**使用深度和法线纹理**

**深度纹理**

**What:**

**Why:**

**How:**

**Unity是如何得到深度纹理的？**

在Unity中，深度纹理可以直接来自于真正的深度缓存，也可以是由一个单独的Pass渲染而得，这取决于使用的渲染路径和硬件。通常来讲，当使用延迟渲染路径时，深度纹理理所当然可以访问到，因为延迟渲染会把这些信息渲染到G-buffer中。而当无法直接获取深度缓存时，深度和法线纹理是通过一个单独的Pass渲染而得到的。具体实现是，Unity会使用着色器替换技术选择那些渲染类型为Opaque的物体，判断它们使用的渲染队列是否小于等于2500(内置的Background，Geometry和AlphaTest渲染队列均在此范围)，如果满足条件，就把它渲染到深度和法线纹理中。

**深度纹理和法线纹理的选择**

在Unity中，我们可以选择让一个摄像机生成一张深度纹理或是一张深度+法线纹理。

**选择生成一张深度纹理时，Unity内部怎么处理？**

Unity会直接获取深度缓存或是使用着色器替换技术，选取需要的不透明物体，并使用它投射阴影时使用的Pass(即LightMode被设置为ShadowCaster的Pass)来得到深度纹理。如果Shader中不包含这样一个Pass，那么物体就不会出现在深度纹理中。

**如果选择生成一张深度+法线纹理，Unity会怎么处理？**

Unity会创建一张和屏幕分辨率相同，精度为32位的纹理，其中观察空间下的法线信息会被编码进纹理的R和G通道，而深度信息被编码进B和A通道。法线信息的获取在延迟渲染中非常容易，只需合并深度和法线缓存即可。而在前向渲染中，默认情况下是不会创建法线缓存的，因此Unity底层使用了一个单独的Pass把整个场景在此渲染一遍来完成。这个Pass被包含在Unity内置的一个UnityShader中。

builtin\_shaders-xxx/DefaultResources/Camera-DepthNormalTexture.shader。

**如何获取深度和法线纹理？**

获取深度纹理：

camera.depthTextureMode = DepthTextureMode.Depth

在Shader中通过声明\_CameraDepthTexture变量来访问它。

获取深度+法线纹理：

camera.depthTextureMode = DepthTextureMode.DepthNormals

在Shader中通过声明\_CameraDepthNormalsTexture变量来访问它。

我们还可以组合这些模式，让一个摄像机同时产生一张深度和深度+法线纹理。

**Unity内置的宏**

深度纹理采样宏：

float d = SAMPLE\_DEPTH\_TEXTURE(\_CameraDepthTexture, i.uv)

SAMPLE\_DEPTH\_TEXTURE\_PROJ宏

SAMPLE\_DEPTH\_TEXTURE\_PROJ宏同样接受两个参数，深度纹理和一个float3或float4类型的纹理坐标，纹理坐标的前两个分量首先会除以最后一个分量，在进行纹理采样。如果提供了第四个分量，还会进行一次比较，通常用于阴影的实现。

SAMPLE\_DEPTH\_TEXTURE\_PROJ的第二个参数通常有顶点着色器输出插值而得的屏幕坐标：

float d = SAMPLE\_DEPTH\_TEXTURE\_PROJ(\_CameraDepthTexture,

UNITY\_PROJ\_COORD(i.scrPos))

其中，i.scrPos是在顶点着色器中通过调用ComputeScreenPos(o.pos)得到的屏幕坐标。

**如何获取深度值？**

这些深度值往往是非线性的，这种非线性来自于透视投影使用的裁剪矩阵。然而，在我们的计算过程中通常是需要线性的深度值，也就是说，我们需要把投影后的深度值变换到线性空间下，例如视角空间下的深度值。

Unity提供了两个辅助函数LinearEyeDepth和Linear01Depth。

LinearEyeDepth负责把深度纹理的采样结果转换到视角空间下的深度值，而Linear01Depth则会返回一个范围在[0,1]的线性深度值。

**如何获取深度+法线纹理？**

inline void DecodeDepthNormal(float4 enc, out float depth, out float3 normal){

depth = DecodeFloatRG(enc.zw);

normal = DecodeViewNormalStereo(enc);

}

获取的是范围为[0,1]的线性深度值和视角空间下的法线方向。

**实例：查看深度和法线纹理**

**实例：再谈运动模糊**

使用速度映射图。速度映射图中存储了每个像素的速度，然后使用这个速度来决定模糊的方向和大小。速度缓冲的生成有多种方法，一种方法是把场景中所有物体的速度渲染到一张纹理中。但这种方法的缺点在于需要修改场景中所有物体的Shader代码，使其添加计算速度的代码并输出到一个渲染纹理中。

我们使用Gems3介绍的一种方法，利用深度纹理在片元着色器中为每个像素计算其在世界空间下的位置。得到世界空间中的顶点坐标后，我们使用前一帧的视角\*投影矩阵对其进行变换，得到该位置在前一帧中的NDC坐标。然后，我们计算前一帧和当前帧的位置差，生成该像素的速度。

本节实现的运动模糊适用于场景静止，摄像机快速运动的情况，这是因为我们在计算时只考虑了摄像机的运动。读者可以在Unity自带的ImageEffect包中找到更多的运动模糊的实现方法。

**实例：全局雾效**

基于屏幕后处理的全局雾效的关键是，根据深度纹理来重建每个像素在世界空间下的位置。首先对图像空间下的视椎体射线(从摄像机出发，指向图像上的某点的射线)进行插值，这条射线存储了该像素在世界空间下到摄像机的方向信息。然后，我们把该射线和线性化后的视角空间下的深度值相乘，再加上摄像机的世界位置，就可以得到该像素在世界空间下的位置。

**实例：边缘检测**