**点光源和聚光灯**

**点光源**

点光源发出的光线在某点的亮度值与该点到光源距离的平方成反比。

其他光源类型的数据

1. 我们定义非定向光源的数量最大值为64。

2. 定义光源数量，颜色以及光源位置发送到GPU。

光源随距离衰减

光照越远，亮度越低。应遵循公式：i / d2。

其中i为光照强度，d为光照距离，这被称为反平方定律。

限制光照范围

限制光照的最大范围，超过这个范围就将光照强度设为0。​max(0, 1 – (d2/ r2)2)2

**聚光灯光源**

聚光角度

聚光灯有一个角度用来控制光锥的宽度，这个角度是从中间测量的。除此之外，还有一个单独的内角，控线光线以及何时开始衰减公式如下：

saturate(da + b)2

a = 1 / (cos(ri / 2) – cos(ro / 2))

b = -cos(ro / 2)a

配置内角

聚光灯始终可以配置外角角度，但在URP被引入之前是没有单独的内角的。所以灯光的Inspector面板中没有暴露内角角度，渲染管线可以通过覆盖灯光的Inspector面板来修改灯光，这是通过创建编辑器脚本来扩展LightEditor，且给它CustomEditorForRenderPipeline属性完成这个操作。该属性第一个参数必须是Light类型，第二个参数是我们希望覆盖Inspector面板的渲染管线资产类型。

1. 在CustomRP的Editor子文件夹下创建CustomLightEditor脚本。

2. 要替换Inspector面板，首先重写OnInspectorGUI方法，我们需要做的额外操作就是首先检查是否仅选择了聚光灯，通过settings中的属性可以进行光源类型的判断，然后调用DrawInnerAndOuterSpotAngle方法绘制一个调节内外聚光角度滑块，最后调用ApplyModifiedProperties应用该滑块所做的修改即可。

烘焙光照和阴影

烘焙光照

只需要将点光源和聚光灯的灯光组件Mode属性改为Baked，进行烘焙即可（若要烘焙阴影，修改Shadow Type选项）。然后会发现烘焙后光照比较亮，因为Unity默认使用了错误的灯光衰减，和旧版渲染管线的结果相匹配。

1. 我们可以告诉Unity使用不同的衰减，通过在Unity编辑器中执行光照烘焙之前提供一个委托方法。

2. 仅对于编辑器，需要重写lightmapper设置光照数据，通过提供一个委托方法，来传入一个Light数组。最后输出一个NativeArray<LightDataGI>结构委托的类型是 Lightmaing.RequestLightsDelegate。

3. 根据不同的光源类型，创建一个LightDataGI结构，调用LightmapperUtils的Extract方法，参数是光源和光源引用结构，然后调用光源数据的Init方法。

4. 对所有的灯光数据的衰减类型设置为FalloffType.InverseSquared。

5. 调用Lightmapping.SetDelegate方法，把我们定义的委托作为参数传递过去。

6. 当我们的渲染管线被处理时我们还需要清理和重置委托，通过重写Dispose方法，先进行清理，然后调用Lightmapping.ResetDelegate来重置委托。

阴影蒙版

把点光源和聚光灯的Mode设置为Mixed也能将阴影烘焙到ShadowMask中。每个光源都使用一个通道，就像方向光一样。但由于其范围有限，因此多个光源可以使用同一通道，只要它们不重叠。因此，阴影蒙版可以支持任意数量的光，但每个纹素最多只能支持四个。如果多个光源在尝试声明同一通道时重叠，那么最不重要的灯将强制设置为Baked模式，直到不再发生冲突。

(代码实现)

逐对象光源

目前，所有可见光都会渲染对象的每个片元，这对于方向光来说很好，但非定向光源通常只影响该物体表面的一小部分片元，因此许多计算都是多余的，而且会影响渲染效率，我们需要减少每个片元的评估光源数量，使用Unity的逐对象光源索引。

Unity确定哪些灯光会影响哪些对象，并将此信息发送到 GPU。然后在渲染每个对象时，只评估相关灯光，而忽略其它对象。因此灯光是根据每个对象而不是每个片元确定的。这通常适用于小对象，但不适合大型对象，因为如果光线只影响对象的一小部分，则它将评估其整个表面。此外，有多少灯光会影响每个对象是有限制的，因此较大的对象更容易缺少照明。

由于逐对象的光源索引不一定理想，可能会丢失一些照明，因此我们把它作为可配置选项，这样可以更方便地比较视觉效果和性能开销。

1. 给CameraRenderer脚本的DrawVisibleGeometry方法添加一个bool参数以指示是否使用逐对象光源，如果为true则绘制设置时启用PerObjectData.LightData或PerObjectData.LightIndices。

2. Unity会对每个对象创建一个活跃光源列表，这个列表包含了场景内所有存在的光源（无论该光源是否可见），并且包含方向光，然后根据光源的重要性进行了排序。下面我们清理这些灯光列表，只保留可见的非定向光源索引。

3. 在循环可见光的时候，只需要包含点光源和聚光灯的索引，所有其它类型的光源应跳过（将其索引设置为-1即可）。然后我们需要更改剩余的光源的索引以匹配我们的光源。

4. 然后还需要消除所有不可见光源的索引，这里添加一个for循环在第一个循环完成后执行此操作，完成后必须通过cullingResults调用SetLightIndexMap方法将调整后的光源索引列表发送回Unity，这时indexMap就不需要了，调用它的Dispose方法进行释放。

6. 在UnityInput.hlsl的UnityPerDraw缓冲区中声明2个相关的属性，其中unity\_LightData的Y分量中包含了灯光数量，unity\_LightIndices的两个分量都包含一个光源索引，所以每个对象最多支持8个。

7. 在Lighting.hlsl的GetLighting方法中判断\_LIGHTS\_PER\_OBJECT关键字是否被定义。如果定义了，则使用unity\_LightData的Y分量中的光源数量进行循环，且从unity\_LightIndices中检索出合适的光源索引，可以通过将迭代器除以4和通过与4取模得到正确的向量（此时，着色器的编译器可能会提示整数除法和取模操作速度慢，可以通过将迭代器j转换成uint类型来忽略提示）。我们最多有8个光源索引可以使用，但Y分量中存储的灯光数量可能会超过该数值，我们在循环时对其进行一下限制。

使用逐对象光源会使得GPU Instancing的批处理效率变低，因为只有灯光计数和光源索引列表相匹配的对象才会被分组，SRP Batcher并不受影响，每个对象仍然拥有自己优化后的Draw Call。