[2024.02.27]Unity中的基础透明度

前言

在实时渲染中，要实现透明效果，通常会在渲染模型时控制它的透明通道。当开启透明混合后，当一个物体被渲染到屏幕上时，每个片元除了颜色值和深度值之外，还有另一个属性——透明度。当透明度为1时，表示该像素是完全不透明的，而当其为0时，则表示该像素完全不会显示。

在Unity中，我们通常使用两种方法来实现透明效果：第一种是使用透明度测试（Alpha Test），这种方法其实无法得到真正的半透明效果；另一种是透明度混合（Alpha Blending）。

当场景中包含很多模型时，对于不透明（opaque）物体，不考虑他们的渲染顺序也能得到正确的排序效果，这是由于强大的深度缓冲（depth buffer，或称zbuffer）的存在。zbuffer是用于解决可见性问题的，它可以决定哪个物体的哪些部分会被渲染在前面，而哪些部分会被其他物体遮挡。它的基本思想是：根据深度缓存中的值来判断该片元距离摄像机的距离，当渲染一个片元时，需要把它的深度值和已经存在于深度缓冲中的值进行比较（如果开启了深度测试），如果它的值距离摄像机更远，那么说明这个片元不应该被渲染到屏幕上（有物体挡住了它）；否则，这个片元应该覆盖掉此时颜色缓冲中的像素值，并把它的深度值更新到深度缓冲中（如果开启了深度写入）。

zbuffer可以让我们不用关心不透明物体的渲染顺序，但如果想要实现透明效果，使用透明度混合时，我们要关闭深度写入（ZWrite），透明度测试和透明度混合的基本原理如下：

1. 透明度测试：它的机制是，当某个片元的透明度不满足要求时（通常是小于某个阈值），那么它对应的片元就会被舍弃。被舍弃的片元将不会再进行任何处理，也不会对颜色缓冲产生任何影响；满足要求的片元就会按照普通的不透明物体的处理方式来处理它，也会进行深度测试、深度写入等。透明度测试不需要关闭深度写入，它和其他的不透明物体的不同就是，它会根据透明度来舍弃一些片元，要么完全透明，要么完全不透明。
2. 透明度混合：这种方法可以得到真正的半透明效果。它会使用当前片元的透明度作为混合因子，与已经存储在颜色缓冲中的颜色值进行混合，得到新的颜色。但是，透明度混合需要关闭深度写入，这就需要注意物体的渲染顺序了。同时，在透明度混合中只关闭了深度写入，但没关闭深度测试。这意味着，当使用透明度混合渲染一个片元时，还是会比较它的深度值与当前深度缓冲中的深度值，如果它的深度值距离摄像机更远，那么就不再进行混合操作。这一点决定了，当一个不透明物体出现在一个透明物体的前面，而我们先渲染了不透明物体，它仍然可以正常地遮挡住透明物体。即，对于透明度混合来说，深度缓冲是只读的。

渲染顺序

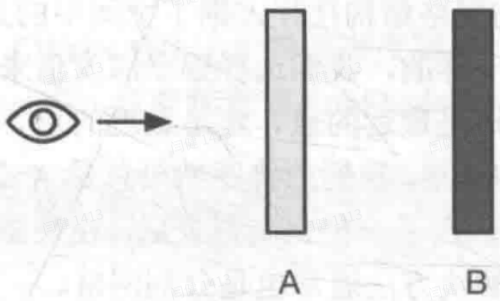
在进行透明度混合时，如果不关闭深度写入，则一个半透明表面背后的表面，本来应该是可见的，但由于深度测试时，判断其背后的表面在zbuffer中，处于更深的深度，导致其被提出，我们也就无法通过半透明表面看到后面的物体了。由此可见，关闭深度写入导致渲染顺序将变得非常重要。

假设对于半透物体A和不透物体B：

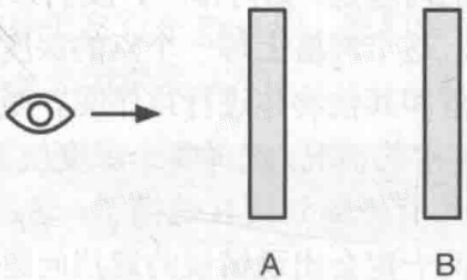
1. 假如先渲染B，再渲染A，则B会先写入深度缓冲和颜色缓冲，随后再写入A，透明物体仍然会进行深度测试，当A距离摄像机更近时，我们会使用A的透明度来和颜色缓冲中的B的颜色进行混合，得到正确的半透明效果。
2. 假如先渲染A，再渲染B，则渲染A时，深度缓冲区中没有任何有效数据，则A直接写入颜色缓冲，但半透物体关闭了深度写入，则A不会修改深度值。当渲染B时，由于没有任何深度数据，B会直接写入深度缓冲和颜色缓冲，结果就是B覆盖了A的颜色。

又假设，A和B都是半透物体，则：

1. 先渲染B再渲染A，则B会正常写入颜色缓冲，然后A会和颜色缓冲中的B进行颜色混合，得到正确的半透效果。
2. 先渲染A再渲染B，则A会先写入颜色缓冲，随后B会和颜色缓冲中的A进行混合，这样混合结果会完全反过来，表现就是B在A的前面，得到错误的半透明结果。



▲图 8.1　场景中有两个物体，其中 A（黄色）是半透明物体，B（紫色）是不透明物体

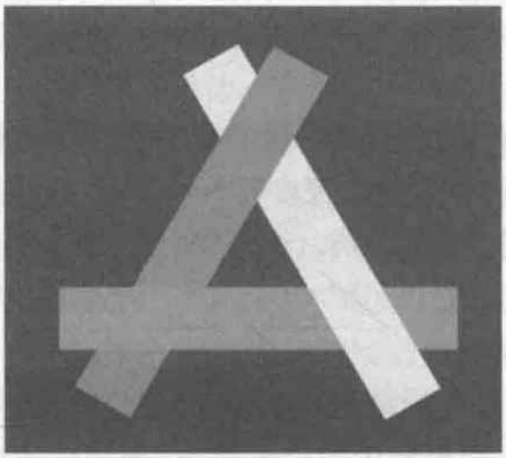


▲图 8.2　场景中有两个物体，其中 A 和 B 都是半透明物体

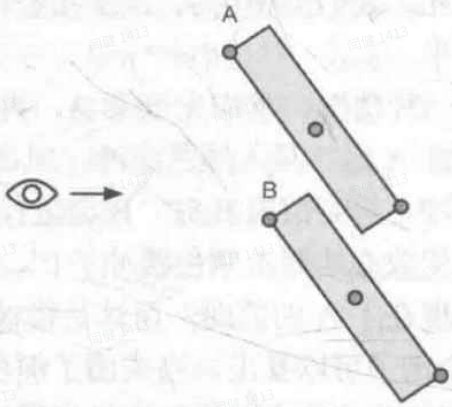
所以，从上述可以看出，只要有半透明物体参与透明度混合，它们之间的渲染顺序都是很重要的。渲染引擎一般都会先对物体进行排序，再渲染，常用手段是：

1. 先渲染所有的不透明物体，并开启他们的深度测试和深度写入。
2. 把半透明物体按他们距离摄像机的远近进行排序，然后按照从后往前的顺序渲染这些半透明物体，并开启它们的深度测试，但关闭深度写入。

但遗憾的是，因为zbuffer中的值是像素级别的，即每个像素有一个深度值，现在要对单个物体进行排序，则要么A在B前面渲染，要么A在B后面渲染。如果我们通过深度值来表示距离摄像机的远近，如果遇到了循环重叠的情况，则永远无法得到正确的结果。



▲图 8.3　循环重叠的半透明物体总是无法得到正确的半透明效果



▲图 8.4　使用哪个深度对物体进行排序。红色点分别标明了网格上距离摄像机最近的点、最远的点以及网格中点

或者如图8.4中，假如选择物体的某块网格来做深度排序，那么要选网格的中点？或者近点和远点？可以看到无论选择哪个点，结果都是错误的，排序结果总会是A在B的前面，但实际上A有一部分被B遮挡了。而这种问题的解决方法，通常也是分割网格。

Unity Shader的渲染顺序

为了解决渲染顺序的问题，Unity提供了**渲染队列（RenderQueue）**这一解决方案。我们使用SubShader的Queue标签来决定我们的模型将归于哪个渲染队列，且索引号越小表示越早被渲染。

Unity提前定义的渲染队列		
名称	队列索引	描述
Background	1000	该渲染队列会在任何其他队列之前被渲染，通常用于渲染那些需要绘制在背景上的物体。
Geometry	2000	默认的渲染队列，大多数物体都使用这个队列，不透明物体也使用这个队列。
AlphaTest	2450	需要透明度测试的物体使用这个队列。在所有不透明物体渲染之后再渲染它们会更加高效。
Transparent	3000	这个队列中的物体会在所有Geometry和AlphaTest物体渲染后，再按照**从后往前**的顺序进行渲染。任何使用了透明度混合（例如关闭了深度写入的Shader）的物体都应该使用该队列。
Overlay	4000	该队列用于实现一些叠加效果。任何需要在最后渲染的物体都应该使用该队列。

因此，如果我们想要通过透明度测试实现透明效果，代码中应该包含类似以下的代码：

```
SubShader{
    Tags { "Queue"="AlphaTest" }
    Pass {
        ...
    }
}
```

又或者想要通过透明度混合来实现透明效果，代码中应该包含类似下面的代码：

```
SubShader{
    Tags { "Queue"="Transparent" }
    Pass {
        ZWrite Off
        ...
    }
}
```

其中，**ZWrite Off**用于关闭深度写入，在这里我们选择把它写在Pass中。我们也可以把它写在SubShader中，这意味着该SubShader下的所有Pass都会关闭深度写入。

透明度测试

只要一个片元的透明度不满足条件（通常是小于某个阈值），那么它对应的片元就会被舍弃。被舍弃的片元将不会再进行任何处理，也不会对颜色缓冲产生任何影响；否则就会按照普通的不透明物体的方式来处理它。

通常，我们会在片元着色器中使用clip函数来进行透明度测试。clip是CG中的一个函数，它的定义如下：

```
//如果给定参数的任何一个分量是负数，就会舍弃当前像素的输出颜色。
void clip(float4 x)
//void clip(float3 x)
//void clip(float2 x)
//void clip(float1 x)
//void clip(float x)

//它等同于
void clip(float4 x)
{
    if (any(x < 0))
        discard;
}
```

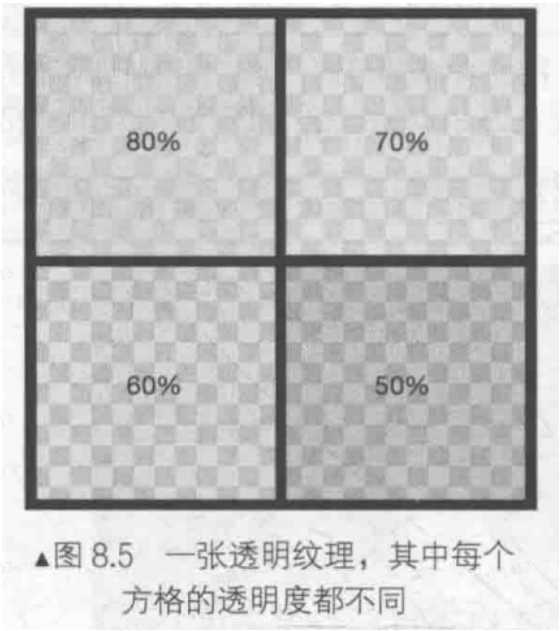
下面是代码实例：

```
Shader "Unity Shaders Book/C_8/C_8 Alpha Test"
{
    Properties
    {
        _Color("Main Tint", Color) = (1, 1, 1, 1)
        _MainTex("Main Tex", 2D) = "white"{}
        //为了在材质面板中控制透明度测试时使用的阈值，范围在[0,1]之间
        _Cutoff("Alpha Cutoff", Range(0, 1)) = 0.5
    }

    SubShader
    {
        //通常，使用了透明度测试的Shader都应该在SubShader中设置这三个标签。
        Tags
        {
            //透明度测试使用的渲染队列时AlphaTest的队列，因此需要设置Queue的标签为AlphaTest;
            "Queue"="AlphaTest"
            //RenderType标签可以把这个Shader归入到提前定义的组，这里是TransparentCutout组中，以指明该Shader是一个使用了透明度测试的Shader。
            //RenderType标签常用于着色器替换功能。
            "IgnoreProjector"="True"
            //此外，IgnoreProjector设置为True，意味着这个Shader不会受到投影器（Projectors）的影响。
            "RenderType"="TransparentCutout"
        }
        Pass
        {
            Tags
            {
                "LightMode"="ForwardBase"
            }

//              Cull Off

            CGPROGRAM

            #pragma vertex vert
            #pragma fragment frag

            #include "Lighting.cginc"

            fixed4 _Color;
```

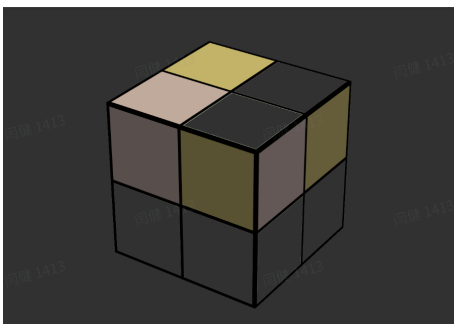
```
41              sampler2D _MainTex;
42              float4 _MainTex_ST;
43              //因为_Cutoff的范围是[0,1]，我们可以使用fixed精度来存储它。
44              fixed _Cutoff;
45
46              struct a2v
47              {
48                  float4 vertex : POSITION;
49                  float3 normal : NORMAL;
50                  float4 texcoord : TEXCOORD0;
51              };
52
53              struct v2f
54              {
55                  float4 pos : SV_POSITION;
56                  float3 worldNormal : TEXCOORD0;
57                  float3 worldPos : TEXCOORD1;
58                  float2 uv : TEXCOORD2;
59              };
60
61              v2f vert(a2v input)
62              {
63                  v2f output;
64                  output.pos = UnityObjectToClipPos(input.vertex);
65                  output.worldNormal = UnityObjectToWorldNormal(input.normal);
66                  output.worldPos = mul(unity_ObjectToWorld, input.vertex).xyz;
67                  output.uv = TRANSFORM_TEX(input.texcoord, _MainTex);
68                  return output;
69              }
70
71              fixed4 frag(v2f i) : SV_Target
72              {
73                  fixed3 worldNormal = normalize(i.worldNormal);
74                  fixed3 worldLightDir =
    normalize(UnityWorldSpaceLightDir(i.worldPos));
75                  fixed4 texColor = tex2D(_MainTex, i.uv);
76
77                  //Alpha Test
78                  clip(texColor.a - _Cutoff);
79                  //等同于
80                  // if ((texColor.a - _Cutoff) < 0.0)
81                  // {
82                  //     discard;
83                  // }
84
85                  fixed3 albedo = texColor.rgb * _Color.rgb;
86                  fixed3 ambient = UNITY_LIGHTMODEL_AMBIENT.xyz * albedo;
```

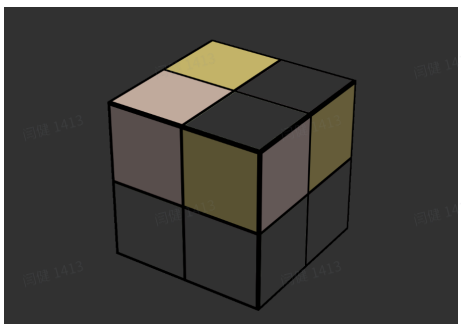
```
87                  //兰伯特光照计算漫反射
88                  fixed3 diffuse = _LightColor0.rgb * albedo * max(0,
    dot(worldNormal, worldLightDir));
89                  return fixed4(ambient + diffuse, 1.0);
90              }
91
92              ENDCG
93          }
94      }
95
96      // 使用内置的VertexLit Shader兜底
97      Fallback "Transparent/Cutout/VertexLit"
98  }
```

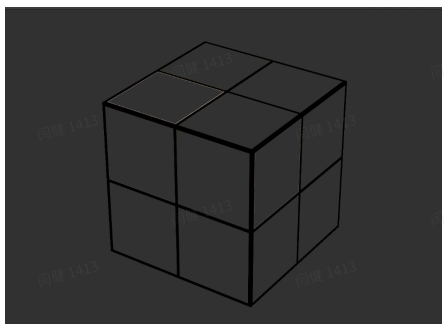
我们通过_Cutoff来控制透明度测试的阈值，小于该阈值的片元将被剔除为不可见片元；同时我们使用的素材，根据不同的区域划分为不同的透明度，根据_Cutoff值不同，透明区块的分布也会发生如下变化：



▲图 8.5　一张透明纹理，其中每个方格的透明度都不同







透明度混合

通过透明度混合可以得到真正的半透明效果，它会是用当前片元的透明度作为混合因子，与已经存储在颜色缓冲中的颜色值进行混合，得到新的颜色。同时，透明度混合需要关闭深度写入，这使得我们

需要非常小心物体的渲染顺序。

为了进行混合，我们需要使用Unity提供的混合命令——Blend。Blend是Unity提供的设置混合模式的命令。想要实现半透明效果就需要把当前自身的颜色和已经存在于颜色缓冲中的颜色值进行混合，混合时使用的函数就是由该指令决定的。

ShaderLab的Blend命令	
语义	描述
Blend Off	关闭混合
Blend SrcFactor DstFactor	开启混合并设置混合因子。源颜色（该片元产生的颜色）会乘以SrcFactor，而目标颜色（已经存在于颜色缓存的颜色）会乘以DstFactor，然后把两者相加后再存入颜色缓冲中。
Blend Blend SrcFactor DstFactor、SrcFactorA DstFactorA	和上面几乎一样，只是使用不同的因子来混合透明通道
BlendOp BlendOperation	使用BlendOperation对源颜色和目标颜色进行其他操作。

在使用Blend SrcFactor DstFactor进行混合时，需要注意，这个命令在设置混合因子的同时也开启了混合模式。这是因为只有开启了混合之后，设置片元的透明通道才有意义，而Unity在我们使用Blend命令的时候就自动帮我们打开了。只有设置了混合因子并打开混合模式后，才能得到正确的透明效果。当我们把源颜色的混合因子SrcFactor设置为SrcAlpha，而目标颜色的混合因子DstFactor设置为OneMinusScrAlpha `（1 - ScrAlpha）`，则混合后的颜色是：

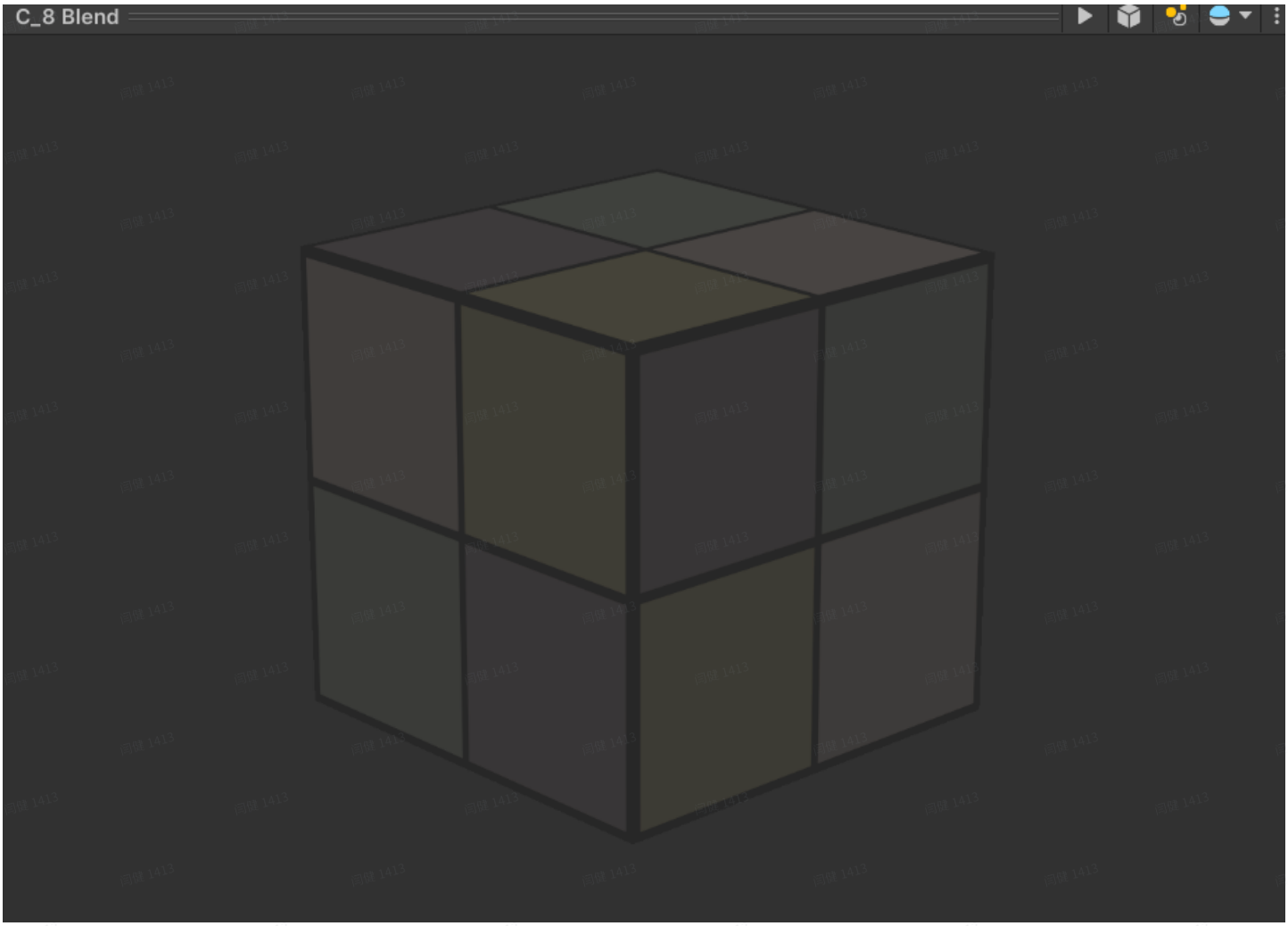
$$DstColor_{new}=SrcAlpha\times SrcColor+(1-SrcAlpha)\times DstColor_{old}$$

通常，透明度混合使用的就是这样的混合命令，实例的Shader代码和透明度测试的代码很相近：

```
Shader "Unity Shaders Book/C_8/C_8 Blend"
{
    Properties
    {
        _Color("Main Tint", Color) = (1, 1, 1, 1)
        _MainTex("Main Tex", 2D) = "white"{}
        //_AlphaScale替代透明度测试中的_Cutoff，用于控制整体的透明度
        _AlphaScale("Alpha Scale", Range(0, 1)) = 1
    }

    SubShader
    {
```

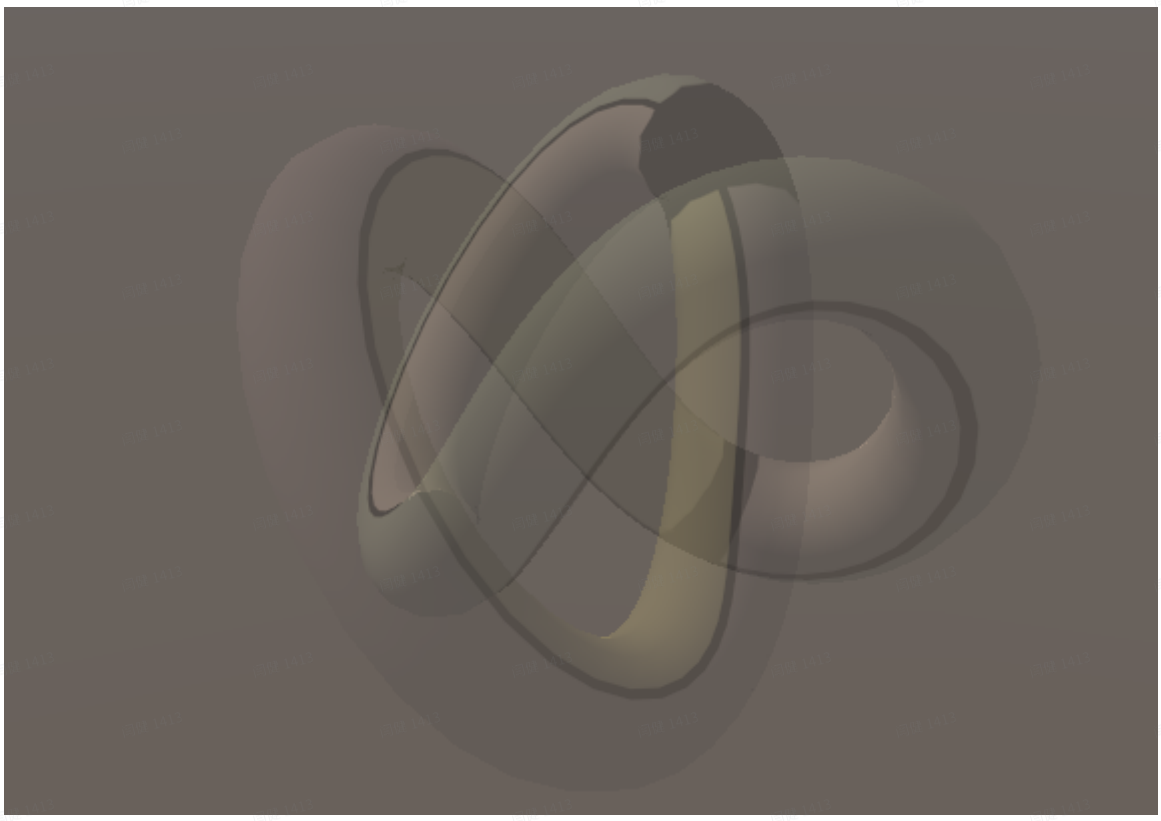
```
        Tags
        {
            //Blend应该放在Transparent队列中去渲染
            "Queue"="Transparent"
            //忽略投影带来的影响
            "IgnoreProjector"="True"
            //同时指明其类型为使用了透明度混合的Shader
            "RenderType"="Transparent"
        }

        Pass
        {
            Tags
            {
                "LightMode"="ForwardBase"
            }

            ZWrite Off
            Blend SrcAlpha OneMinusSrcAlpha

            CGPROGRAM

            #include "Lighting.cginc"

            fixed4 _Color;
            sampler2D _MainTex;
            float4 _MainTex_ST;
            fixed _AlphaScale;

            #pragma vertex vert;
            #pragma fragment frag;

            struct a2v
            {
                float4 vertex : POSITION;
                float3 normal : NORMAL;
                float4 texcoord : TEXCOORD0;
            };

            struct v2f
            {
                float4 pos : SV_POSITION;
                float3 worldNormal : TEXCOORD0;
                float3 worldPos : TEXCOORD1;
                float2 uv : TEXCOORD2;
            };

```

```
60              v2f vert(a2v input)
61              {
62                  v2f output;
63                  output.pos = UnityObjectToClipPos(input.vertex);
64                  output.worldNormal = UnityObjectToWorldNormal(input.normal);
65                  output.worldPos = mul(unity_ObjectToWorld, input.vertex).xyz;
66                  output.uv = TRANSFORM_TEX(input.texcoord, _MainTex);
67                  return output;
68              }
69
70              fixed4 frag(v2f i) : SV_Target
71              {
72                  fixed3 worldNormal = normalize(i.worldNormal);
73                  fixed3 worldLightDir =
    normalize(UnityWorldSpaceLightDir(i.worldPos));
74                  fixed4 texColor = tex2D(_MainTex, i.uv);
75
76                  fixed3 albedo = texColor.rgb * _Color.rgb;
77                  fixed3 ambient = UNITY_LIGHTMODEL_AMBIENT.xyz * albedo;
78                  //兰伯特光照计算漫反射
79                  fixed3 diffuse = _LightColor0.rgb * albedo * max(0,
    dot(worldNormal, worldLightDir));
80                  //对a值进行调整
81                  return fixed4(ambient + diffuse, texColor.a * _AlphaScale);
82              }
83
84              ENDCG
85          }
86      }
87      Fallback "Transparent/VertexLit"
88  }
```



_AlphaScale为0.2时的效果

同时，观察另一个模型可以看到，在循环交叠的情况下，这时渲染效果发生了异常：



这时，可以尝试利用深度写入，让模型像半透明物体一样进行淡入淡出。

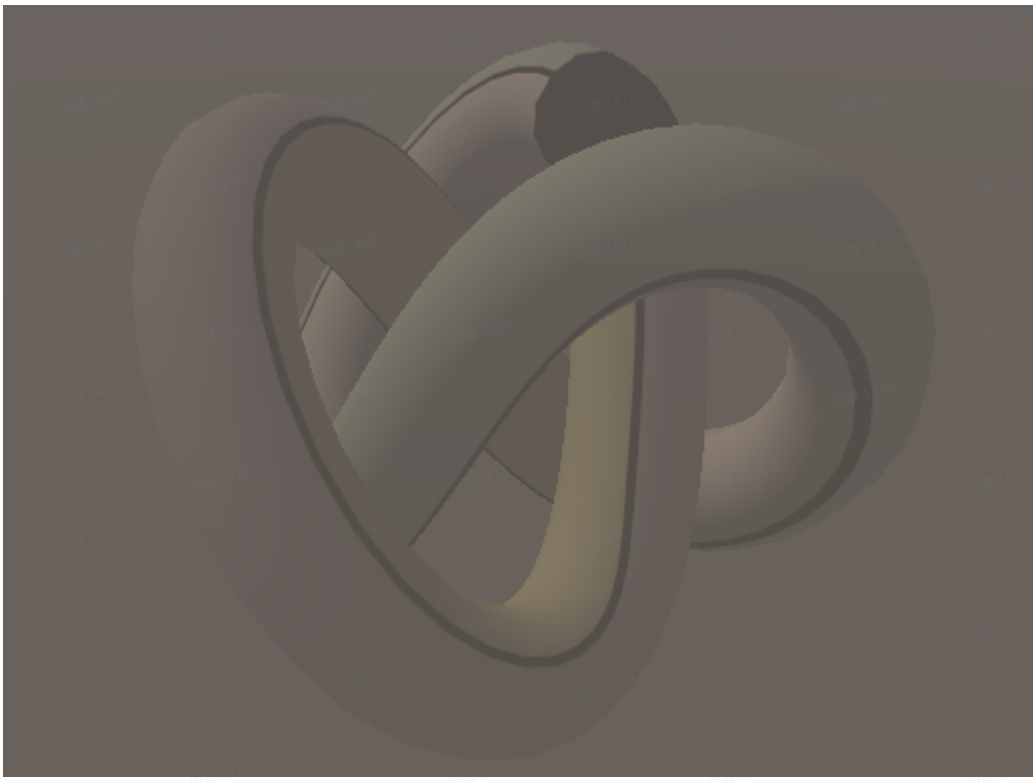
开启深度写入的半透明效果

为了解决上面的因为关闭深度写入而造成的错误排序，一种解决方法是使用两个Pass来渲染模型：第一个Pass开启深度写入，但不输出颜色，它的目的仅仅是为了把该模型的深度值写入深度缓冲中；第二个Pass进行正常的透明度混合，由于上一个Pass已经得到了逐像素的正确的深度信息，该Pass就可以按照像素级别的深度排序结果进行透明渲染。然而，多使用一个Pass会对性能造成一定的影响。

事实上，它相比上一节的Shader只多出了一个Pass：

```
1
2  //用于计算深度的Pass
3  Pass
4  {
5      //新Pass的目的仅仅是为了把模型的深度信息写入深度缓冲中，从而剔除模型中被自身遮挡的片元。
6      //因此,Pass的第一行开启了深度写入。
7      //在第二行，ColorMask用于设置颜色通道的写掩码。当ColorMask设置为0时，意味着该Pass不写入任何颜色通道，也不输出任何颜色
8      //同理ColorMask还可以设置RGB|A|...等其他任何RGBA的组合
9
10     ZWrite On
11     ColorMask 0
12 }
```

最终结果如图：



ShaderLab的混合命令

除了透明度混合之外，Blend还可以选择与颜色缓存中的颜色进行混合，其与两个操作数有关：**源颜色（source color）**和**目标颜色（destination color）**。由片元着色器产生的源颜色**S**，与从颜色缓冲中读取到的目标颜色**D**。对他们进行混合后得到的输出颜色**O**，再重新写入颜色缓冲中。这些颜色，他们都包含了RGBA是个通道的值，而非仅仅是RGB通道。

混合等式和参数

混合是一个逐片元的操作，而且它不是可编程的，但确是高度可配置的。如我们已知的**源颜色S**、**目标颜色D**和**输出颜色O**使用一个等式来计算，我们把这个等式称为**混合等式**。当进行混合时，需要使用两个混合等式：一个用于混合RGB通道，一个用于混合A通道。设置混合状态时，实际上就是在设置混合等式中的操作和因子。

ShaderLab的Blend命令	
语义	描述
Blend SrcFactor DstFactor	开启混合并设置混合因子。源颜色（该片元产生的颜色）会乘以SrcFactor，而目标颜色（已经存在于颜色缓存的颜色）会乘以DstFactor，然后把两者相加后再存入颜色缓冲中。
Blend Blend SrcFactor DstFactor、SrcFactorA DstFactorA	和上面几乎一样，只是使用不同的因子来混合透明通道

之前我们已经见过这几个混合命令了，第一个命令只提供了两个因子，这意味着将使用同样的混合因子来混合RGB通道和A通道，将这些因子进行加法混合时是用的混合公式为：

$$O_{rgb} = SrcFactor \times S_{rgb} + DstFactor \times D_{rgb}$$
$$O_{a} = SrcFactorA \times S_{a} + DstFactorA \times D_{a}$$

此外，这些混合因子的值可以为：

ShaderLab中的混合因子	
参数	描述
One	因子为1
Zero	因子为0
SrcColor	

	因子为源颜色值。当用于混合RGB的混合等式时，使用SrcColor的RGB分量作为混合因子；当用于混合A的混合等式时，使用SrcColor的A分量作为混合因子。
SrcAlpha	因子为源颜色的透明度值（A通道）
DstColor	因子为目标颜色。当用于混合RGB的混合等式时，使用DstColor的RGB分量作为混合因子；当用于混合A的混合等式时，使用DstColor的A分量作为混合因子。
DstAlpha	因子为目标颜色的透明度值（A通道）
OneMinusSrcColor	因子为(1-源颜色)。当混合RGB时，使用RGB分量作为混合因子；当混合A时，使用A分量作为混合因子。
OneMinusSrcAlpha	因子为(1-源颜色的透明度值)。
OneMinusDstColor	因子为(1-目标颜色)。当混合RGB时，使用RGB分量作为混合因子；当混合A时，使用A分量作为混合因子。
OneMinusDstAlpha	因子为(1-源颜色的透明度值)。

使用上面的指令进行设置时，RGB通道和A通道的混合因子都是一样的。可以用**Blend SrcFactor DstFactor，SrcFactorA DstFactorA**指令使用不同的参数混合A通道。

```
1 //在混合后，用源颜色的透明度作为输出颜色的透明度
2 Blend SrcAlpha OneMinusSrcAlpha, One Zero
```

混合操作

混合操作命令指的是ShaderLab中的**BlendOp BlendOperation**命令：

表 8.5 ShaderLab 中的混合操作

操　作	描　述
Add	将混合后的源颜色和目的颜色相加。默认的混合操作。使用的混合等式是： $O_{rgb} = SrcFactor \times S_{rgb} + DstFactor \times D_{rgb}$ $O_a = SrcFactorA \times S_a + DstFactorA \times D_a$
Sub	用混合后的源颜色减去混合后的目的颜色。使用的混合等式是： $O_{rgb} = SrcFactor \times S_{rgb} - DstFactor \times D_{rgb}$ $O_a = SrcFactorA \times S_a - DstFactorA \times D_a$
RevSub	用混合后的目的颜色减去混合后的源颜色。使用的混合等式是： $O_{rgb} = DstFactor \times D_{rgb} - SrcFactor \times S_{rgb}$ $O_a = DstFactorA \times D_a - SrcFactorA \times S_a$
Min	使用源颜色和目的颜色中较小的值，是逐分量比较的。使用的混合等式是： $O_{rgba} = (\min(S_r, D_r), \min(S_g, D_g), \min(S_b, D_b), \min(S_a, D_a))$
Max	使用源颜色和目的颜色中较大的值，是逐分量比较的。使用的混合等式是： $O_{rgba} = (\max(S_r, D_r), \max(S_g, D_g), \max(S_b, D_b), \max(S_a, D_a))$
其他逻辑操作	仅在 DirectX 11.1 中支持

混合操作命令通常是与混合因子命令一起工作的。当使用Min或者Max混合操作时，混合因子实际上是不起作用的，它们仅会判断原始的源颜色和目的颜色之间的比较结果。

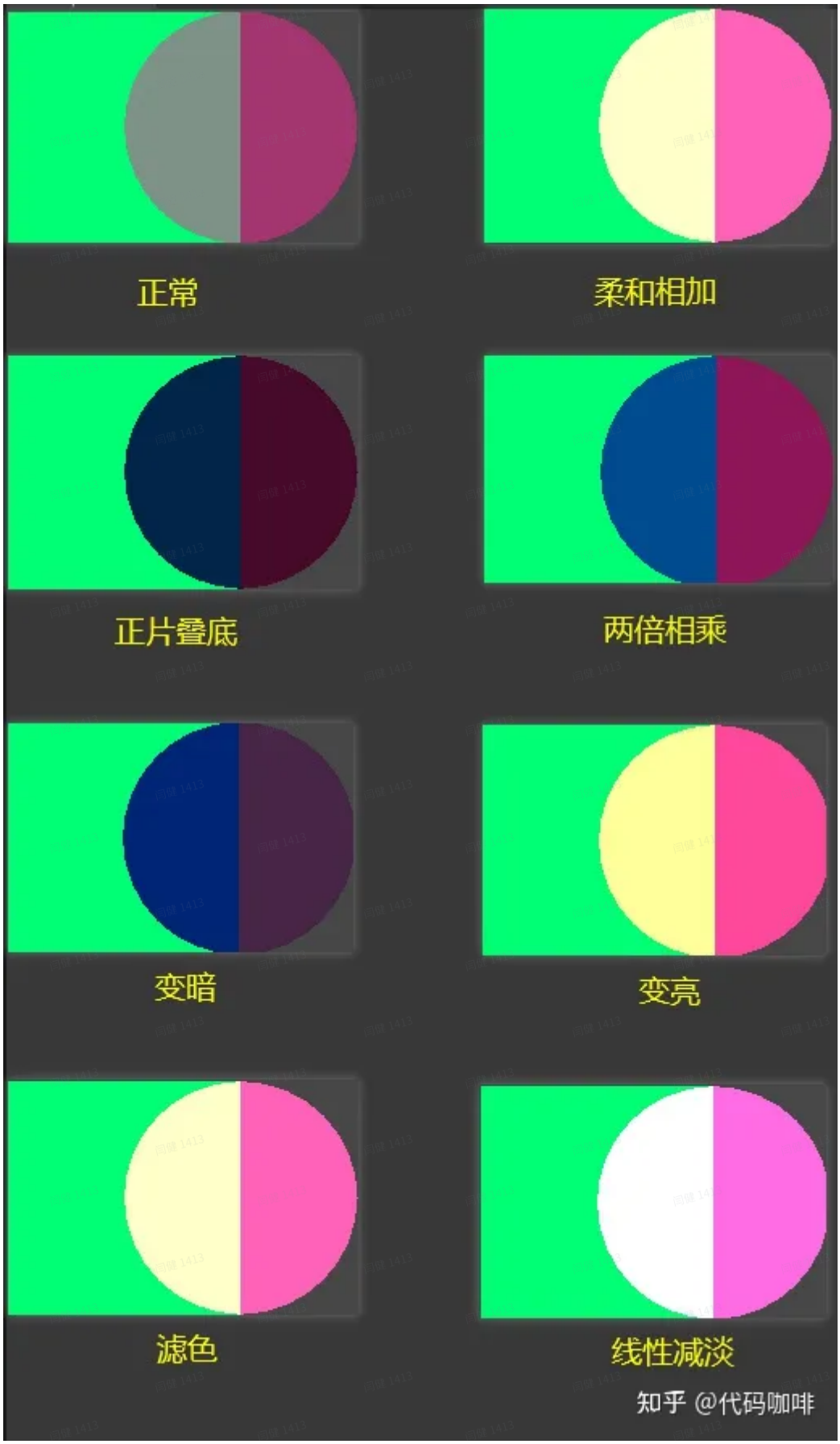
常见的混合类型

通过混合操作和混合因子命令的组合，我们可以得到一些类似Photoshop混合模式中的混合效果：

```
1  //正常(Normal)，即透明度混合
2  Blend SrcAlpha OneMinusSrcAlpha
3
4  //柔和相加(Soft Additive)
5  Blend OneMinusDstColor One
6
7  //正片叠底(Multiply)，即相乘
8  Blend DstColor Zero
9
10 //两倍相乘(2x Multiply)
11 Blend DstColor SrcColor
12
13 //变暗(Darken)
14 BlendOp Min
15 Blend One One
16
```

```
17 //变量(Lighten)
18 BlendOp Max
19 Blend One One
20
21 //滤色(Screen)
22 Blend OneMinusDstColor One
23 //等同于
24 Blend One OneMinusSrcColor
25
26 //线性减淡(Linnear Dodge)
27 Blend One One
```

对应的效果如图：



双面渲染的透明效果

在现实生活中的透明物体来说，其内部的结构也应该是可以观察到的，无论是透明度测试还是透明度混合，都应该能正常观察到正方体内部及其背面的形状。而默认情况下渲染引擎剔除了物体背面（相

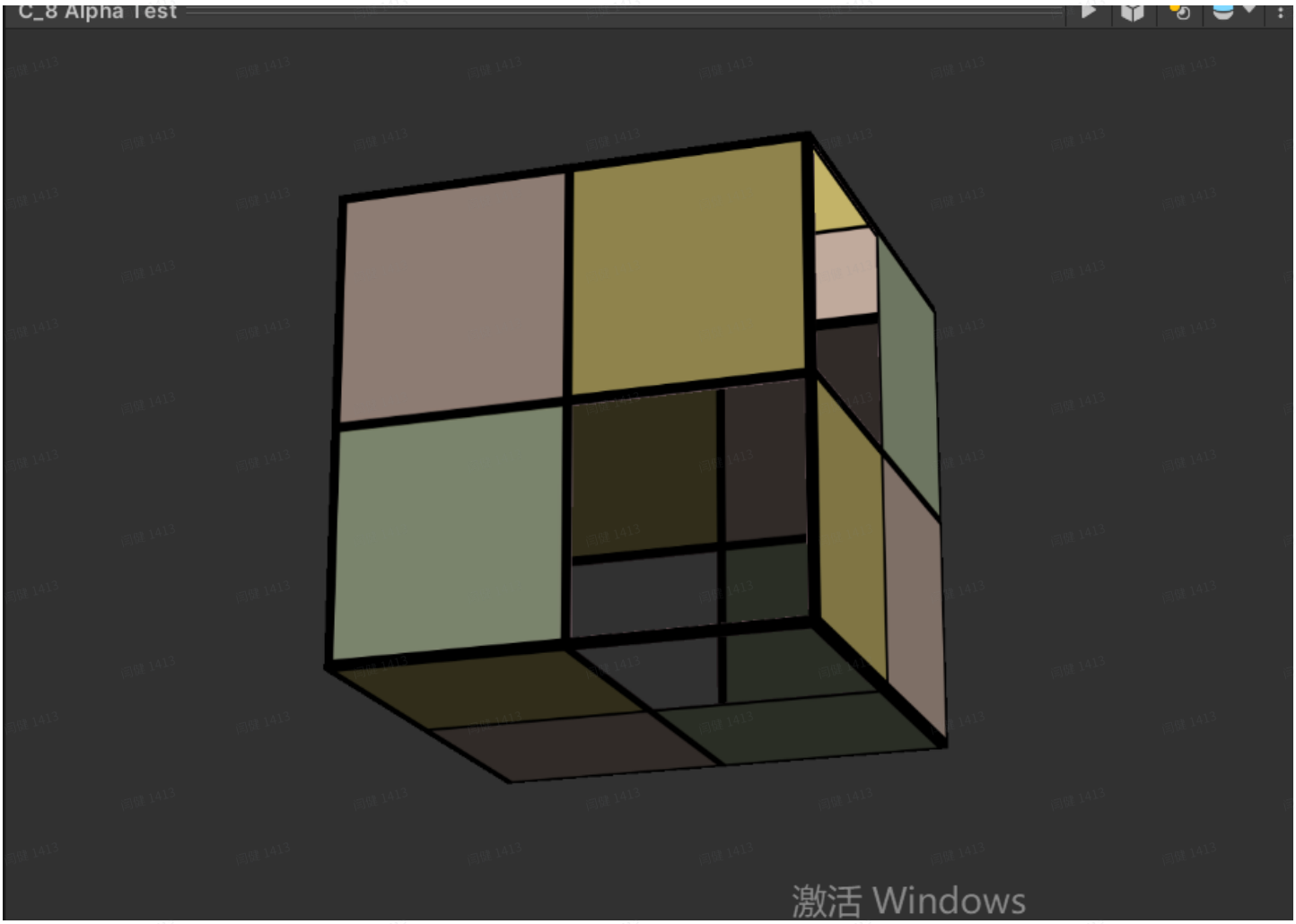
对于摄像机的方向）的渲染图元，只渲染了物体的正面。如果我们想要得到双面渲染的效果，可以使用Cull指令来控制需要剔除哪个面的渲染图元。在Unity中，Cull的语法如下：

```
Cull Back | Front | Off
```

1. Cull Back，则背对着摄像机的渲染图元就不会被渲染，这也是默认的剔除状态。
2. Cull Front，则朝向摄像机的渲染图元就不会被渲染。
3. Cull Off，会关闭剔除功能，所有的渲染图元都会被渲染，由于此时需要渲染的图元数目会成倍增加，因此除非是特殊效果，否则不会关闭剔除的。

透明度测试的双面渲染

我们需要在Pass的渲染设置中使用Cull指令来关闭剔除，即可实现双面渲染的效果。可以在之前的AlphaTest的代码中加上Cull指令就能看到效果，在Pass块下添加Cull Off代码。



透明度混合的双面渲染

透明度混的的双面渲染会麻烦很多，因为需要关闭深度写入，同时渲染顺序是很重要的。在这里，可以把双面渲染的工作分成两个Pass，一个只渲染背面，一个只渲染正面。当Unity顺序执行SubShader

中的各个Pass时，我们就可以保证背面总是在正面被渲染之前渲染，从而保证正确的深度渲染关系。

```
// 修改Shader命名
Shader "Unity Shaders Book/C_8/C_8 Blend BothSide"
{
    Properties
    {
        _Color ("Main Tint", Color) = (1, 1, 1, 1)
        _MainTex ("Main Tex", 2D) = "white" {}
        _AlphaScale ("Alpha Scale", Range(0, 1)) = 1
    }
    SubShader
    {
        Tags {
            "Queue" = "Transparent"
            "IgnoreProjector" = "True"
            "RenderType" = "Transparent"
        }

        // 新增一个Pass处理背面
        Pass
        {
            Tags { "LightMode" = "ForwardBase" }

            // 剔除正面，做背面图元的渲染
            Cull Front

            // 下面是和之前完全一样的代码，直接复制一份就好
            ZWrite Off
            Blend SrcAlpha OneMinusSrcAlpha

            CGPROGRAM

            #pragma vertex vert
            #pragma fragment frag

            #include "Lighting.cginc"

            fixed4 _Color;
            sampler2D _MainTex;
            float4 _MainTex_ST;
            fixed _AlphaScale;

            struct a2v
            {
                float4 vertex : POSITION;
```

```
45              float3 normal : NORMAL;
46              float4 texcoord : TEXCOORD0;
47          };
48
49          struct v2f
50          {
51              float4 pos : SV_POSITION;
52              float3 worldNormal : TEXCOORD0;
53              float3 worldPos : TEXCOORD1;
54              float2 uv : TEXCOORD2;
55          };
56
57          v2f vert(a2v v)
58          {
59              v2f o;
60              o.pos = UnityObjectToClipPos(v.vertex);
61              o.worldNormal = UnityObjectToWorldNormal(v.normal);
62              o.worldPos = mul(unity_ObjectToWorld, v.vertex).xyz;
63              o.uv = TRANSFORM_TEX(v.texcoord, _MainTex);
64              return o;
65          }
66
67          fixed4 frag(v2f i) : SV_Target
68          {
69              fixed3 worldNormal = normalize(i.worldNormal);
70              fixed3 worldLightDir =
    normalize(UnityWorldSpaceLightDir(i.worldPos));
71
72              fixed4 texColor = tex2D(_MainTex, i.uv);
73
74              fixed3 albedo = texColor.rgb * _Color.rgb;
75
76              fixed3 ambient = UNITY_LIGHTMODEL_AMBIENT.xyz * albedo;
77
78              fixed3 diffuse = _LightColor0.rgb * albedo
79                      * max(0, dot(worldNormal, worldLightDir));
80
81              return fixed4(ambient + diffuse, texColor.a * _AlphaScale);
82          }
83          ENDCG
84      }
85
86      Pass
87      {
88          Tags { "LightMode" = "ForwardBase" }
89
90          // 剔除背面，做正面图元的渲染
```

```
            Cull Back

            ZWrite Off
            Blend SrcAlpha OneMinusSrcAlpha

            CGPROGRAM

            #pragma vertex vert
            #pragma fragment frag

            #include "Lighting.cginc"

            fixed4 _Color;
            sampler2D _MainTex;
            float4 _MainTex_ST;
            fixed _AlphaScale;

            struct a2v
            {
                float4 vertex : POSITION;
                float3 normal : NORMAL;
                float4 texcoord : TEXCOORD0;
            };

            struct v2f
            {
                float4 pos : SV_POSITION;
                float3 worldNormal : TEXCOORD0;
                float3 worldPos : TEXCOORD1;
                float2 uv : TEXCOORD2;
            };

            v2f vert (a2v v)
            {
                v2f o;
                o.pos = UnityObjectToClipPos(v.vertex);
                o.worldNormal = UnityObjectToWorldNormal(v.normal);
                o.worldPos = mul(unity_ObjectToWorld, v.vertex).xyz;
                o.uv = TRANSFORM_TEX(v.texcoord, _MainTex);
                return o;
            }

            fixed4 frag(v2f i) : SV_Target
            {
                fixed3 worldNormal = normalize(i.worldNormal);
                fixed3 worldLightDir =
    normalize(UnityWorldSpaceLightDir(i.worldPos));
```

```
137
138                 fixed4 texColor = tex2D(_MainTex, i.uv);
139
140                 fixed3 albedo = texColor.rgb * _Color.rgb;
141
142                 fixed3 ambient = UNITY_LIGHTMODEL_AMBIENT.xyz * albedo;
143
144                 fixed3 diffuse = _LightColor0.rgb * albedo
145                         * max(0, dot(worldNormal, worldLightDir));
146
147                 return fixed4(ambient + diffuse, texColor.a * _AlphaScale);
148             }
149             ENDCG
150         }
151     }
152
153     Fallback "Transparent/VertexLit"
154 }
```

在第一个Pass中，先Cull Front，只渲染背面；

在第二个Pass中，再Cull Back，只渲染正面；