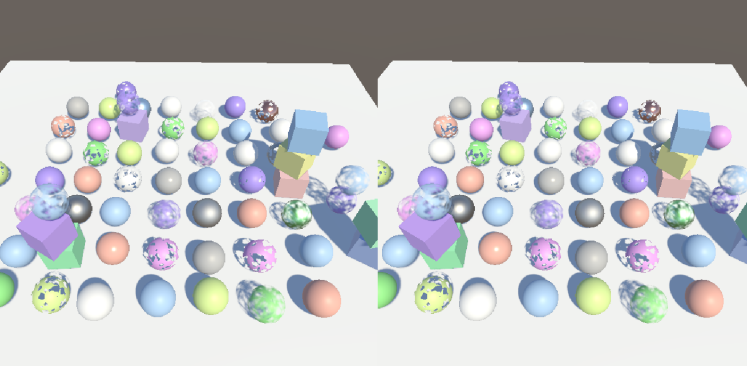
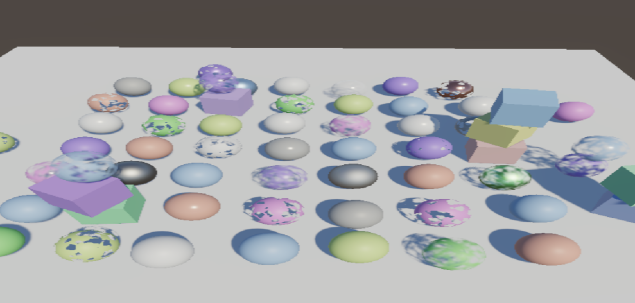
本部分支持多相机多后处理。

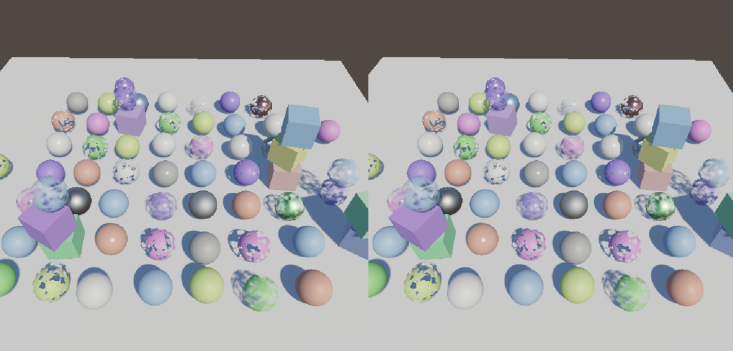
当前不开启后处理时，可以支持多视口（更改相机的viewport）：



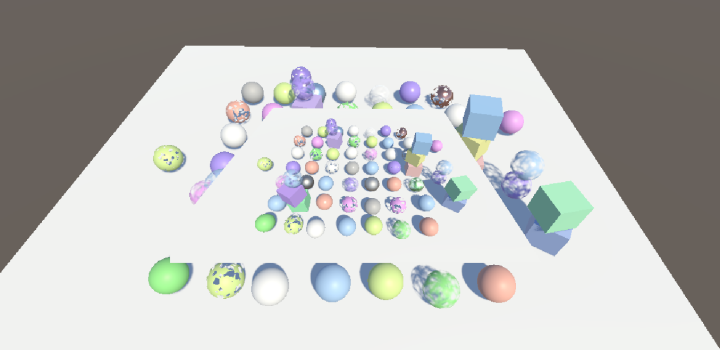
但是打开后处理后，结果出错：



原因是SetRenderTarget会重置viewport，因此对于最后一次绘制，使用定制的DrawFinal：



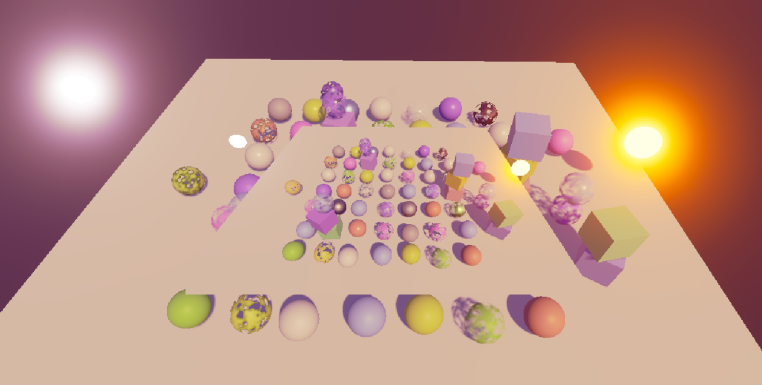
再调整相机变成重叠渲染，我们将前面的相机设为Clear Depth Only，可以显示去掉天空盒的效果：



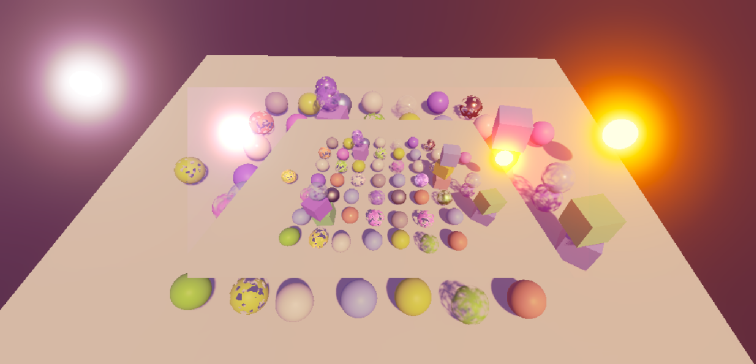
但是postFX还不支持，因为我们没有进行alpha通道的混合。

先试着实现Alpha Blending，注意Final Pass的RT要设置为Load。

为了保留原贴图的alpha通道，修改Bloom操作，将alpha存储为原贴图的alpha。



这里会发现Bloom效果对透明区域的贡献为0，因此我们把Color的Blend设置为One，同时把摄像机的背景颜色设置为纯黑：



我们使用One OneMinusSrcAlpha来混合Alpha值，在Lit和Unlit里声明。

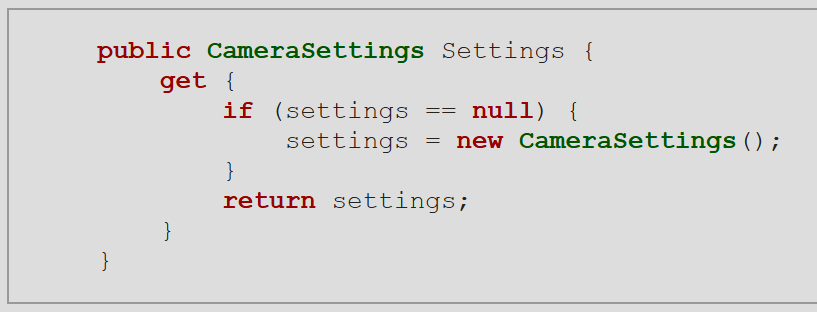
同时判断，如果当前写深度，则Alpha值一直为1。

把混合模式作为可配置选项，添加到一个单独的Component里。

知识点，语法糖：

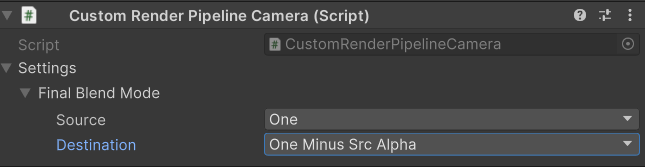


等价于：



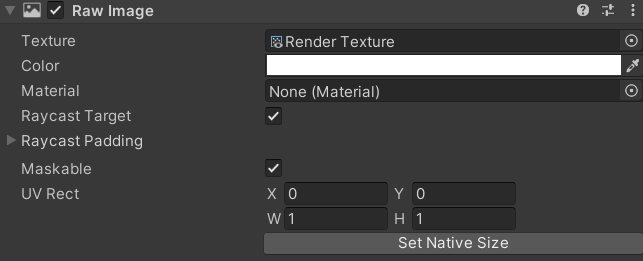
然后在CameraRenderer里获取该组件，并设置Blend模式。

默认是One Zero，所以我们可以自定义混合模式给OverlayCamera。

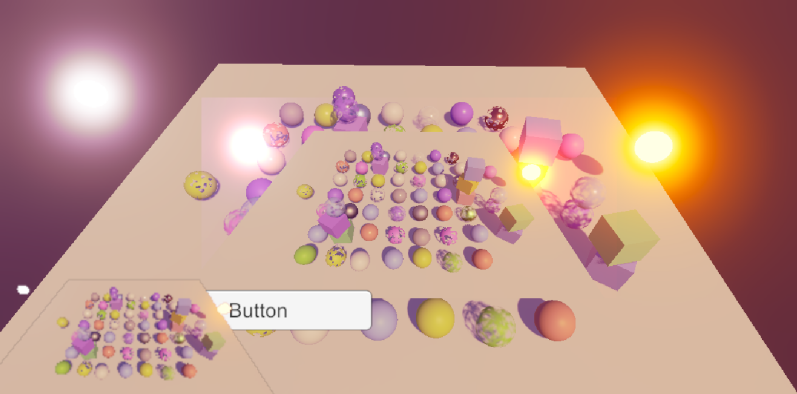


在一些情况下，我们需要让相机渲染到指定的目标纹理（Render Texture）上，再显示到GUI里，通过Assets / Create / Render Texture创建，不需要DepthBuffer，再绑定到相机的TargetTexture上。Unity的绘制逻辑是：首先，具有目标纹理的摄像机按深度增加的顺序渲染，然后是没有目标纹理的摄像机。

通过GameObject / UI / Raw Image创建对象并绑定纹理。



效果如下：

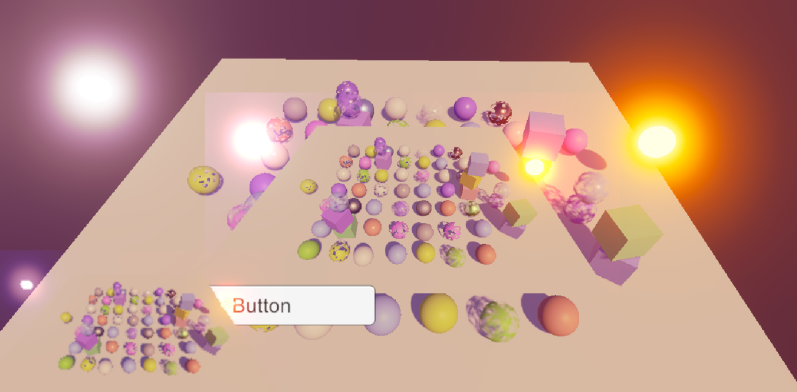


默认的UI材质使用Alpha Blending，所以Bloom失效，并且双线性插值导致透明过渡区会混合黑色背景，形成轮廓。

复制Default-UI着色器，增加Blend的属性。

前往https://unity.com/cn/releases/editor/archive，找到对应的Unity版本，然后从下拉菜单之一中进行选择 Built in shaders 。着色器位于 DefaultResourcesExtra / UI 文件夹中。

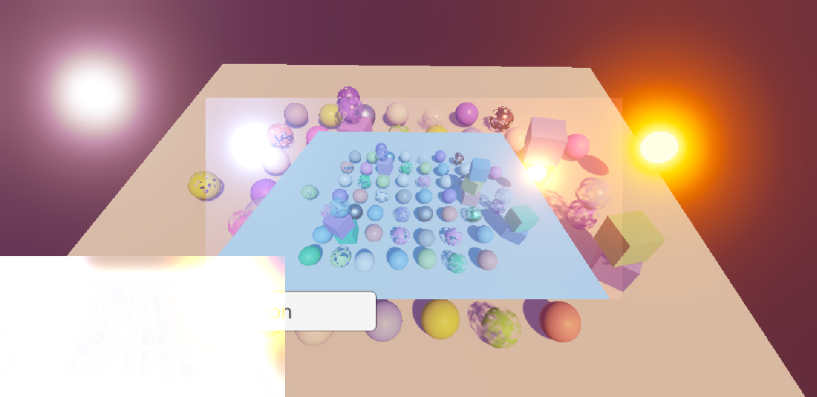
然后正确设置混合模式：



下面，在CameraSettings里添加可配置的后处理Setting，让相机决定是否覆盖全局的设置。



在CameraRenderer里判断是否覆盖，下面是每个相机不同后处理的效果：



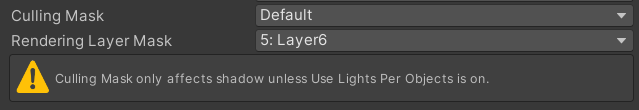
接下来应用相机的渲染层。

相机的Culling Mask可以决定哪些Layer的物体不被显示。

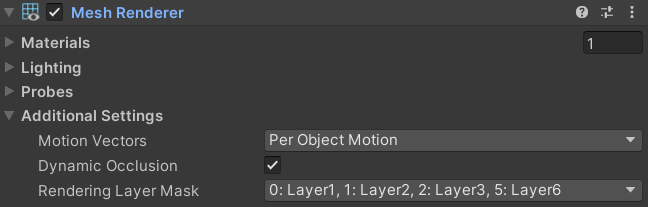
光源也有这样的Mask，但是只会影响阴影，因为绘制阴影贴图时，光源才被视为相机。通过EditorGUILayout.HelpBox，我们可以在前端显示该警告。



对于点光源和聚光灯：



使用SRP时，物体会有Rendering Layer Mask属性：



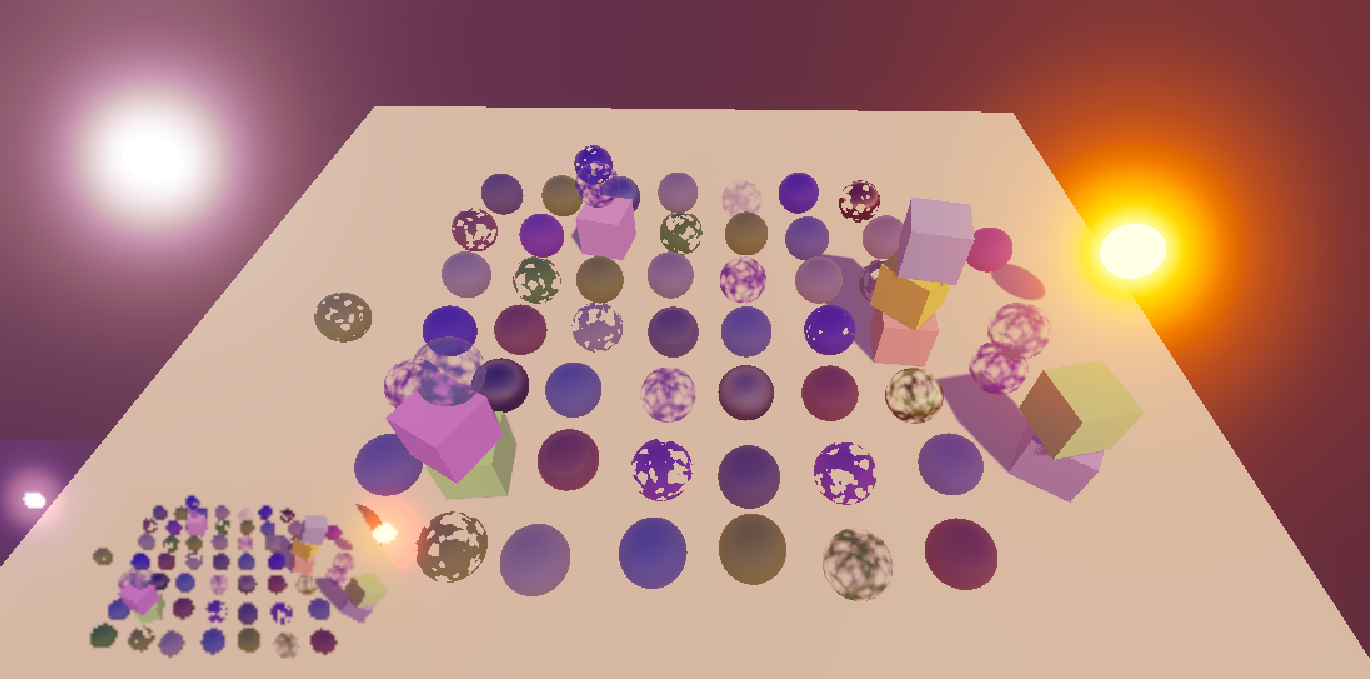
这些layer的名字可以通过重写[RenderPipelineAsset](http://docs.unity3d.com/Documentation/ScriptReference/Rendering.RenderPipelineAsset.html).renderingLayerMaskNames的getter实现。我们需要在Editor中实现它。对于光源，还需要在GUI部分进行额外处理，详细见代码。



在Shadow的DrawSettings里打开useRenderingLayerMaskTest以在阴影绘制中启用Mask。

同时，在shader中，物体自身的Mask通过unity\_RenderingLayer的第一个通道传输到GPU，并且通过asuint()将float按字节转化为uint。而光源的Mask则需要通过和Directions打包到一起上传。然后在计算光照时判断Mask是否有相同的决定光照的影响。

注意，在上传光源Mask时，我们需要重新把int解释为float再上传。在C#中，通过类似union的操作实现reinterpret，详见ReinterpretExtensions.cs。效果：

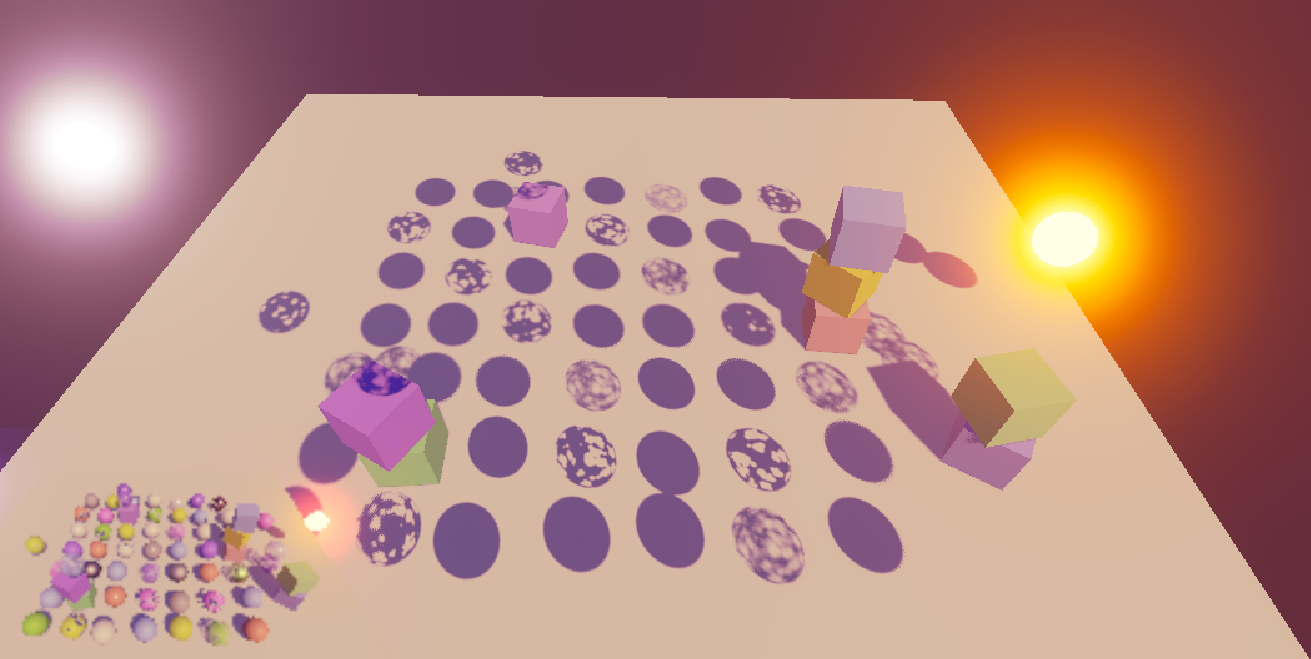


然后我们为相机也添加对应的设置，该设置的GUI是自定义的，为了只自定义CameraSettings里的一条属性的GUI，具体见RenderingLayerMaskDrawer的实现。



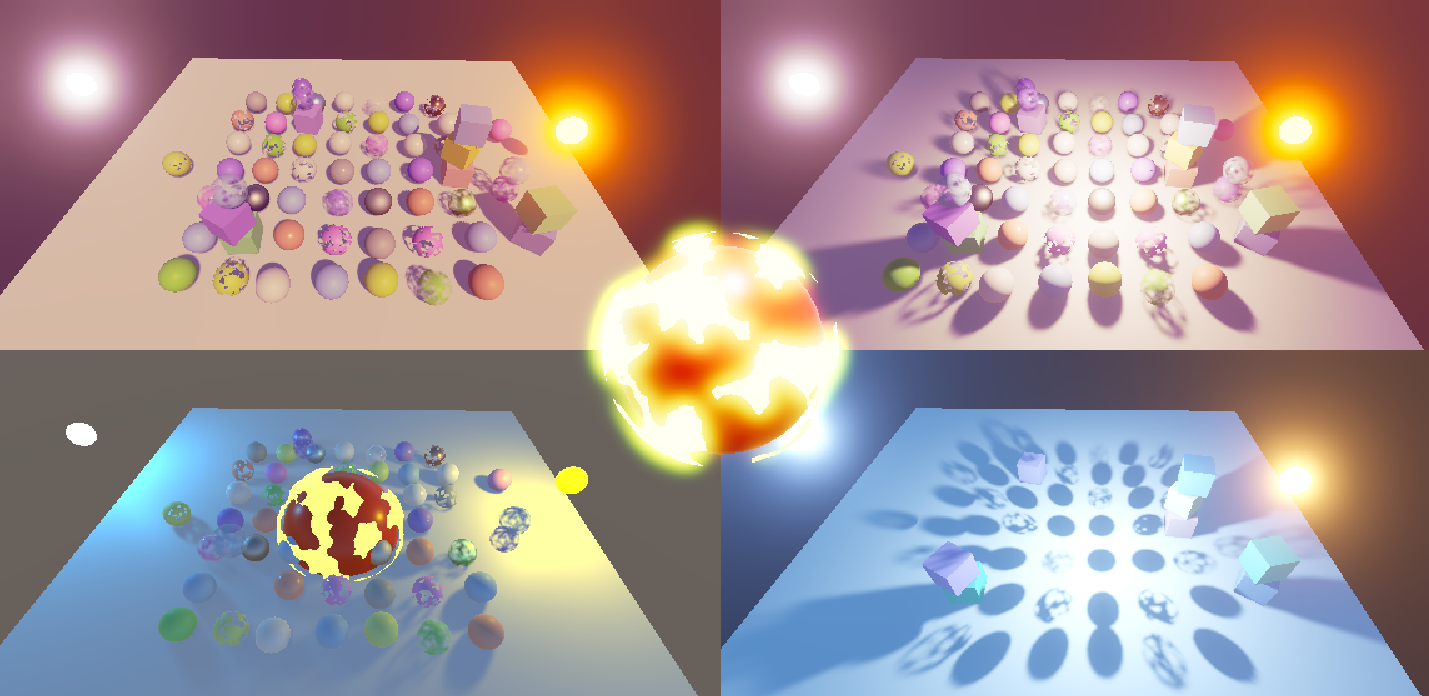
然后把该Mask传给Camera Renderer里的FilterSettings。

**使用自定义的Rendering Mask，可以不绘制某些物体，但是让它们能投射阴影：**



在此基础上，我们还可以把这个Mask应用在灯光上，在SetupLights中做判断。

最终结果：



比如右下角的图，我们把物体分为Layer3和Layer4，让点光源也处理Layer3和Layer4，但是相机只绘制Layer4的，所以结果中有Layer4的物体以及Layer3和4的阴影。