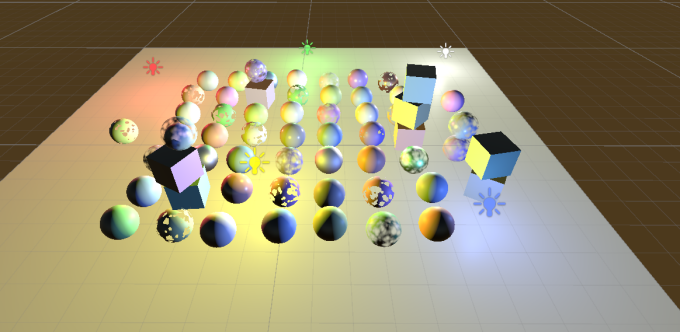
本部分会实现点光源和聚光灯的照明

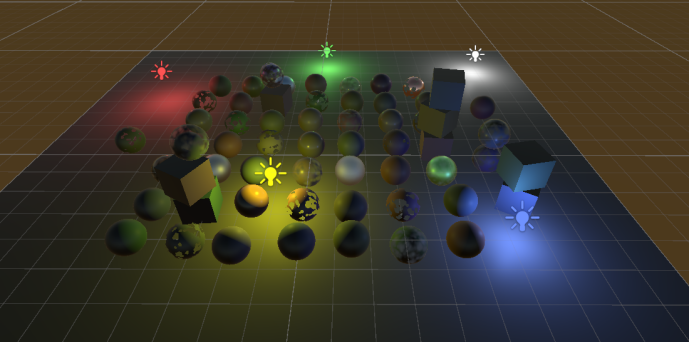
Unity会按照光源重要性排序，我们设置每帧最大的光源数量即可。

我们需要向GPU上传光源数量、颜色和位置数组，仿照方向光写就行。

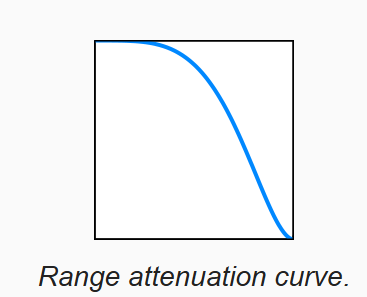
在hlsl里获取光源信息时，依旧返回光源方向，先把其他属性置为默认，查看效果：

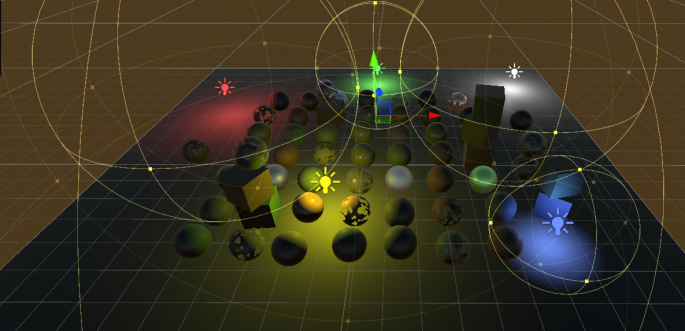


下面需要考虑点光源的平方反比衰减，自带的finalColor表示距离为1时的光源颜色，实现的时候要避免除0：



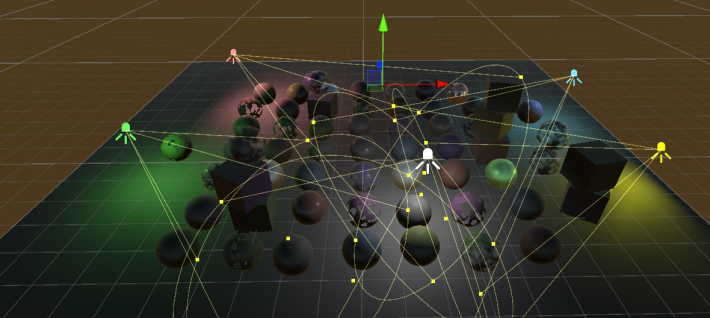
对于超出光照半径的着色点，不应考虑该点光源的贡献，因此加入额外的距离衰减曲线，衰减范围存储在光源位置的w分量上：

 URP使用的函数

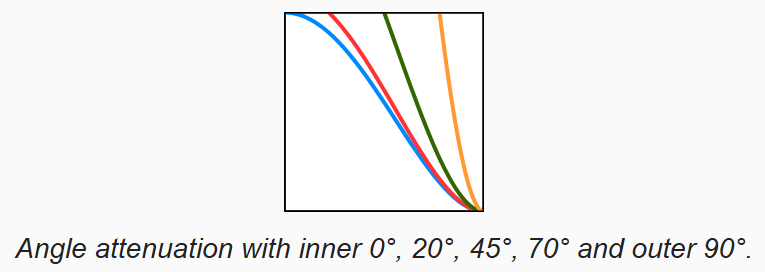


聚光灯可以看成一个点光源，并且被一个带孔的球体遮住。因此，聚光灯多一个方向属性。

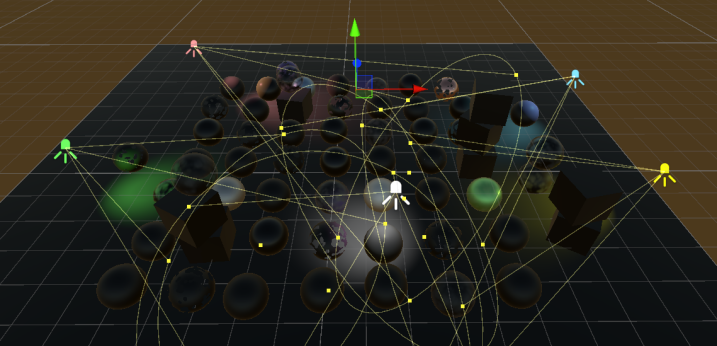
使用点积我们可以得到180度的照明角：



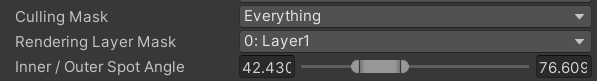
URP通过新的衰减函数实现角度变化：

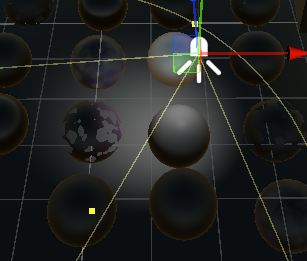
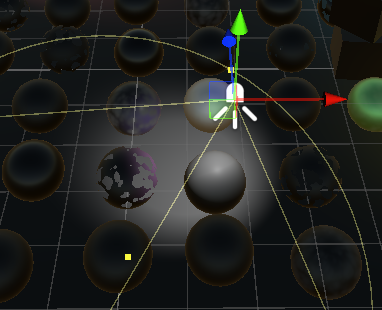
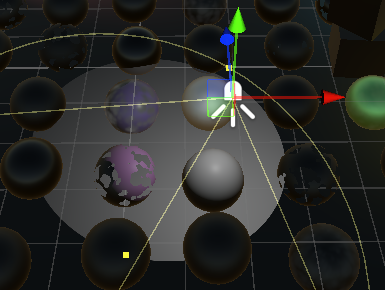


为了实现该函数，我们定义新的SpotAngleArray，用于传输相关信息：

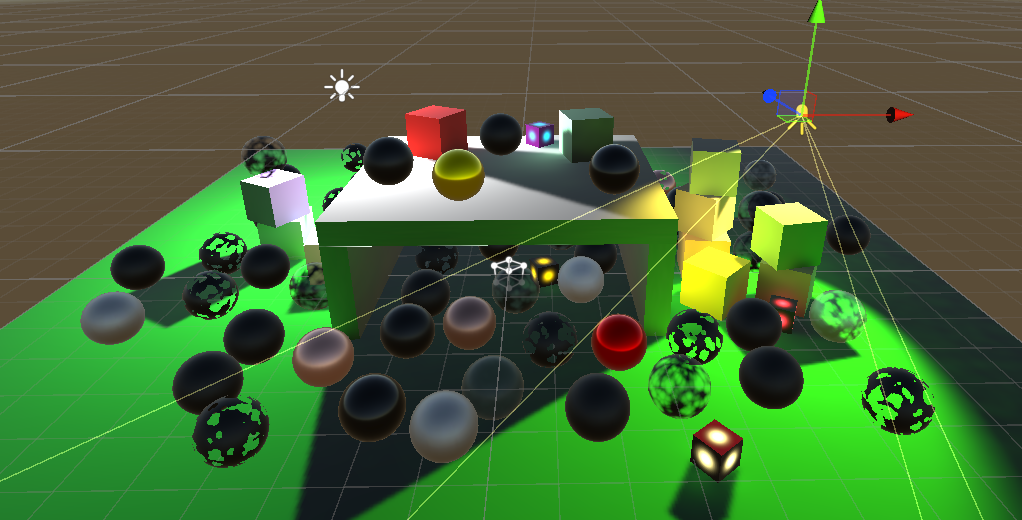


聚光灯的内角是URP引入的，因此我们需要修改RP的默认灯光Inspector，具体代码见CustomLightEditor，先调用默认的editor，再判断当前是否仅选择了spot light，再调用inner /outer spot angle的接口：



下面处理烘焙的问题，在烘焙时，Unity使用的衰减不正确，导致结果过亮：

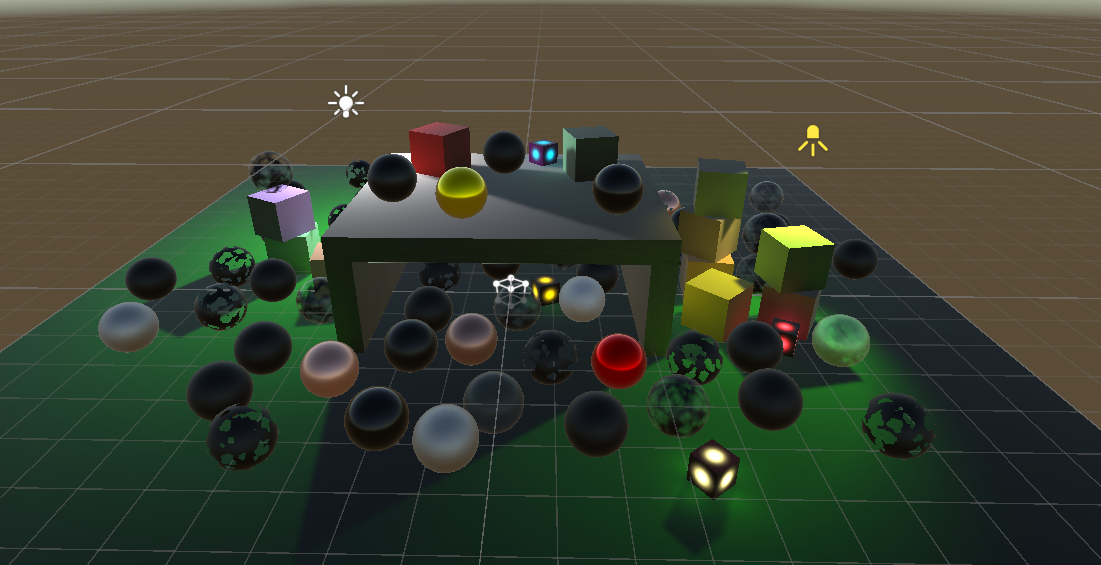


因此，我们要在Unity生成lightmap前通过Custom RP让其执行不同的衰减。通过partial类实现功能的分离，就像CameraRenderer一样。

在指定函数中，遍历光源，并根据类型指定烘焙数据。通过LightmapperUtils.Extract获取光源信息，再通过LightDataGI.Init初始化烘焙信息。

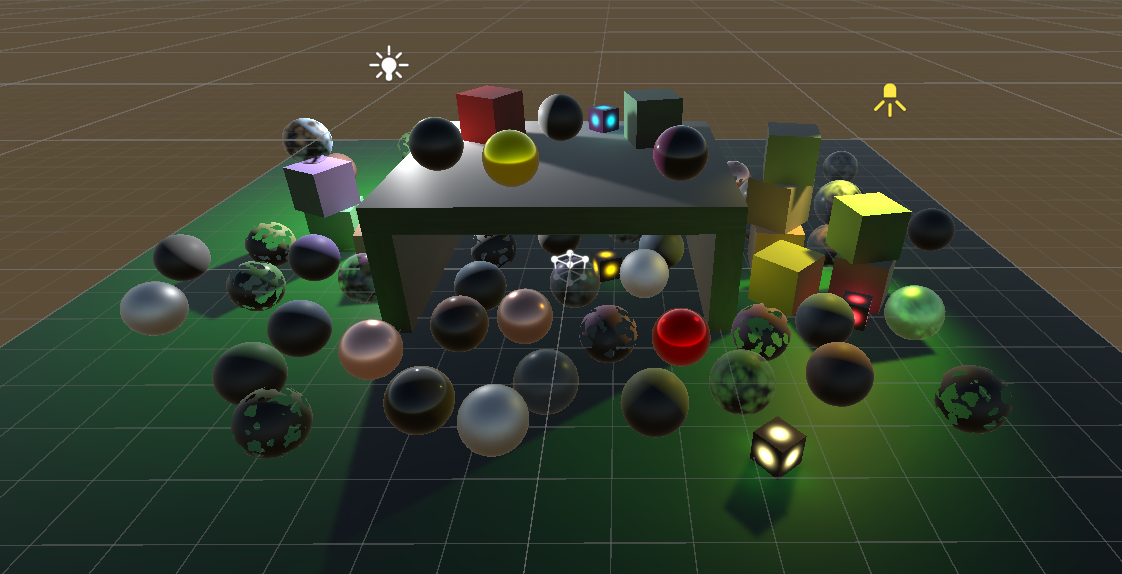
我们可以给面光源强制Baked，让直接光和间接光都烘焙进来。

在输出前，设置光源的衰减方式为FalloffType.InverseSquared：



把光源类型从Baked设置为Mixed的时候，可以看到没有阴影，但是ShadowMask是正确生成的。注意到，一个像素点最多支持四个灯光，超过的灯光会按Baked计算。

在ReserveOtherShadows中设置阴影遮罩数据并上传到GPU，添加GetOtherShadowAttenuation方法计算阴影：



目前，对于每个像素，我们都会遍历所有灯光，这点可以进一步优化，这里我们使用Unity的per-object light indices实现。

这种思路对于小物体表现较好，每个物体单独处理影响该物体的光源信息（而不是每个像素点单独处理）。

为RP Asset添加对应开关，会控制drawSettings里的PerObjectData的Flag。

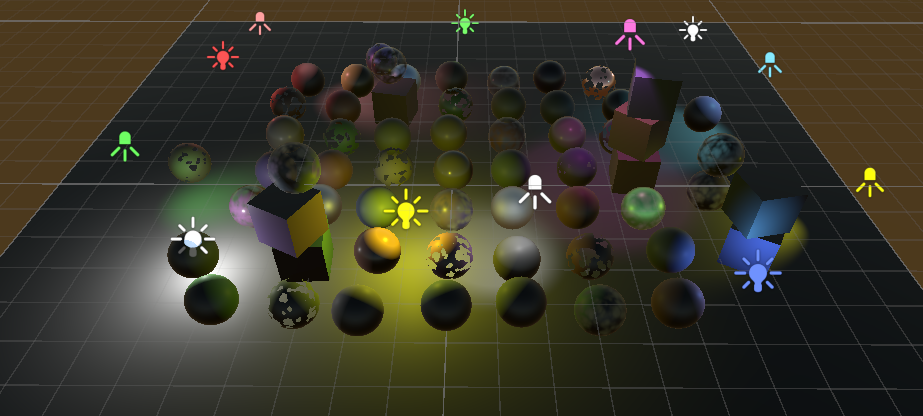
当启用时，我们需要调整IndexMap，让里面只存储Point和Spot Light的Index。

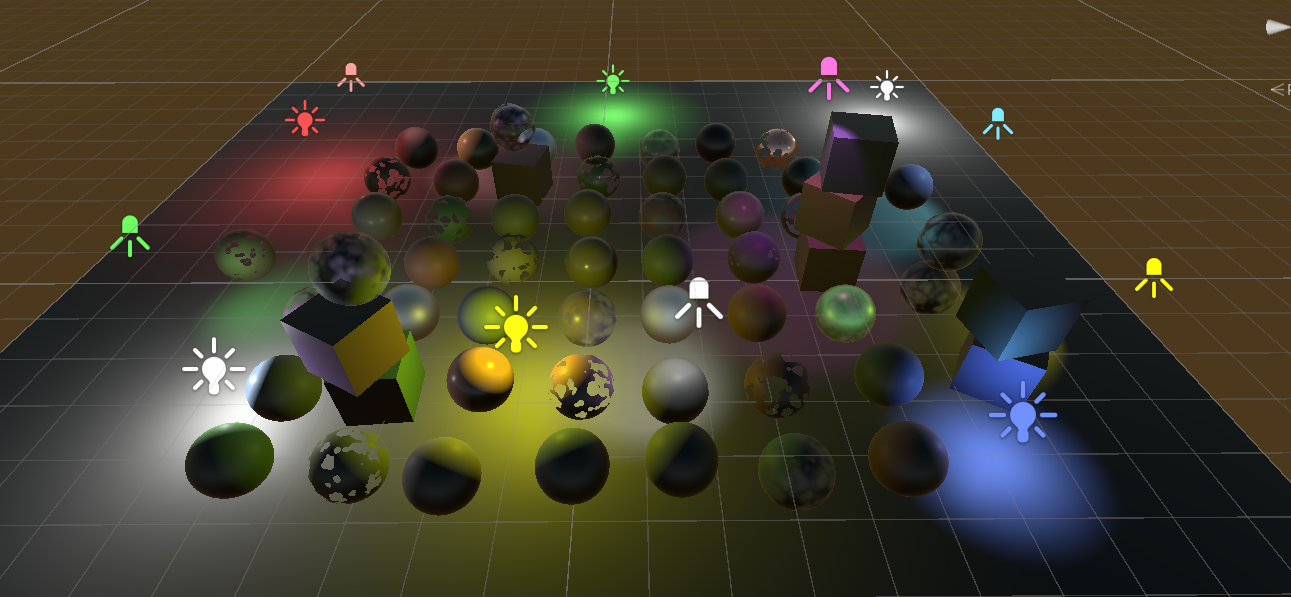
此外，启用时，还需要为Shader设置对应关键字。

在shader中修改GetOtherLight的调用，遍历时通过unity\_LightIndices获取光源索引。

注意到，启用该方法后，GPU instancing的效率会降低。

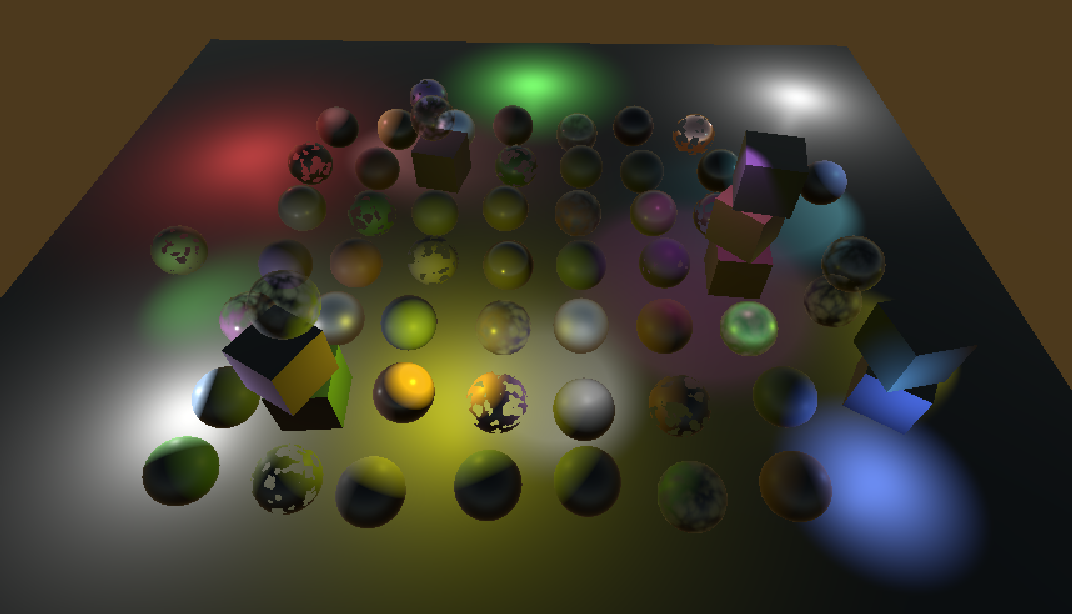
对比启用和不启用：

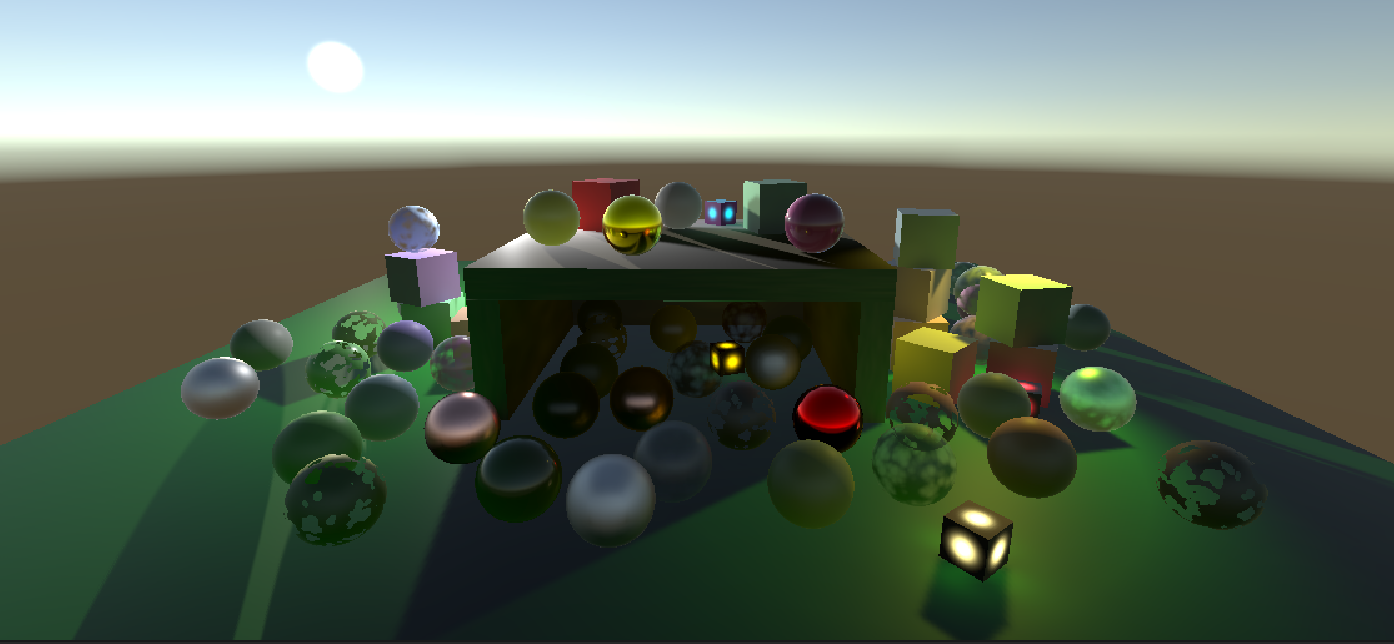




可以看到，对大物体的表现没有那么好。

最终结果：





注：ShadowMask既有LightMap的，也有Light Probe和LPPV的