**4 方向阴影**

**4.1 渲染阴影**

确定阴影区域的方法是把光源想象成一个摄像机。用光源相机执行一次取景操作，使用LightMode标签为ShadowCaster的Pass，把在光源相机所在角度所有可视的片元深度信息存储在一个帧缓冲区中，称为阴影贴图（Shadow Map），其本质是一张深度图。在渲染时把每一个待输出片元再次放到光源相机的角度下计算深度值，如果计算的深度值比阴影贴图的深度值要远，就表示它落在阴影区域中。

**4.1.1 设置阴影**

1. 在渲染阴影之前，我们需要配置一些属性，比如阴影的最大距离和阴影贴图的大小。

**[Serializable]和[SerializeField]**

2. 在Culling方法中，通过传入的阴影最大距离和相机的远截面进行比较，将小的那个作为渲染管线的最大阴影距离。

**4.1.2 创建阴影类**

把阴影的处理逻辑抽离到Shadows.cs脚本。

**4.1.3 带阴影的光源**

在Shadows脚本定义一个字段来限制可以投影的定向光源数量。

**4.1.4 创建阴影图集**

Frame Debugger折叠标签

1. 创建一张阴影图集，渲染纹理的类型指定为Shadowmap。

2. 在相机渲染完后释放临时渲染纹理。

3. 创建渲染纹理后调用buffer.SetRenderTarget方法来指定渲染数据存储到渲染纹理而不是帧缓冲区中，最后调用buffer.ClearRenderTarget方法清除渲染目标的深度数据。

​4. 调整阴影渲染的时机，应在相机正式渲染场景之前渲染阴影。

**4.1.5 渲染阴影**

1，要渲染阴影，首先要创建一个ShadowDrawingSettings实例。

2. 方向光并没有一个真实位置，我们要找出与光的方向匹配的视图和投影矩阵，调用cullingResults.ComputeDirectionalShadowMatricesAndCullingPrimitives方法，它需要9个参数。第1个是可见光的索引，第2、3、4个参数用于设置阴影级联数据，第5个参数是阴影贴图的尺寸，第6个参数是阴影近平面偏移。最后三个参数都是输出参数，一个是视图矩阵，一个是投影矩阵，一个是ShadowSplitData对象。

3.调用buffer.SetViewProjectionMatrices方法应用获取的视图和投影矩阵。最后执行缓冲区命令并调用context.DrawShadows方法渲染阴影投射。

**4.1.6 ShadowCaster Pass**

1. DrawShadows方法只渲染Shader中带有ShadowCaster Pass通道的物体，我们在Lit.shader中添加第二个Pass块， LightMode设为ShadowCaster。这个Pass只需要写入深度数据，所以添加ColorMask 0不写入任何颜色数据，但会进行深度测试，并将深度值写到深度缓冲区中。

2. 片元函数不需要返回任何值，唯一的作用就是裁剪不满足阈值片元。

**4.1.7 支持多光源渲染阴影**

在Shadows脚本中将最大投影灯光数量改为4。

1. 每盏灯渲染到整个阴影图集时，它们会叠加到一起，所以需要拆分图集。

2. 通过命令缓冲区调用SetViewPort方法，偏移量根据图块大小进行缩放，然后以图块大小作为该视口矩形的宽和高。

3. 在RenderDirectionalShadows(int index, int split, int tileSize)方法中设置视图投影矩阵之前调整渲染视口。

**4.2 采样阴影**

渲染了阴影贴图后，还需要对贴图进行采样才能接收物体的投影。

**4.2.1 阴影转换矩阵**

1. 对于每个片元，我们必须从阴影图集中对应的阴影图块采样深度数据，因此需要找到对应在世界空间的阴影纹理坐标，可以通过为每个可投影的定向光创建阴影转换矩阵并将其发送到GPU来实现这点。

2. 通过将获得的光源的投影矩阵和视图矩阵相乘，可以创建一个从世界空间到灯光空间的转换矩阵。

4. 因为我们使用的是阴影图集，它可能包含多个阴影贴图图块，所以需要创建一个ConvertToAtlasMatrix方法来传入世界空间到灯光空间的转换矩阵、图块偏移和图块拆分数，最终得到一个从世界空间到阴影图块空间的转换矩阵。

**4.2.2 获取方向光的阴影数据**

1. 要对光源的阴影采样，需要知道它在阴影图集的图块索引。

**4.2.3 阴影图集采样**

1. 创建Shadows.hlsl文件用来对阴影图集进行采样。因为图集不是常规的纹理，所以用TEXTURE2D\_SHADOW宏进行定义。

通过SAMPLE\_TEXTURE2D\_SHADOW宏对阴影图集采样，它需要图集和采样器，以及阴影纹理空间中的表面位置。

2. 然后定义GetDirectionalShadowAttenuation方法计算阴影衰减，通过转换矩阵将表面位置从世界空间转换到阴影纹理空间，然后对阴影图集进行采样。

3. 采样阴影图集的结果是根据有多少光到达表面来决定的，它是一个［0，1］区间的值，通常叫做阴影衰减因子。如果片元完全被阴影覆盖则为0，如果没有任何阴影遮挡则为1，之间的值表示片元被部分阴影遮挡。

还有一种情况，当灯光的阴影强度属性被降到0时，阴影衰减就不受阴影影响了，衰减值始终为1，所以最终的阴影衰减应该是阴影强度和采样图集得到的衰减因子进行插值得到的。我们调整GetDirectionalShadowAttenuation方法的返回值，且灯光的阴影强度为0时采样阴影图集没有意义，阴影衰减值始终为1。

**4.2.4 灯光的阴影衰减**

1. 在Light结构体中添加一个光源的阴影衰减属性。

2. 在Light文件中定义GetDirectionalShadowData方法来获取CPU传递过来的方向光阴影数据。

3. 给GetDirectionalLight方法添加一个表面信息的传参，然后得到光源的阴影数据并计算阴影衰减。

4. 在Lighting.GetLighting方法中将表面信息surfaceWS传递到GetDirectionalLight方法中。

5. 在IncomingLight方法中将灯光的阴影衰减添加到入射光强度的计算中。

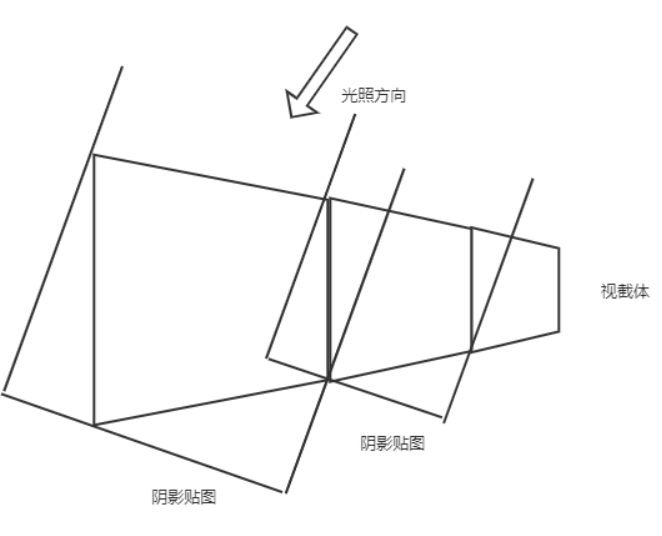
虽然得到了阴影，但是阴影质量很差，不应被投影的表面最终被形成像素化带的自阴影伪影所覆盖，即使提高阴影图集的分辨率只能有所改善，但也不能消除。灯光能够渲染阴影到屏幕中只是实现阴影的第一步，本节后续内容会逐渐完善阴影的各种功能和改善阴影质量。

4.3 级联阴影贴图

透视走样指的是阴影越靠近相机，其边缘的锯齿化越严重。

透视走样的原因：阴影贴图的分辨率是固定的，同样大小的一个阴影所对应的阴影贴图中纹素大小也是固定的。如果使用透视相机，其效果是近大远小，在渲染时，阴影越靠近摄像机，越容易出现多个片元从阴影贴图中的同一纹素进行采样的情况，这几个片元得到的是相同的阴影值，从而产生锯齿边。

级联阴影贴图就是解决这个问题的，它将摄像机的视截体按一定比例分成若干层（Cascade），每个层级对应一个子视截体，每一层都单独计算相关的阴影贴图，在渲染大场景时就可以避免使用单张阴影贴图的各种缺点。



4.3.1 设置级联

我们的管线中为每个方向光支持最多4个级联。在ShadowSettings脚本的Directional结构体中定义一个级联数量滑块。我们定义3个参数来调节前三级级联的区域比例，最后一级会覆盖整个区域。

我们要将每级级联的比例传递到ComputeDirectionalShadowMatricesAndCullingPrimitives方法中，我们将三个比例值封装到一个Vector3向量中。

4.3.2 渲染级联

1. 每个级联都需要自己的阴影转换矩阵，所以转换矩阵的数组的大小应是方向光数量乘以最大级联数量。

2. 在Shadows.hlsl中也定义一个代表级联最大数量的宏和扩张数组大小。

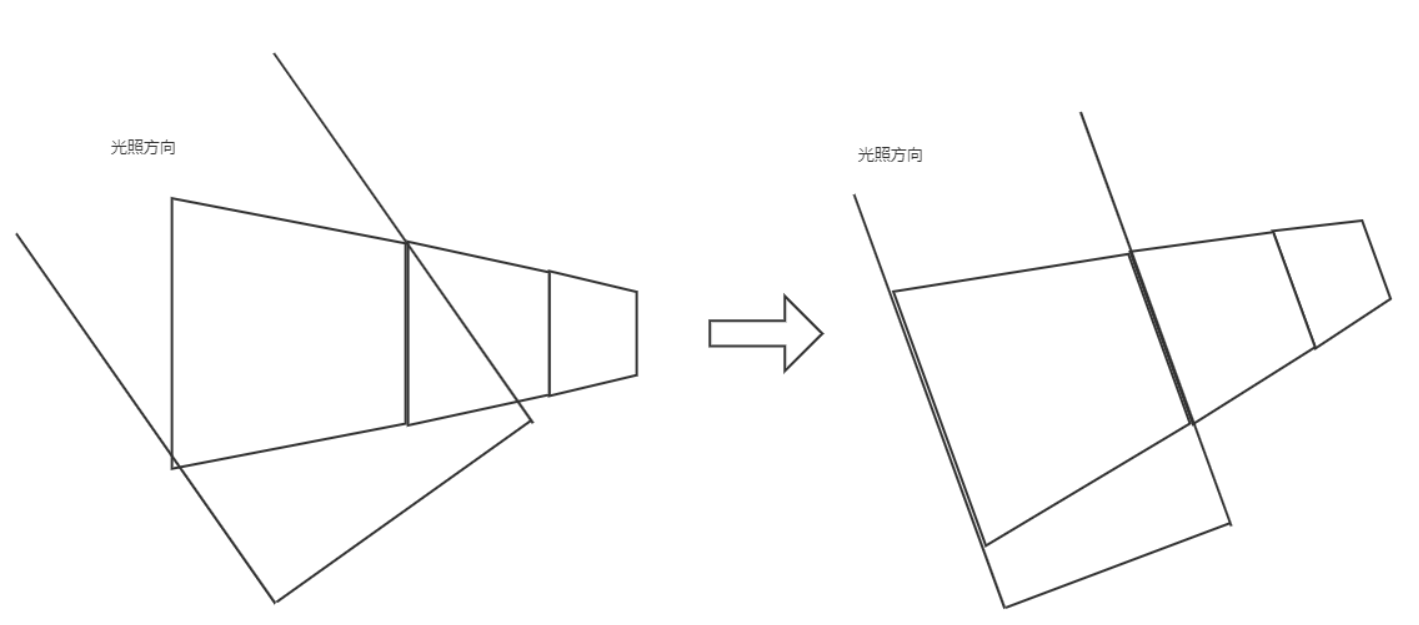
3. 修改ReserveDirectionalShadows方法中阴影图块的索引，需要乘以级联数量。

4. 现在整个阴影图集可以分割成最多4X4=16个图块，因此调整RenderDirectionalShadows方法中的分割图块数量和图块大小。

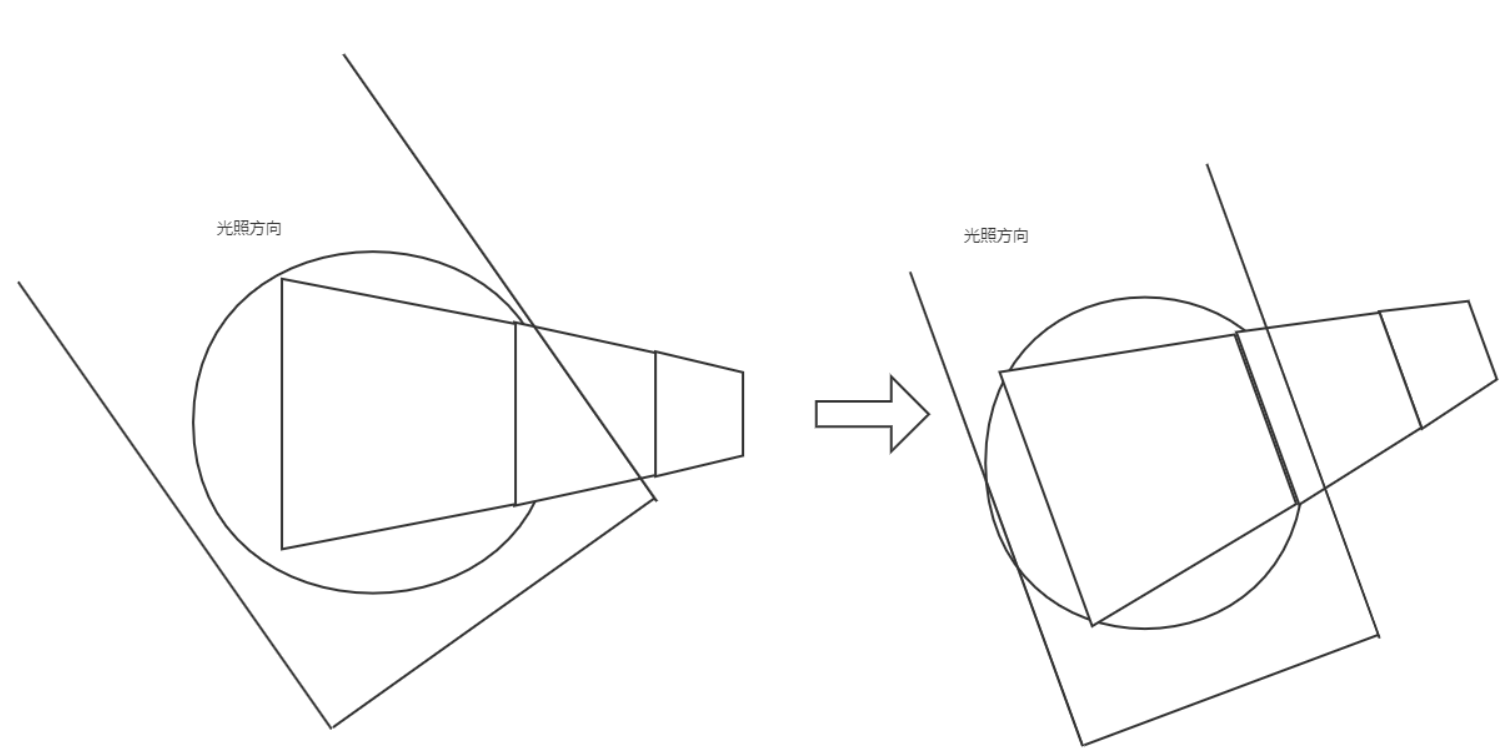
5. 在RenderDirectionalShadows方法中，要为每个级联渲染阴影，需要循环调用ComputeDirectionalShadowMatricesAndCullingPrimitives()方法。之前我们说这9个传参中，第2个参数是当前级联的索引，第3个参数是级联的数量，第4个参数是Vector3类型的各级联比例，我们从阴影配置中拿到它们并设置。最后我们还需要调整图块索引，它等于光源的图块偏移加上级联的索引。

4.3.3 级联包围球

要使用级联阴影，首先要为每个层级对应的子视截体构造一个投影矩阵，构建投影矩阵时，必须是在生成的阴影贴图中，并尽可能减少当前不在视野内的无关区域，也就是说要计算出和当前层级所对应的子视截体尽可能重合的投影矩阵，投影矩阵一般用正交投影，是一个能包住子视截体的且与光源空间坐标系轴对齐的包围盒（AABB）所对应生成的，如下图：



但因为在渲染时，摄像机的位置朝向等属性会及时改变，所以每个层级的子视截体都会不断变换，子视截体的轴对齐包围盒也要跟着变化，但这样可能导致出现前后两帧轴对齐包围盒发生突变，进而导致生成的阴影贴图的有效分辨率可能在连续的两帧中发生突变，产生阴影抖动问题，解决方案是把包围盒改为包围球，包围球随着子视截体的变化而发生大小的变化程度相对于包围盒来说小很多，如下图：



虽然阴影的投影可以契合包围球，但是球体还会覆盖周围的一些空间，导致我们在剔除区域之外也可以看到一些阴影。光的方向和球无关，所以我们所有的方向光都使用相同的包围球。

1. 我们需要知道这些球体应该从哪个级联中采样，因此需要将包围球数据发送到GPU，在Shadows脚本中定义级联包围球和级联数量的着色器标识ID，并定义一个Vector4类型的数组存储包围球数据，其中XYZ分量存储包围球的位置，W分量存储球体半径。

2. 级联包围球是从ComputeDirectionalShadowMatricesAndCullingPrimitives方法中得到的spitData数据的一部分，我们可以直接调用获取到，然后赋值给包围球数组。因为我们想要所有的光源都使用相同的级联，所以只需要拿到第一个方向光的包围球数据即可。

3. 我们后续需要在着色器中判断物体表面的片元是否在包围球中，可以通过该片元到球心距离的平方和球体半径的平方来比较，我们传递数据之前先计算好球体半径的平方，就不用再在着色器中计算了。

4. 渲染级联后在RenderDirectionalShadows方法中将级联数量和级联包围球数据发送到GPU。

4.3.4 采样级联

1. 在Shadows文件的\_CustomShadows缓冲区中定义级联数量和包围球数据数组。

2. 定义一个结构体ShadowData存储表面阴影数据，在其中定义级联索引的属性，再定义一个GetShadowData方法，返回世界空间的表面阴影数据。最初只把级联索引设为0。

3. 将阴影数据添加到Light文件的GetDirectionalShadowData方法中，将级联索引和光源的阴影图块索引相加得到最终的图块索引。

4. 在GetDirectionalLight方法中也添加shadowData参数，把阴影数据发送到GetDirectionalShadowData方法中。

5. 在Lighting.hlsl的GetLighting方法中获取表面阴影数据并传递到GetDirectionalLight方法中。

6. 要选择正确的级联，需要计算两点间距离的平方，在Common.hlsl中定义这么一个方法。

7. 调整Shadows.hlsl文件中的GetShadowData方法得到合适的级联索引。循环所有级联包围球，如果物体表面到球心的距离的平方小于球体半径的平方，就说明该物体应在这层级联的包围球中，则跳出循环并得到正确的级联索引。

由于存在自阴影的伪影，导致级联之间的弯曲过渡边界也清晰可见。如果想试试看效果，可以在Light.hlsl文件的GetDirectionalLight()方法中用级联索引除以4来代替阴影衰减，这样效果就会比较明显，如下图所示：

4.3.5 剔除阴影采样

1. 如果我们最终超过了最后一个级联的范围，很可能没有有效阴影数据，这种情况下也不需要采样阴影了，没有意义。我们在Shadows.hlsl文件的ShadowData结构体中添加一个字段Strength作为一个标识符，如果超出最后一个级联范围就设为0。

2. 在Light.hlsl文件的GetDirectionalShadowData方法中获取阴影强度时乘上这个Strength参数，这样可以剔除掉最后一个级联范围外的所有阴影。

这样不在级联中的阴影就被剔除了，可以把阴影的最大距离调小一点观察一下。

​有些物体的投影在最后一个级联的包围球内没有。原因就是最外面的包围球超出了我们设置的最大阴影距离。我们需要把最大阴影距离传到GPU，然后进行判断，如果超出最大距离就停止采样阴影来解决它。

3. 首先在Shadows脚本中定义阴影最大距离的着色器标识ID，通过RenderDirectionalShadows方法将最大阴影距离传到GPU。

4. 阴影最大距离基于的是视图空间的深度，而不是与相机的距离，为了进行剔除我们需要知道物体表面的深度，在Surface结构体重定义一个depth字段存储表面深度。

5. 在片元函数中，调用源码库中的TransformWorldToView()方法把世界坐标转换到视图空间，并获取负的z坐标作为表面深度。

6. 在Shadows.hlsl文件的\_CustomShadows缓冲区中定义\_ShadowDistance字段获取阴影最大距离，然后在GetShadowData方法中进行判断，当表面深度比最大阴影距离小时，才进行阴影采样。

4.3.6 阴影过渡

还有一个问题，突然切断阴影最大距离处的阴影会显得很突兀，我们通过一种线性淡化的方式使阴影过渡变得柔和自然一些。阴影淡化应从阴影最大距离之前的一段距离开始，直到最大距离时阴影强度为0。我们使用下面的数学公式：



​公式中d是表面的深度，m是阴影最大距离，f是阴影过渡范围，指的是到达阴影最大距离之前的那一小段距离。公式的最终结果应限制在［0，1］之间，下面进行具体实现。

1. 在ShadowSettings脚本中添加一个字段表示阴影过渡距离，默认为0.1。因为这个过渡距离和阴影最大距离在上面的数学公式中有一个相除的计算，所以该值不能为0，我们限制一下该字段的最小值为0.001。

2. 在Shadows脚本中，定义一个阴影过渡距离着色器标识ID，代替原来的阴影最大距离标识ID。然后把阴影最大距离和阴影过渡距离的倒数传递给GPU，因为在着色器中乘法比除法效率更高。

3. 在Shadows文件的\_CustomShadows缓冲区中将阴影最大距离替换成新的字段。

4. 最后定义一个FadedShadowStrength方法来计算阴影过渡时的强度，通过套用上面的数学公式完成。在GetShadowData方法中调用该方法得到有线性过渡的阴影强度。

4.3.7 级联过渡

我们也可以使用和阴影过渡相同的方法，在最后一个级联边缘对阴影进行平滑过渡，而不是硬切。在ShadowSettings脚本中定义一个级联阴影过渡的字段，并设置一个初值0.1。

和之前的过渡公式的区别是分子使用级联距离的平方除以包围球半径的平方，而不是之前线性的深度除以阴影最大距离，这意味着该阴影过渡是非线性的。



其中r是包围球的半径，我们要保持配置的过渡距离不变，需要将f替换成 1−(1−f)2，然后将该值存储在阴影过渡向量中的Z分量中并取反，一同发送到GPU。

在Shadow.hlsl文件的GetShadowData方法中判断要渲染的对象是否在最后一个级联的范围中，如果是则计算级联的过渡阴影强度，和阴影最大距离的过渡阴影强度相乘得到最终阴影强度。

4.4 阴影质量