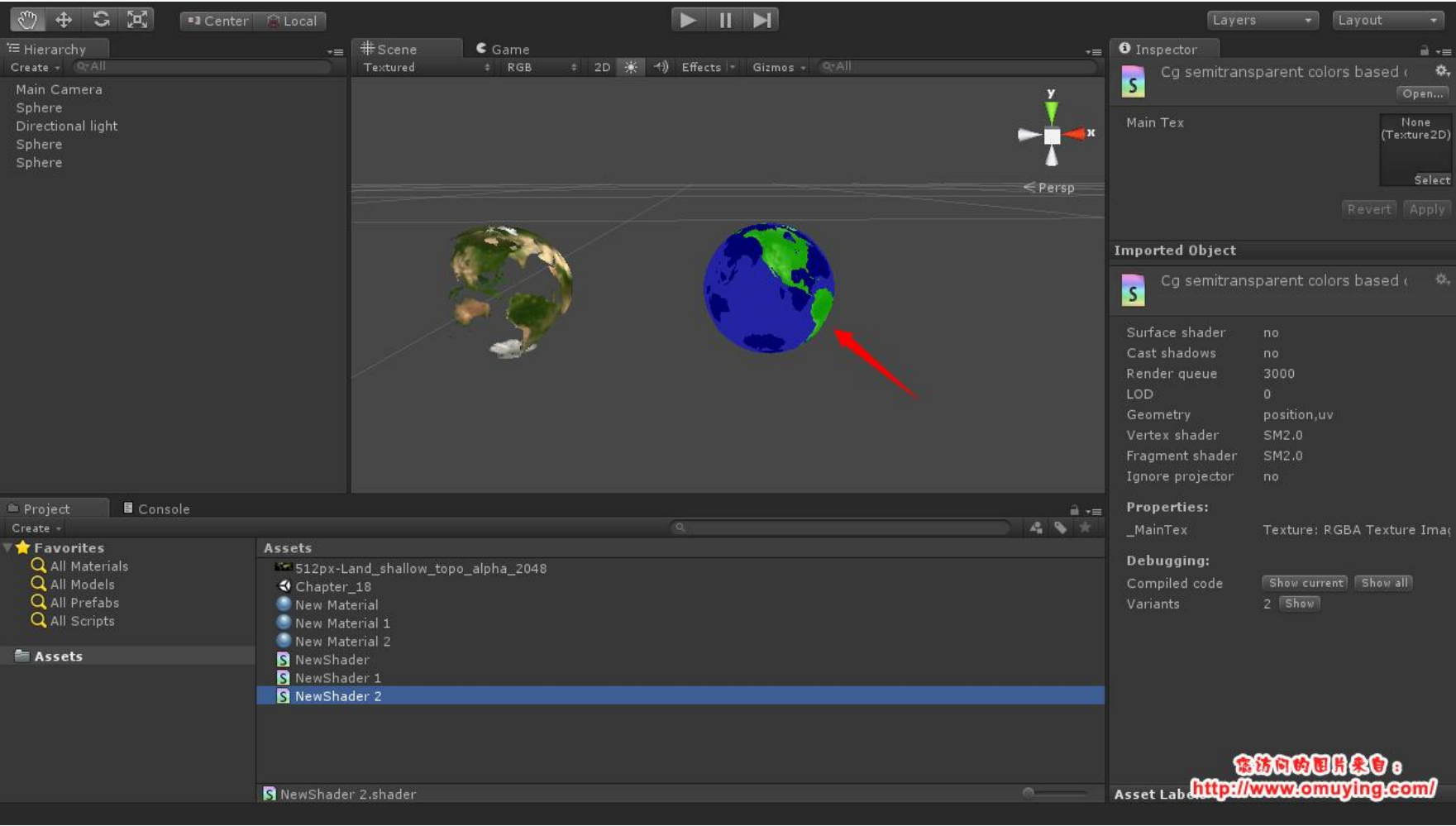
修改绿色值之后的效果如图：



其中绿色分量通过乘以 2 来加深，红色、蓝色分量通过乘以 0.5 来减淡，但是这样会导致绿色值过于饱和，不过我们可以用下面的公式来改善：

```
1 | color = float4(0.5 * color.r, 1.0 - 0.5 * (1.0 - color.g), 0.5 * color.b, 1.0);
```

改善之后的效果如图：



在实践中，人们会尝试各种可能性，例如颜色变换，另外还可以指定用户自定义属性来提高着色器的使用价值（比如上面的因子 0.5）。

恭喜你，在本章节中你应该了解：

- 1、片段擦除与 alpha 纹理贴图结合。
- 2、如何在 blending 中使用 alpha 纹理贴图。
- 3、如何用颜色表现 alpha 纹理贴图。

资源下载地址：[点击下载](#)，共下载 20 次。

前一篇：[第十七章节：光泽纹理（关于光泽贴图）](#)

后一篇：[第十九章节：纹理层（关于多重纹理）](#)



赞
6 人



打酱油
1 人



呵呵
0 人



鄙视
0 人



正能量
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

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