LOD和反射

LOD技术

LOD的原理是根据物体的包围盒高度所占当前屏幕的高度的百分比来调用不同精细度的模型。

LOD的缺点是需要准备几种不同精细度的模型，一来需要美术对同一对象制作几个不同层次的精细度模型，会增加美术的工作量；二来模型数量增多会导致游戏运行时占用更多的内存空间，这是一种典型的空间换时间的手段。

Project Settings->Quality->Other中有一个LOD Bias属性，该属性可以缩放阈值。该属性值为2意味着实际上评估物体高度占比时会将LOD百分比阈值翻倍，也就是物体包围盒高度所占当前屏幕高度30%以上的比例时就可以使用LOD 0级别中精细度最高的模型来渲染，而不是60%，所以如果想跟组件中的LOD百分比阈值同步，就将LOD Bias属性设为1即可。

LOD级别过渡

现在LOD的级别在临界阈值之间进行切换时会突然变换过去，这在视觉上会比较突兀，我们需要使得LOD级别切换时能够平滑过渡来改善视觉效果。将Fade Mode切换成Cross Fade模式即可，这使得旧的级别淡出，新的级别淡入。然后每个LOD级别下面会多出一个Fade Transition Width属性，它代表过渡区域占当前LOD级别的比例，其范围在[0，1]之间，如果该值为0表示当前LOD级别和下一个LOD级别切换时没有过渡，该值为1表示进入到当前LOD级别时将立即往下一个LOD级别过渡，当值为0.5时，表示进入到当前LOD级别的50%（中间）处开始往下一个LOD级别过渡，所以该值越小会过渡的越快，可以调节到一个合适的值。

1. 当启用Cross Fade时，其实两个LOD级别的对象会同时渲染出来，然后着色器将以某种方式进行混合，Unity通常使用屏幕空间抖动或着混合来实现Cross Fade。下面我们在Lit.shader的CustomLit和ShadowCaster Pass中添加一个LOD\_FADE\_CROSSFADE关键字的声明指令。

2. 在UnityInput.hlsl的UnityPerDraw缓冲区中我们已经定义过unity\_LODFade向量，它代表对象的过渡程度，其中X分量存储的是过渡因子，Y分量其实也存储了相同的因子，只不过它被量化为16步，我们不使用它。现在做个测试，在LitPass.hlsl的片元函数的开头返回该向量的X分量来观察过渡因子。上图白色是LOD 1级别中的小球，黑色是LOD 2级别中的小球，可以看出淡出的对象过渡因子从1开始然后减到0。

抖动

1. 要混合两个LOD级别，可以使用裁剪，要用类似于近似半透明阴影的方法，由于我们需要对表面和阴影进行裁剪，因此在Common.hlsl中定义一个ClipLOD方法来完成裁剪，它需要一个裁剪空间的顶点坐标的XY分量和过渡因子作为参数，如果LOD\_FADE\_CROSSFADE被定义了，则使用过渡因子减去抖动值来进行裁剪。抖动值可以使用一个简单的计算公式来得到，这里我们在垂直方向进行渐变。

2. 在LitPass.hlsl和ShadowCasterPass.hlsl片元函数的最开始处调用该方法，之前的测试代码可以删掉了。

3. 我们得到了条纹状的渲染结果，但在交叉过渡时只有一个LOD级别中的对象出现，这是因为两个级别的其中一个的过渡因子为负数，要在裁剪时进行判断。如果过渡因子为负，则应和抖动值相加而不是相减来解决这个问题。

4. 最后，我们调用InterleavedGradientNoise方法来获取正常的抖动值。

虽然抖动创建了一个非常平滑的过渡，但像半透明阴影一样，过渡的阴影不是很稳定，我们可以通过勾选Animate Cross-fading来改善这点，这会忽略过渡区域的宽度，而是通过一个LOD的级别阈值时快速地过渡。默认过渡动画会持续半秒，不过可以通过调整LODGroup.crossFadeAnimationDuration字段来修改动画持续时间。

反射

间接BRDF

1. 我们已经支持了基于BRDF漫反射颜色的漫反射全局照明，现在我们开始支持镜面反射全局照明。首先在BRDF.hlsl定义一个IndirectBRDF方法获取基于BRDF的间接照明，它有四个参数，分别是表面信息、BRDF数据、从全局照明中获得的漫反射和镜面反射颜色，最初只返回反射的漫反射光照。

2. 然后通过全局照明中的镜面反射颜色乘以BRDF中的镜面反射颜色得到镜面反射照明，但表面的粗糙度会散射镜面反射，所以最终反射到人眼的镜面反射应该是减弱的。接着我们将镜面反射除以表面的粗糙度的平方加一，这对较低粗糙度的表面影响不大，但是针对高粗糙度的表面可以使得镜面反射强度减半。最后将镜面反射和漫反射照明相加得到最终基于BRDF的间接照明。

3. 在Light.hlsl的GetLighting方法中调用该方法获取最终的间接照明，而不是直接计算漫反射间接照明，其中第四个代表全局照明中的镜面反射颜色的参数，我们先设为白色。

采样环境的Cubemap

1. 镜面反射反映了环境，默认情况下是天空盒，它是一个立方体纹理（Cube Map），我们在GI.hlsl中声明该纹理和对应采样器。

2. 定义一个SampleEnvironment方法对Cube Map进行采样，通过SAMPLE\_TEXTURECUBE\_LOD宏对纹理进行采样，然后返回采样后的RGB颜色数据，它还需要另外两个参数，分别是3D纹理坐标UVW和纹理Mipmap等级。UVW可以通过reflect方法由负的视角方向和法线方向得到反射方向，Mipmap等级我们设置为0作为最高级，也就是对全分辨率的Cube Map进行采样。

3. 在GI结构体中定义一个镜面反射颜色属性，然后在GetGI方法中通过采样Cube Map获得环境的镜面反射。

4. 然后将正确的镜面反射颜色传递给Lighting文件的GetLighting方法对IndirectBRDF方法的调用中。

5. 最后需要在CameraRenderer脚本的DrawVisibleGeometry方法中添加反射探针的标志PerObjectData.ReflectionProbes。

6. 当粗糙表面散射镜面反射时，不仅会降低反射强度，且使得反射不均匀，像是失去了焦点一样。这个可以通过将环境的Cube Map的模糊版本存储在较低的Mipmap级别中来近似实现这个效果，要得到正确的Mipmap级别，需要知道感知粗糙度，我们将这个属性添加到BRDF结构体中，并在GetBRDF方法中给其赋值。

7. 在GI.hlsl中include源码库中的ImageBasedLighting.hlsl文件，调用它里面的PerceptualRoughnessToMipmapLevel方法通过感知粗糙度来计算出正确的Mipmap级别，然后将它用于Cube Map的采样中。

8. 在GetGI方法中调用SampleEnvironment方法时传递BRDF数据。

9.最后在片元函数中传递BRDF数据。

菲涅尔反射

1. 菲涅尔反射是一种光学现象，当光线照到物体上时一部分发生了反射，一部分发生折射或散射，几乎所有的物体都包含菲涅尔效果，尤其是玻璃这种反光的物体 ，实时渲染中经常会使用菲涅尔反射来根据视角的方向控制反射程度。要计算菲涅尔反射需要菲涅尔等式，真实世界该等式比较复杂，这里使用一个Schlick菲涅尔近似等式的变种。理想情况下它会用纯白色代替BRDF的镜面反射颜色，但粗糙度会影响反射，我们通过将表面光滑度和反射率加在一起得到菲涅尔颜色，其最大值为1。我们在BRDF结构体中定义这么一个菲涅尔属性来存储该值。

2. 在IndirectBRDF方法中，通过1减去表面法线和视角方向的点积，再进行4次相乘叠加得到菲涅尔强度。然后通过该值对BRDF的镜面反射颜色和菲涅尔颜色之间进行插值得到最终环境的反射颜色。

3. 菲涅尔反射主要沿几何体边缘添加反射，当环境立方体纹理不匹配对象后面的颜色时，反射会显得怪异且分散，结构内球体边缘的明亮反射就是一个很好的例子。降低光滑度可以减弱或消除菲涅尔反射，但也会使得整个表面变暗，所以我们在Shader的属性中创建一个菲涅尔滑块属性来控制反射强度。

4. 在LitInput.hlsl的UnityPerMaterial缓冲区中添加该属性，并创建一个GetFresnel方法得到该值。

5. 在Surface.hlsl的结构体中添加菲涅尔反射强度的属性。

6. 在LitPass.hlsl的片元函数给该属性赋值，最后在IndirectBRDF方法中计算菲涅尔强度时乘以该滑块值。

反射探针

反射探针（Reflection Probe）会动态地产生周围环境的贴图，来产生环境映射的效果。用GameObject->Light->Reflection Probe创建反射探针，它通过组件上的Importance和Box Size属性控制每个探针影响哪块区域。

反射探针通过渲染立方体贴图（Cube Map）获取周围的环境，意味着它会渲染6次场景，立方体贴图每个面渲染一次。默认情况下该类型设置为Baked烘焙模式，它会在编辑阶段生成一个存储了探针周围环境景象的立方体纹理，然后对场景中标记为Reflection Probe Static的游戏对象进行取景烘焙。烘焙完成后，立方体纹理不会发生变换，所以不会受物体位置实时变化的影响。当然你也可以设置为Realtime，它可以在运行时生成并更新立方体纹理，此时对场景中游戏对象进行取景生成时就不限于静态的了，但这种类型的探针非常耗费性能，刷新探针以及更新贴图内容也需要耗费较长时间，所以不建议选择这个。

每个对象的MeshRenderer组件上都有一些用于调节反射探针的选项，当一个游戏对象横跨了多个反射探针时，且Reflection Probes属性选项不为Off时，Mesh Renderer组件会自动把游戏对象所触及的所有反射探针添加到一个它维护的内容数组中。

其中Anchor Override属性可以微调使用的反射探针，而不用担心矩形框的大小和位置。Reflection Probes属性默认为Blend Probes，它会启用反射探针，且游戏对象所占空间如果和多个反射探针的作用区域重叠，则混合最好的两个反射探针。如果游戏对象附近没有反射探针，渲染器将使用天空盒作为默认反射，但默认反射和反射探针之间不会混合，且此模式和SRP Batcher不兼容，同时Unity其它RP不支持它，当然我们也不会支持。还有两种常用的模式，一个是Off，表示不使用反射探针；一个是Simple，表示只使用内部数组中Weight值最大的反射探针。

解码探针

1. 现在我们要解码立方体纹理中的数据，它可以是HDR或LDR，其强度也可以调整，这是通过在UnityInput.hlsl的UnityPerDraw缓冲区中声明unity\_SpecCube0\_HDR属性来提供这些设置的。

2. 最后在GI.hlsl的SampleEnvironment方法中的末尾通过原始环境数据和环境设置作为参数，调用DecodeHDREnvironment方法来得到正确的环境反射颜色。