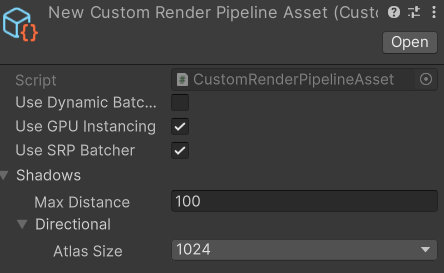
本部分包含方向光源的阴影实现，并支持了级联阴影方案，还使用了bias和PCF改善阴影。

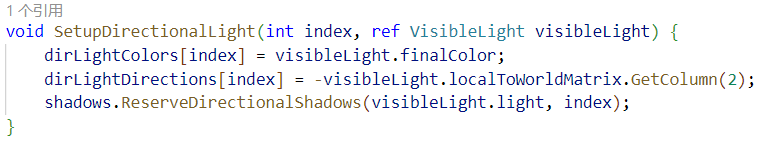
该部分内容较多，我在代码处也加了一些注释方便阅读。

使用RP Asset配置阴影参数，并在构造时传给RP。



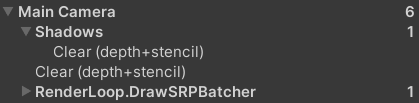
对应的参数进一步传给Renderer，先在剔除的时候使用maxShadowDistance，其次在管理光源的结构体中，创建管理阴影的结构体实例(Shadows)。

然后在设置平行光的时候，存储可以投射阴影的光源index。



接着，在设置好光源后，调用Shadows类的Render方法，在方法中，先申请阴影贴图，再将该帖图设置为RenderTarget。

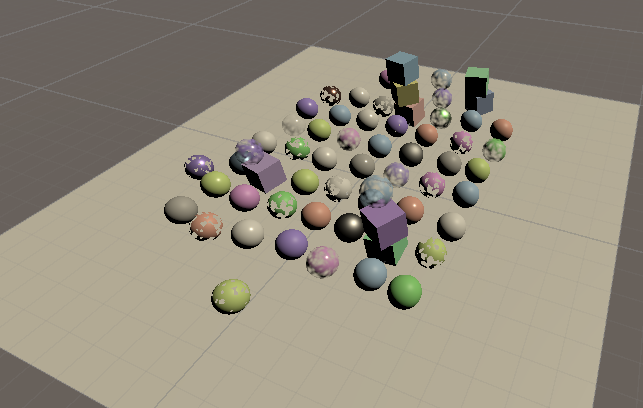
这时，我们需要调整Pipeline的顺序，把设置光照的部分与设置相机的部分对调，目的是在绘制正常的场景前，就完成阴影贴图的绘制工作。



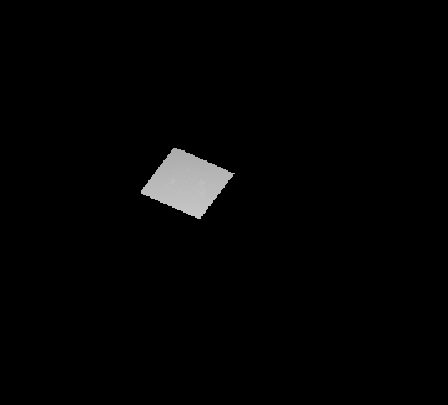
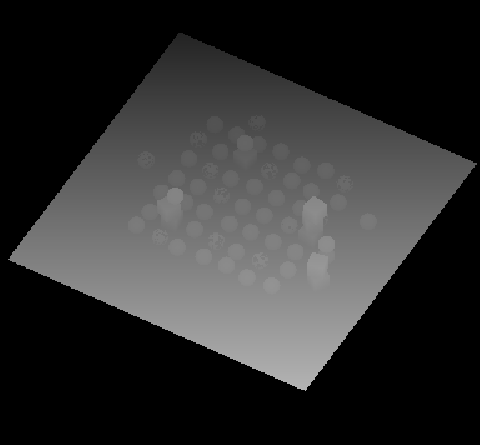
配置好shadowSettings（具体见代码），就可以通过Context调用DrawShadows了。

但是，该方法只会对包含ShadowCaster这个Pass的材质进行绘制，所以要为着色器添加对应Pass。并且，该Pass只会写入深度。对于Lit材质对应的ShadowCaster，只需要拷贝它着色部分的代码，并删除绘制颜色的部分即可。

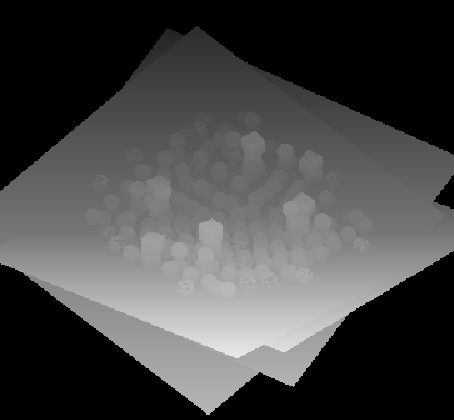
测试场景，一个方向光：



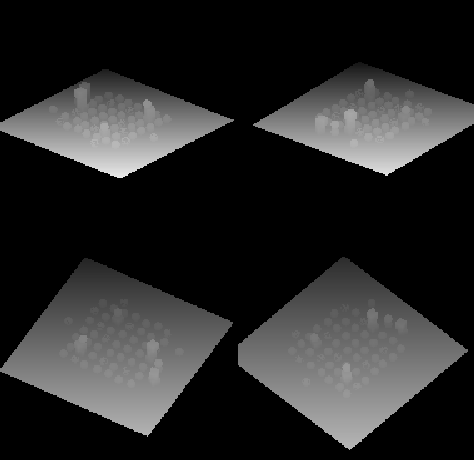
改变最大阴影距离，查看阴影贴图绘制结果：

但是当使用多个光源时，它们的阴影贴图会叠加到一起：



修改Viewport的设置，让它们不会重合：



为了能进行阴影深度的比较，将变换矩阵存储起来，并且需要对矩阵进行预处理，分别是：1.消除reverse\_z的影响; 2.将裁剪空间坐标变换为屏幕空间坐标（并且需要把深度范围也变到0-1）; 3.将viewport变换也考虑进来

以上是思路，下面是对于矩阵变换公式的具体解释：

<https://zhuanlan.zhihu.com/p/683661359>

我们需要让光源记录对应的Shadow数据，把光源的ShadowStrength和在Shadows里的index打包作为Vector数组上传。

对于前面创建的上传到GPU的数据和贴图，编写对应的hlsl文件来声明接口。

注意，阴影贴图的采样器设置为：sampler\_linear\_clamp\_compare。

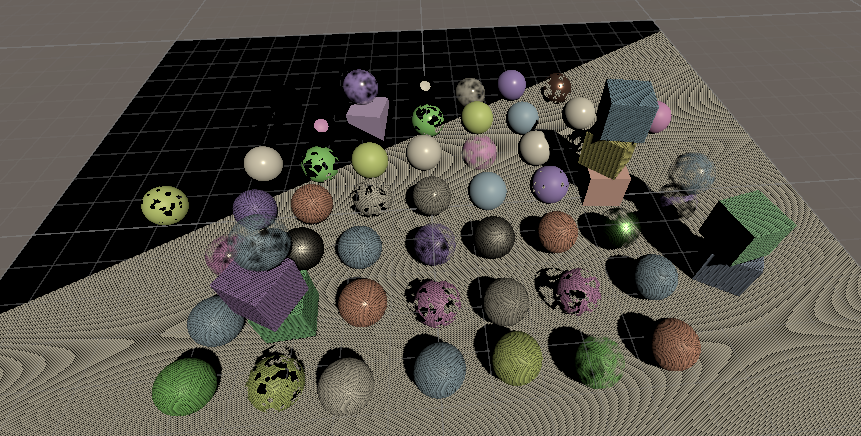
对阴影贴图采样时，通过SAMPLE\_TEXTURE2D\_SHADOW实现深度值的比较，得到结果。

在对阴影贴图进行采样时，结果会根据阴影强度进行衰减：

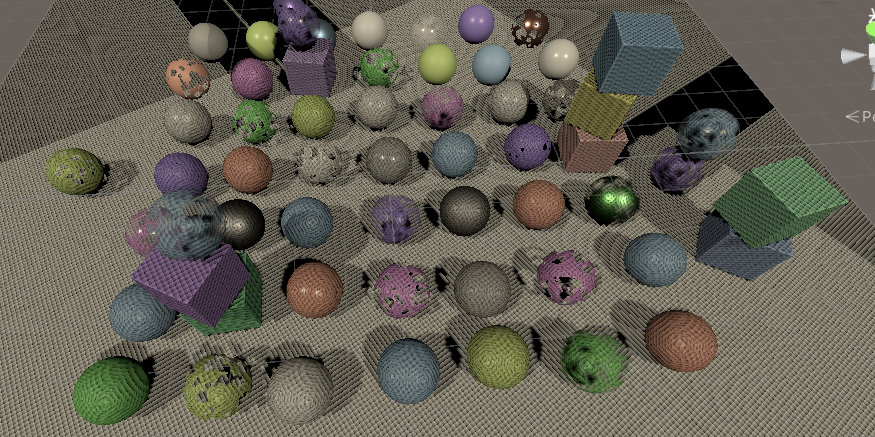


我们把这个结果记录在光源的结构体里（attenuation）。

会得到范围外有问题，且存在自遮挡的阴影结果，我们之后处理它们。

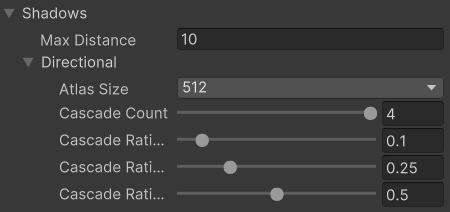


两个光源的结果：



对于阴影而言，理想情况是阴影贴图的像素大小等于显示贴图的像素大小。因此我们应该根据视线距离改变阴影贴图分辨率，这就是级联阴影的方法。

级联阴影设置也放在RP Asset里：



然后，调整阴影相关数组大小，并调整tile和split大小，在绘制阴影贴图时，传入Cascade相关参数生成对应的渲染设置。

单个光源：

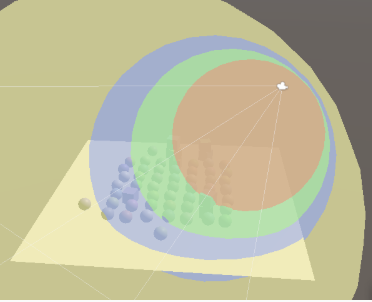


四个光源：



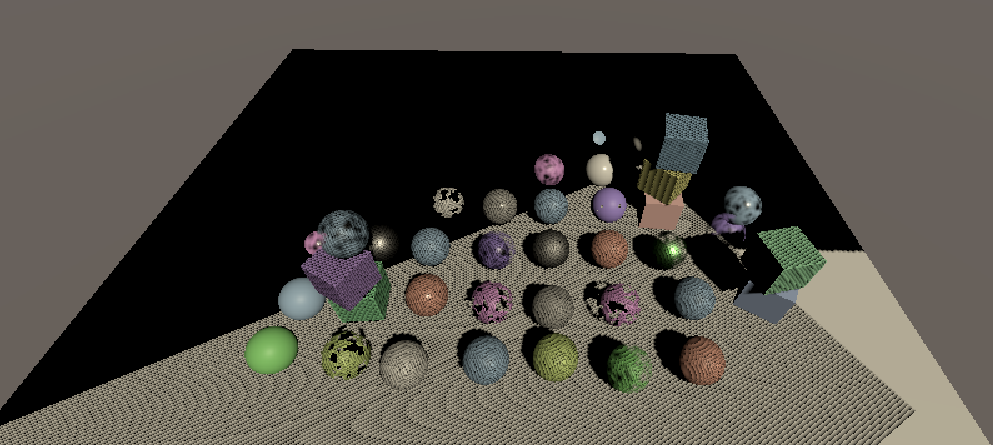
每一级裁剪都是一个裁剪球体，把它们上传到GPU，以确定选择的层级。

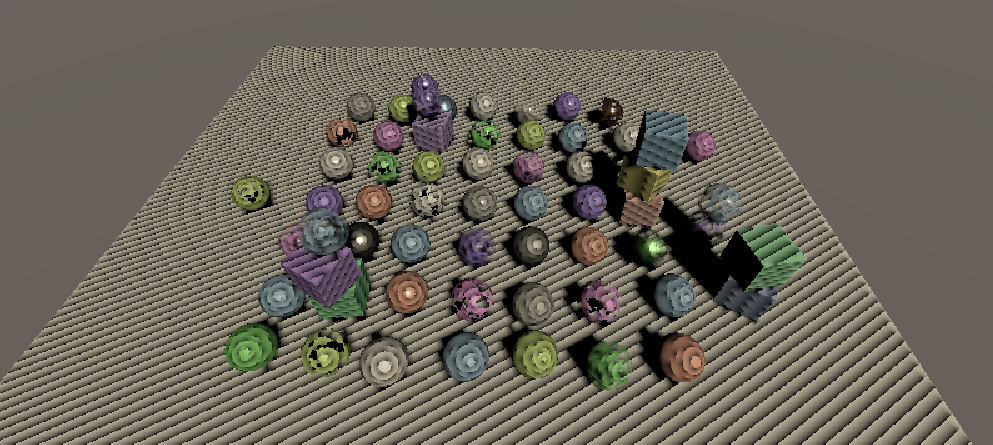
并且，对于不同的方向光，因为我们的最大阴影距离和层级比例是一致的，所以只需要一个阴影的裁剪球体数组即可。



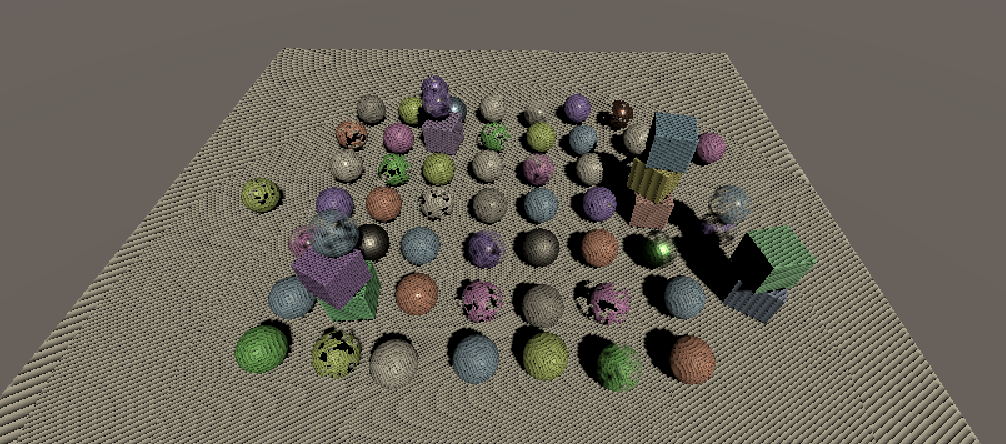
把级联选择的结果作为一个结构体，该结果由片元的世界空间位置决定。用级联选择的结果来决定阴影贴图的选择。

手动更改结果，查看阴影贴图是否选择正确：



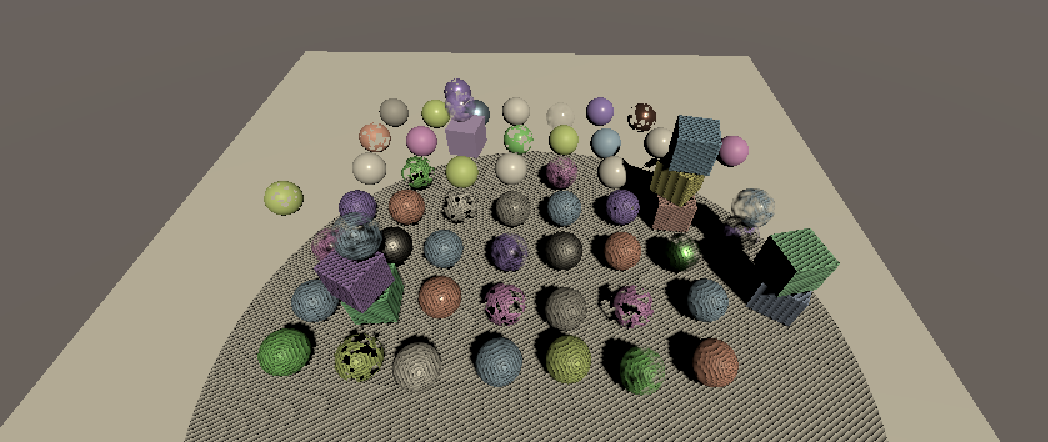


通过判断着色点是否在每一层级的CullingSphere内，确定层级，遍历时先遍历最小的Sphere：



我们还可以返回一个强度，超出级联范围时，返回0，否则返回1。

最大阴影距离为12的结果：

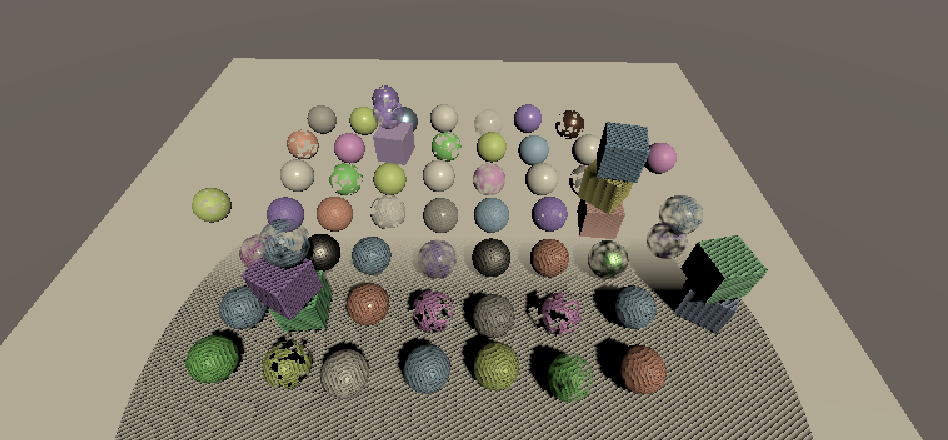


但是，一些物体在最高层级的级联里也没有阴影，我们需要用最大阴影距离来二次修正。

（最大距离是基于ViewSpace的深度的，所以要计算对应深度存在surface里）



我们可以用一个线性方程生成淡出效果，淡出的范围作为配置项放在RP Asset里。

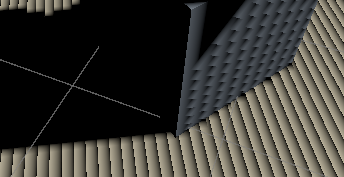
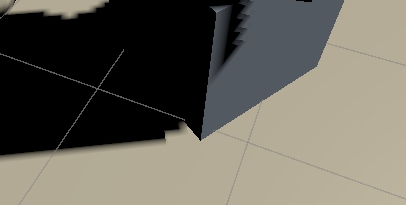


同理，我们还可以给最高级联层级实现阴影淡出效果（该配置放在Directional里）：

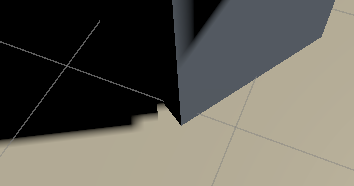


下面要解决自阴影的artifact的问题，最简单的方法是加一个全局bias（SetGlobalDepthBias）。

但是，bias太大会导致阴影离开本体：

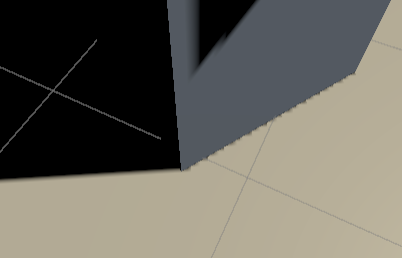
使用slope bias可以解决这一问题，对于正面照亮的表面，应用的bias为0，随着表面法线与光线方向增加，bias增大（此参数用于放大深度的导数）。



但是调整这个参数并不直观。我们撤回该操作，选择其他方法。

这种artifact的大小和级联层级相关，所以我们要记录一些级联信息上传到GPU中。

自阴影的产生是因为阴影贴图的像素大小比一个片元像素大小更大，那么我们沿法线移动表面位置即可。考虑一维情况，移动的距离等于世界空间下的阴影像素大小（可以保证不采样到自身）。Hlsl里实现时，在阴影采样前进行偏移。



可以看到，墙的深度影响到了地板的阴影，所以我们可以添加一些Slope-scale Bias。

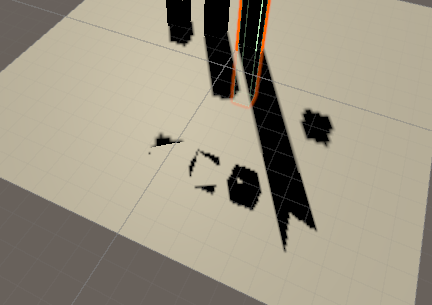
使用光源自带的ShadowBias控制SlopeBias，使用ShadowNormalBias控制刚刚实现的Bias。



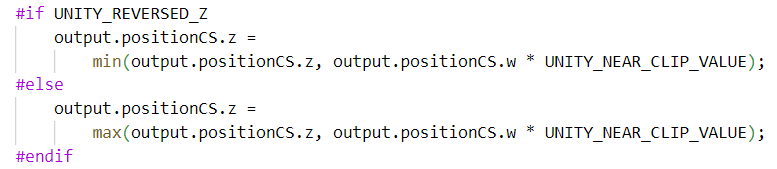


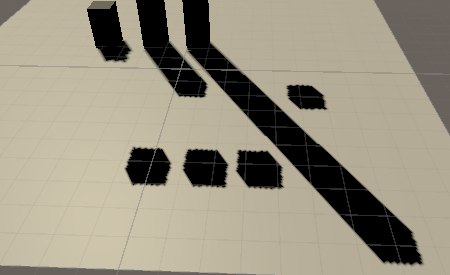
继续解决其他问题：

为了提高深度的精度，unity会把近平面往远处推，这可能导致不必要的裁切：



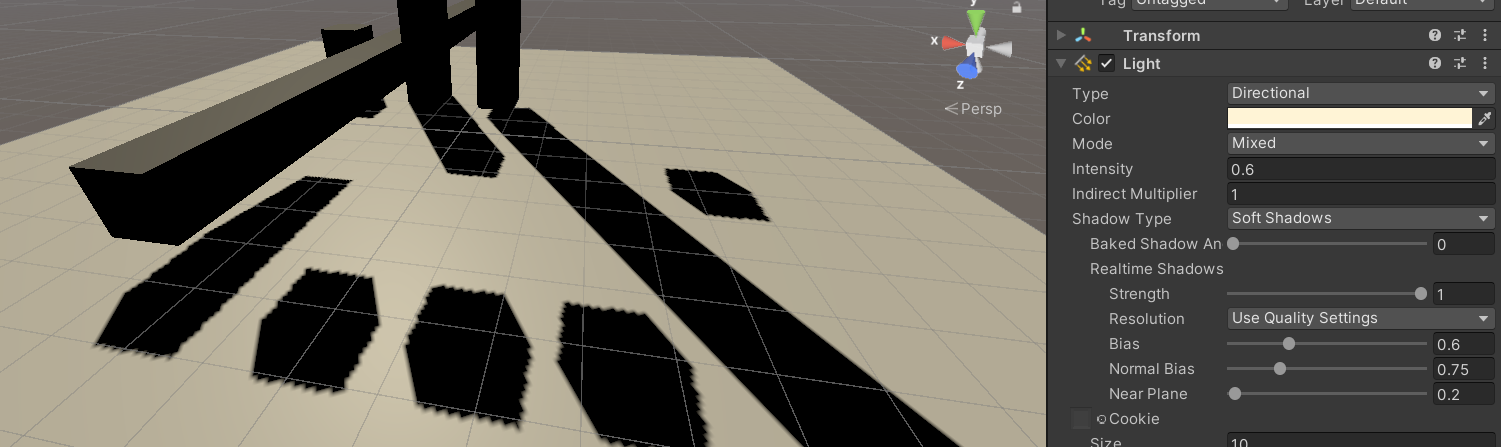
把超出视口范围的物体“贴”在近平面上，就可以绘制阴影了。

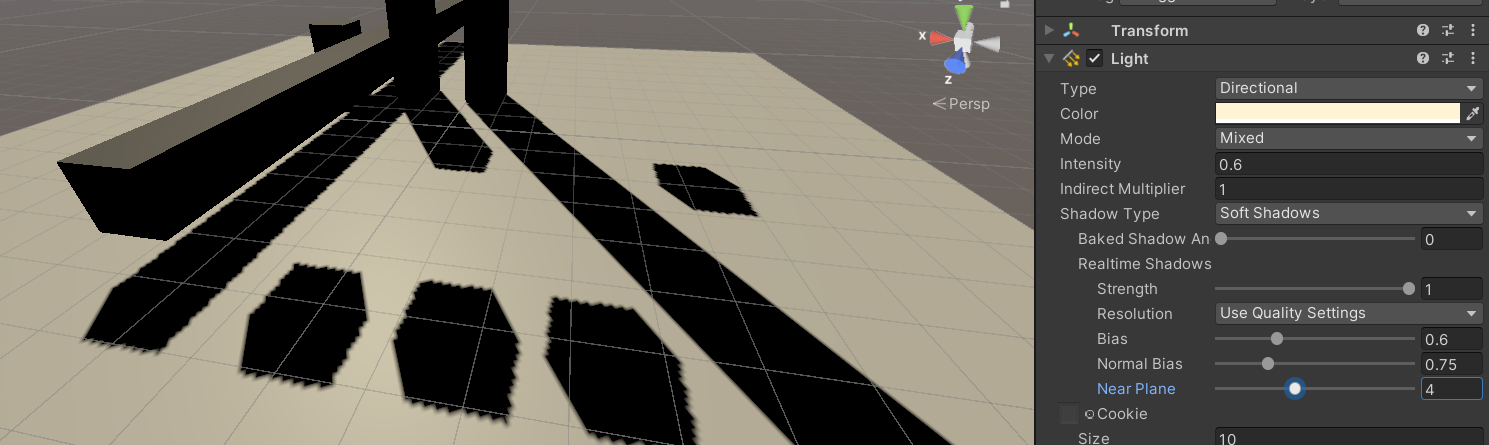




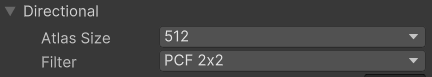
但是如果物体穿过了近平面，可能还是会有问题。

通过获取光源的NearPlaneOffset，并在调ComputeDirectionalShadowMatricesAndCullingPrimitives时设置该变量即可。

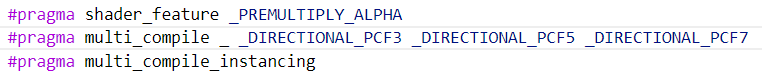




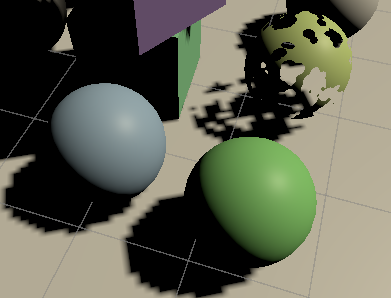
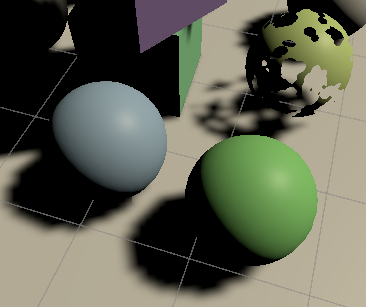
接下来实现软阴影，首先，软阴影涉及多次贴图采样，也就是要设置Filter，我们把它的设置记为全局统一的设置。

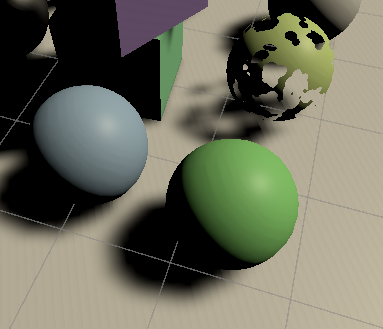
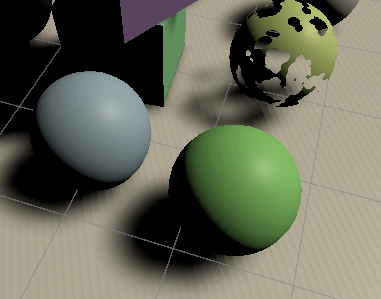


通过该选项控制shader的keyword（默认是2x2的）



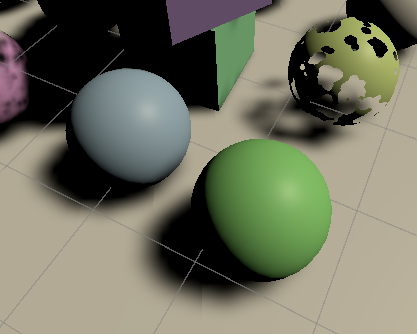
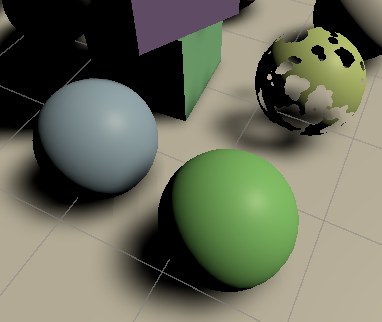
在采样时，根据关键字配置对应的DIRECTIONAL\_FILTER\_SETUP函数，可以根据当前的屏幕空间坐标返回需要采样的点以及各点的权重。然后再调用SampleDirectionalShadowAtlas进行加权求和，得到最终的结果。

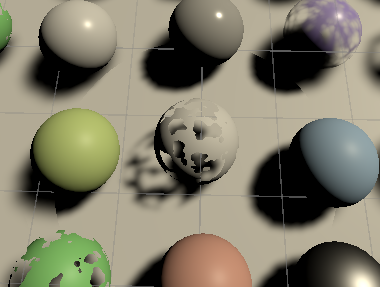
 

这会再次导致自阴影，因为我们把bias增量乘上filter的大小即可。

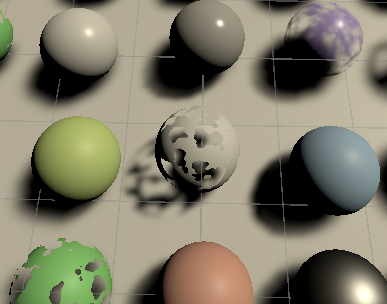
此外，我们的裁切球的半径也要减小filter可能采样到的大小。

接下来，处理软阴影的级联过渡问题：



就像给最高层级淡出一样，对于其他层级，将两个层级的采样结果根据淡出比例混合。



（注意，设置淡化比例时，不要让淡化范围超过两个层级）

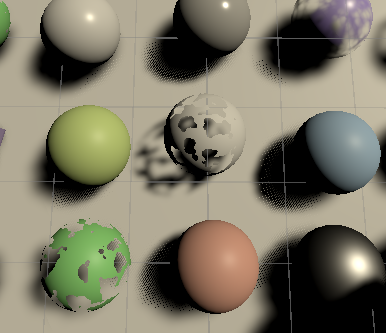
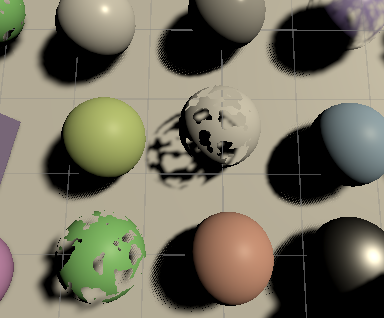
但是该方法需要采样两次，开销较大，我们尝试再实现抖动的方法。

先把这几种模式设置为可配置属性，作为关键字引入。（和PCF Filter一样）

在片元着色器中，根据裁剪空间的xy值，调用InterleavedGradientNoise生成抖动偏移量。

对于该方法，如果前面得到的混合值比抖动偏移量小，则采样下一个层级，否则不改变采样层级。

抖动方法会受阴影贴图分辨率影响。一种改善方法是增加分辨率，另一种是使用TAA。

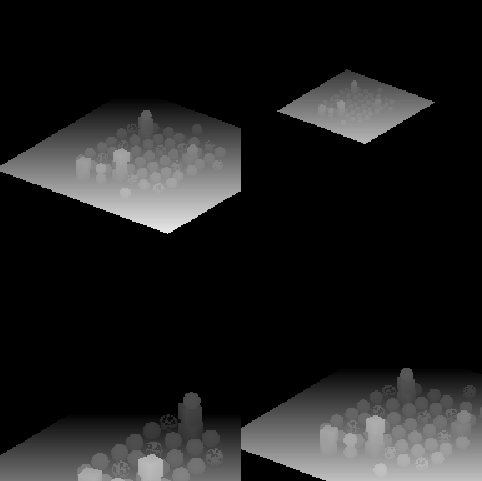
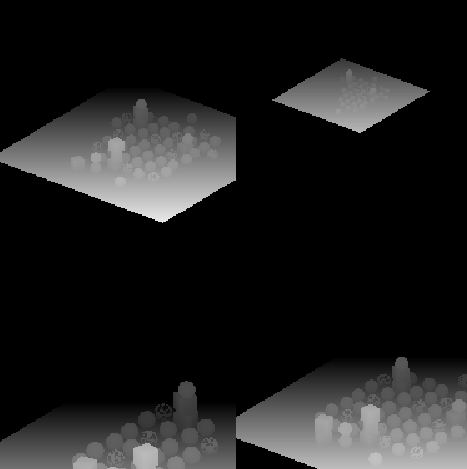
 

抖动方法只需采样一次PCF。

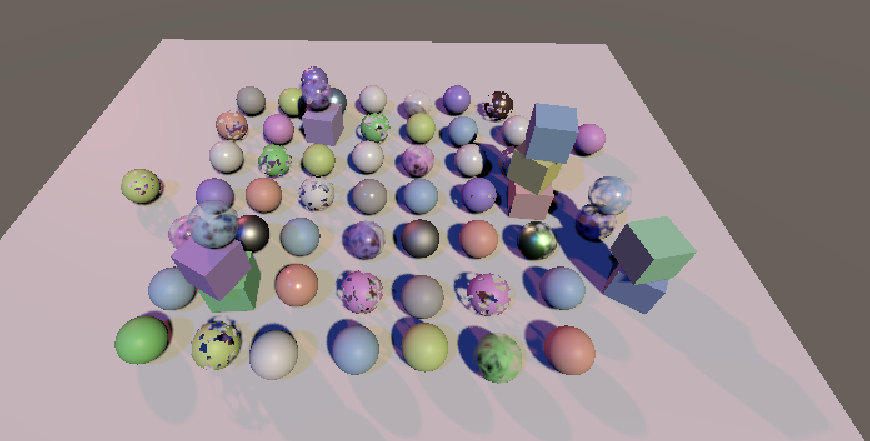
因为更小层级包含的物体不一定需要被更大层级采样到，所以可以设置splitData的CullingFactor来减少阴影贴图生成时的物体数量，该比例应该和cascadeFade相关，因为我们希望淡化区域的物体在两个层级都存在。



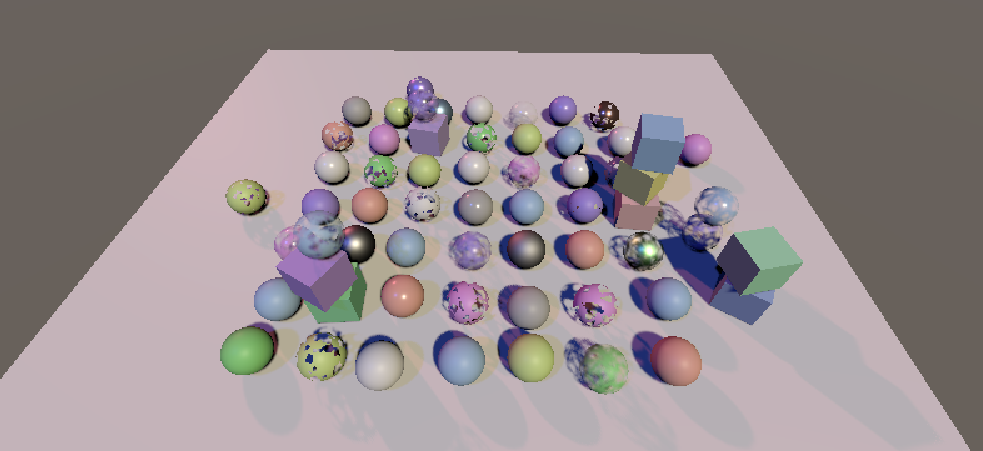
Factor为0和1的效果：

现在的效果：



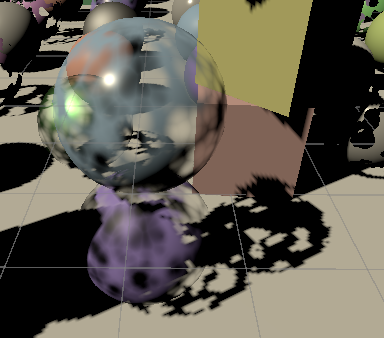
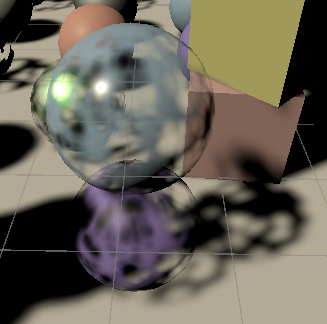
对比他的，就差一个透明物体的处理了：



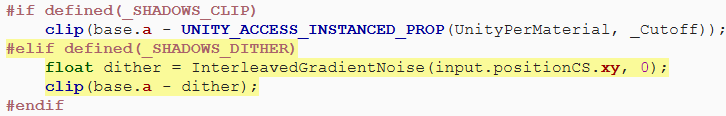
接下来处理透明物体。

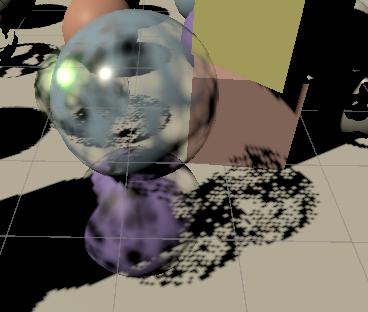
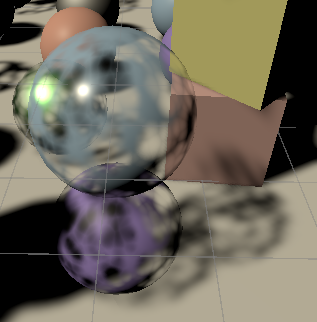
用为材质增加阴影模式的配置，On表示Opaque，Clip表示Clip，Dither表示Fade或Transparent，因为是材质的可配置关键字，所以用shader\_feature而不是multi\_compile（区别见：https://zhuanlan.zhihu.com/p/633432859）

因为和原来材质的alpha\_clip分开了，所以对透明材质而言，可以实现clip的阴影：

 PCF: 

而抖动的阴影则是根据屏幕空间坐标采样噪声，根据噪声值和alpha值clip。

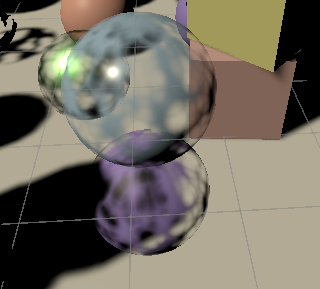


 PCF: 

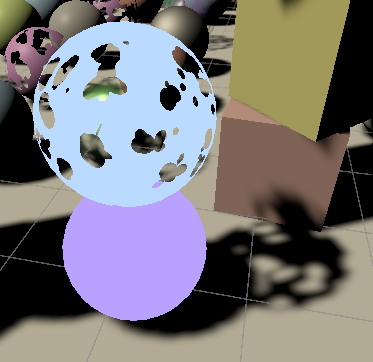
但是Dither的方法受时域影响较大，一般还是用clip的方法生成透明物体的阴影。

下面，通过禁用材质的ShadowCaster Pass实现阴影的关闭。（在前端切换的时候关闭，逻辑在GUI的EndChangeCheck后面）

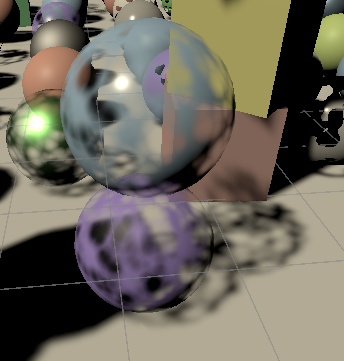
这种关闭是按材质的，按物体的关闭直接修改对应物体的MeshRenderer组件即可。



我们把Lit.Shader里ShadowCaster的部分拷贝到Unlit里就可以实现Unlit材质的阴影了：



我们还可以设置关键字让材质不接收阴影，在GetDirectionalShadowAttenuation那里直接返回即可。



紫色球没有接收左边箱子的阴影

最终版效果：

