点光源和聚光灯

点光源

点光源是一个无限小的点，其照亮的范围是一个球体的光源，光的亮点随着光源的距离变大而逐渐变小，当超出照亮范围时亮度值为0，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. 跟方向光一样，我们场景中支持的其它类型的光源数量也是有限制的，非定向光源的照射范围有限。通常对于任何给定帧，我们只能看到非定向光线的子集，因此我们可以支持的最大值适用于单帧，而非整个场景。如果在某个范围的光源数量比我们设置的最大数量多，则应该忽略掉多余的光源，Unity会根据重要性对可见光源列表进行排序，如果光源不发生变化，哪些光源被忽略掉是固定的。如果有相机的移动或其它改变，有可能导致曝光的情况，所以最大光源数应定的高一些，我们在Lighting脚本中定义非定向光源的数量最大值为64。

2. 就像方向光一样，GPU需要知道场景中的光源数量，颜色以及光源位置，我们定义这三个属性的着色器标识ID，然后定义两个数组存储每个光源颜色和位置数据。

3. 在SetupLights方法中追踪定向光和非定向光的计数，若光源数量大于0，将相关光源数据发送到GPU。

4. 接下来在Light.hlsl中定义一个代表最大光源数量的宏，并在\_CustomLight缓冲区中声明其它光源类型的颜色和位置属性，然后定义一个GetOtherLightCount方法返回非定向光源的数量。

点光源的支持

1. 在Lighting脚本中定义一个SetupPointLight方法，将点光源的颜色和位置信息存储到数组。

2. 调整SetupLights方法中的代码，使用switch语句区分光源的类型，在未到达最大光源数量之前根据光源类型进行数据的存储。

3. 现在点光源数据已经传递到GPU了，在Light.hlsl中定义一个GetOtherLight方法获取指定索引非定向光源的颜色和方向的数据，现在不支持投影，所以阴影衰减值为1。

4. 最后在Lighting.hlsl的GetLighting方法添加一个for循环，遍历所有非定向光源获取照明结果。

光源随距离衰减

现在我们场景中物体可以受点光源影响了，但是现在很亮，光照强度应随着距离而进行衰减，光照越远，亮度越低。应遵循公式：i/d2。

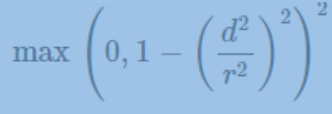
其中i为光照强度，d为光照距离，这被称为反平方定律。下面是距离衰减曲线，我们可以知道离光源距离小于1的时候，可能会变得非常亮。

1. 在Light.hlsl的GetOtherLight方法中，利用1除以光照距离的平方应用距离衰减，另外要保证距离的平方值不为0，给它设置为一个很小的正值。

限制光照范围

尽管现在点光源强度随着距离的增加衰减的很快，但理论上光照还影响着所有物体，尽管它通常无法明显感知出来，漫反射不明显，但镜面反射在更远的距离有时候仍然可见。

我们要让渲染变得更有效率，需要限制光照的最大范围，超过这个范围就将光照强度设为0。点光源包含在一个包围球中，球体由光源位置和范围而定，且球体边界的光照不应突然消失，而应该通过距离衰减平滑过渡。Unity的URP和烘焙系统使用了下面的公式来定义距离衰减曲线，其中r是光照的范围，我们也会使用这个公式。​



1. 在Lighting脚本的SetupPointLight方法中，将光照范围的平方的倒数存储在光源位置的W分量中，存储计算好的值是为了减少着色器的计算量。

2. 然后在Light.hlsl的GetOtherLight方法中套用公式计算，最终使得光照强度随着范围和距离进行衰减。

聚光灯光源

聚光灯光源和点光源类似，它也有光源位置和作用范围，区别是照亮范围是一个圆锥体，光源位置在圆锥体的锥顶处。

光照方向

1. 聚光灯还有一个方向属性，在Lighting脚本中定义非定向光源的光照方向着色器标识ID，和存储方向数据的数组。

2. 在SetupLights方法中把方向数据传到GPU。

3. 新建一个SetupSpotLight方法，内容和SetupPointLight方法差不多，这里通过转换矩阵的第三列并求反得到光源的光照方向并存储。

4. 然后在SetupLights方法的switch分支判断中添加一个聚光灯case。

5. 在Light.hlsl的\_CustomLight缓冲区中声明一个\_OtherLightDirections数组接收光照方向数据。

6. 最后在GetOtherLight方法中计算聚光灯的衰减值，先通过聚光灯方向和光照方向点积得到光照衰减，使得聚光角度在90度时达到0，照亮灯光前方的一切。

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，所以一个90度的角度看起来就像现在的一样。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，URP和lightmapper通过在saturate之前对点积结果缩放和添加一些东西，然后对结果进行平方来做到这一点，公式如下：

saturate(da + b)2

(见书本)

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，内容如下。

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。将CustomRenderPipeline改为内部类，然后在构造函数的末尾调用InitializeForEditor方法。

2. 新建脚本CustomRenderPipeline.Editor，作为CustomRenderPipeline编辑模式的局部类，内容如下。

3. 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，我们将使用lambda表达式定义该方法，因为在其它地方不需要它。

4. 我们必须为每个光配置一个LightDataGI结构，并将其添加到输出中。因为需要为每个光源类型使用不同的处理代码，所以添加switch语句。默认情况下调用光照数据的InitNoBake方法，传入光源的实例ID，指示Unity不要烘焙光照。

5. 接下来根据不同的光源类型，创建一个专业的光源结构，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。现在对所有类型的光源执行此操作，我们目前不支持区域光源，如果存在，需要把强制把该光源的Mode属性设置为烘焙模式。

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

7. 现在要使Unity调用我们写好的委托，需要创建InitializeForEditor方法，其中调用Lightmapping.SetDelegate方法，把我们定义的委托作为参数传递过去。

8. 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。

1. 要将阴影蒙版用于点光源和聚光灯，先在Shadow脚本中定义一个ReserveOtherShadows方法，用于存储非定向光源的阴影数据。如果混合光源的模式为ShadowMask，只需要配置阴影强度和Mask通道。

2. 在Lighting.cs中添加非定向光源的阴影数据的着色器标识ID和数组。

3. 在SetupLights方法中将该数组发送到GPU。

4. 在SetupPointLight和SetupSpotLight方法中存储点光源和聚光灯的阴影数据。

5. 在Shadows.hlsl中定义一个代表非定向光源的阴影数据的结构体，和一个GetOtherShadowAttenuation方法，我们使用和方向光阴影相同的方法。如果阴影强度大于0，则总是调用GetBakedShadow方法，否则阴影衰减为1，表示没有阴影。

6. 在Light.hlsl的\_CustomLight缓冲区中声明非定向光源的阴影数据数组，定义一个GetOtherShadowData方法获取阴影强度和Mask通道，然后在GetOtherLight方法中计算非定向光源的光照衰减时乘以光源的阴影衰减。