**一．纹理与材质的基本概念**

**1.在虚幻引擎的PBR流程中，纹理大概可分为五种：**

1.Albedo:基本色贴图，记录了材质的基本颜色。

2.Normal：记录了材质表面的法线信息。

3.AO/Cavity：环境光遮蔽贴图。记录了表面模拟的环境光阴影。

4.Roughness：粗糙度贴图，记录的材质表面的粗糙度值。

5.Metal/Spec：金属度贴图。

**2.命名规范**：

大多数情况下，贴图使用T\_作为纹理前缀。

材质球为M\_

**3.纹理尺寸**

支持的标准分辨率，满足2的n次幂的数字：

1\*1， 2\*2， 4\*4， 16\*16， 32\*32， 1024\*1024， 2048\*2048， 4096\*4096

编辑器默认情况下最高支持到8192\*8192

**4.纹理比例**

可以是非正方形，但需要满足是2的n次幂的数字：

1028\*2048,2048\*4096

**5.Mipmap多级渐进式纹理：**

为纹理生成多张缩小2倍的纹理，优化性能与远距离视觉表现。距离相机较远时，使用较低的层级

**6.纹理文件格式以及纹理压缩**

文件的格式与文件的大小没有关系，所以我们要尽可能使用没有压缩率的文件或者没有被压缩过的文件。比如多使用PNG，PSD格式的文件，少用JPG格式的文件。

在设置图片压缩模式时，尽量选择默认的压缩模式。

1.Default默认：BC就是按照像素块压缩。普通贴图导入的默认选项

2.Normalmap法线贴图：导入法线贴图的正确设置，否则无法正常使用。

3.Maske遮罩：使用灰度图时的设置（粗糙度，金属度，不是给人看的贴图）

5.Grayscale灰度：贴图上只有灰度信息的贴图。此压缩方式压缩率非常小。

适合高质量灰度图使用。

6.DIsplacementmap高质量置换贴图：低压缩率，适合置换贴图。

7.vectorD isplacementmap更高质量的向量置换贴图。

8.Alpha：透明度压缩设置

9.BC7最高质量：最高质量同时也是最大的压缩方式

**7.纹理通道与sRPG，Alpha通道。**

1.图像由RGB三个通道混合而成，它们的亮度记录了该通道的颜色多少。除此之外，图像还包括一个A通道，记录图像的透明度。但在引擎中，他并不一定指的是透明度，你可以指定任何纹理。

2.SRGB与伽马矫正

由于人眼对亮度的变化感知并不是线性的，而摄像机或是图像处理程序是线性感光、工作流，所以它们最终会对图像进行一次Gamma矫正，以还原人眼效果。

因为矫正会把图像暗处变得更亮，灰度图同样会变得更亮。如果是粗糙度，那么就会出现比原本更粗糙的效果，如果是金属度，则会获得更多的金属质感。

在UE4中，默认勾选了SRGB选项，它是用来反Gamma矫正 (A2.2)，回到图片最初的样子。但是我们的灰度图并没有被Gamma矫正过，执行反向矫正就会使得整体更黑，那么就会得到错误的灰度图信息。

所以我们如果要使用的是图片的灰度信息，那么我们应该取消掉“SRGB”选项，或者将压缩模式设置为“Mask”，来得到最初的图片灰度信息。

总结就是：给人看的贴图需要勾选sRPG，所有给电脑看的贴图都不需要sRPG.

3. Alpha通道会占用非常多的储存空间，所以没有Alpha通道的纹理应该勾选：无透明度压缩选项。

**8.使用PS合并RGB信息通道图片。**

分离通道后合并图层，再合并通道即可。

**9.光栅化，前向渲染以及延迟渲染**

**1.什么是光栅化？**

我们的显示器是由像素组成的，也就是说所有的3D模型、材质光照等等数据，最终都会被绘制为像素来在显示器上显示，把这些东西绘制在屏幕像素上的过程就叫做光栅化。

**2.光栅化的过程**

1.读取三个顶点，连线为三角形;

2.判断像素点是否在三角形内，如果在，那么就用像素着色器计算出来的颜色对它进行上色。

3.重复上面的步骤，直到绘制完成所有三角形。

**3.前向渲染Forward Rendering**

前向渲染会对场景里每一个几何体进行光栅化，就需要对每一个几何体计算场景所有的灯光来得到它们正确的光影材质表现。

计算完成后，现在我们知道了这些几何体应该如何被绘制在屏幕上，我们当然不希望看到被遮挡的物体被绘制出来，所以接下来对它进行深度测试，把被遮挡的多边形进行剔除，这样就是正确的画面效果了。也就是说实际上被剔除的这些几何体和当前不可见的灯光的计算是完全被浪费掉的，同时因为每个几何体都需要计算所有灯光，那么计算量将非常大，即使是GPU也很难负担，也就意味着场景中的灯光数量将会被严格限制。

较新的前向渲染实际上已经可以进行灯光剔除等操作来降低光照的计算量，相对来说也能同时使用更多的光源。

**4.延迟渲染 Deferred Rendering**

因为遮挡的灯光和几何体是不可见的，那么我们如果能够直接不去计算它，也就可以极大的减少无用的性能消耗，所以延迟渲染就诞生了。

延迟渲染增加了一个G-Buffer缓存，G是Geometry 几何体的意思，几何体的深度检测会先被执行，再把各种数据光栅化到这个缓存区里，在这里它不会计算光照。

G-Buffer生成后，开始计算光照，实际上接下来的计算基本上全是G缓存中的图像合成过程那么由此可见一个灯光的计算实际上只会对它覆盖的像素计算一次，那么灯光渲染的成本就大大降低了。而且G-Buffer里包含的各通道信息同样可以在后期处理里中使用，也就意味着延迟渲染的后期能够表现的效果就非常的多了。

延迟渲染和前向渲染需要根据项目需求来决定，并没有孰强孰弱之分。

**10.遮蔽剔除和半透明材质**

**1.计算遮蔽关系（按照机能消耗以及计算先后顺序排列）**

**1.距离剔除：**相机的渲染距离往往不是无限远的，在渲染时，超出距离的模型会被剔除掉。这种剔除方式消耗最低，因此第一位计算。

**2.视锥剔除：**相机的渲染是类似金字塔的锥形范围，超出锥形范围的模型会被剔除掉。超出锥形范围的模型将会被直接剔除掉，不进行渲染。

**3.遮蔽剔除：**检测每一个模型的可见性，相对高消耗的剔除检测，所以它通常会在最后执行。

因为在这之前的剔除操作已经剔除了很大一部分模型，检测模型可见

性的工作量就会降低。

以上的剔除都是引擎自动完成的，距离剔除需要手动设置，在UE4中

还能烘培预计算可视性数据，这些不在本套教学的讨论范围之内。

**2深度检测**

以上的步骤只能剔除完全不可见或者完全被遮挡的物体，那么对于半个被遮挡的物体，可以通过深度检测来剔除几何体上一部分像素，让它不会出现遮挡处的透视效果。

**3.半透明材质**

G-Buffer缓存会先进行几何体深度检测，再进行光照计算完成光栅化，那么也就是说实际上半透明表现是有问题的，因为被遮挡的物体已经被深度检测剔除掉了，而半透明需要知道背后的像素是什么，所以半透明需要单独对待，不能进行深度检测。这也就是为什么在延迟渲染中半透明实现起来非常困难的原因。

UE4中可以单独为半透明材质开启前向渲染功能，这是一个消耗非常高的材质类型，无法完成深度检测，所以也会出现半透明物体的渲染排序问题。

**4.虚幻引擎在渲染半透明时的排序问题**

两个半透明材质的模型排列在一起进行渲染时，引擎所采用的是两个模型的原点作为判断谁在谁后面的依据进行渲染。因此会产生随机排序的错误。

虚幻引擎可以强制规定半透明材质物体的渲染排序。在【细节面板-渲染-高级-半透明排序优先级（数字越大，优先级越高）】

**二.虚幻引擎可视化高级着色器语言.**

**1.向量和颜色**

**1.向量的数学意义**

它实际上就是一些数字的组合，也就是数组。根据包含的数据不同，在数学中可以这样记录:它们分别是一维向量，二维向量，三维向量。

其中一维向量也可称为标量。

**2.向量的几何意义**

在几何里，向量是有大小，有方向的一个有向线段。

向量的大小也就是它的长度，叫做模，向量没有负数的长度。向量的方向则是向量在空间中的指向。

要注意的一点，向量是没有位置的，它记录的是在对应轴上平行的位移。

零向量是唯一一个没有方向的向量，但将他理解为点是不准确的。我们应该将他们理解为“没有位移”

向量的负向量实际上就是一个和原大小相等，方向相反的向量

**3.向量在3D里的运用**

它在3D里的运用非常广泛，举几个简单的例子，比如判断玩家的朝向，判断相机是否对着某个物体，判断玩家距离某个地方有多远，这些都是向量的运算。

**2.向量的数学运算**

标量的本质是三个通道相等的向量。

计算二维向量的模长公式：根号下x的平方加y的平方。

**3.Time**

Time节点输出的是虚幻引擎从启动至现在的秒数。单位是秒（s）。

**4.Debug**

Debug是调试节点。用于制作材质时的调试步骤。Debug共有四种不同维度向量的调试节点。适用于不同调试情景下的不同功能。

2.作用：Vector（输入接口）：连接需要调试的节点，输出数学结果。

**5.材质与材质实例**

节点的本质是一段封装好的High Level Shader Language（高级着色器语言）

代码例句float3 ExampleVariableTexture = Texture2DSample(Tex, TexSampler, UV);

float3 ExampleVariableColor = float3(1,0,0);

float3 output;

output= ExampleVariableTexture\*ExampleVariableColor;

return output;

//纹理采样器和(1,0,0) 相乘

**6.材质静态开关Switch（StaticSwitchParameter）**

材质里某些功能的开启和关闭开关。使用开关会生成多套HLSL代码，然后进行切换被关闭的节点在运行时没有任何性能消耗。

因为每一个开关都会编译出多套着色器代码，也就意味着滥用静态开关就会导致着色器大大增加，所以需要合理使用它。

**功能：**在编写材质节点时，使用此节点后，可以利用材质实例进行多种不同效果之间来回切换，但是因为是静态开关，因此无法在游戏运行中切换。

**用法：**输入端为布尔值（True，False），每个布尔值可以输入一个不同的节点。输出端输出其中一个布尔值输入的结果。在材质此节点的细节面板里可以设置其默认值。引擎默认输出False，默认值勾选对号则输出True。

这个节点为参数类型，因此可以在材质实例中进行设置。

**三，UV，纹理采样**

**1.纹理采样器 TextureSample**

在虚幻材质里，所有的纹理只有经过采样之后才能够进行使用。纹理贴图在导入材质编辑器的过程中默认应用纹理采样器，想要取消纹理采样器的应用则需要右键转换为纹理对象。

**用法：**输入端分别为UVs，Tex，Apply View MipBias。其中UVs输入纹理坐标，Tex输入纹理贴图。

**2.纹理坐标TexCoord**

纹理坐标节点决定了如何对纹理进行采样，它默认输出XY轴0~1，那么就可以完整采样整个纹理。UE里坐标轴原点位于贴图的左上角。对纹理采样的计算几乎全都是基于该节点，但你也可以通过一些方法自制纹理采样来达到特殊的效果。

**功能：**输出一个二维向量。对应xy轴的0-1.

**用法:**此节点决定了纹理采样的范围。在细节面板里可以调整纹理采样的范围。

**利用纹理坐标实现扭曲效果：**

纹理坐标与糟粕贴图相加可以获得被打乱的纹理坐标的效果，得到扭曲的效果

**3.Mask**

分离向量通道

**4.Append**

合并向量

**5. Panner**

平移节点，输入端为Coordinate，输入纹理坐标

Time为平移单位，输入常量

Speed为速度

**6.** **ScaleUVsByCenter**

沿中心点缩放

**7.Rotator**

旋转节点，Time输入旋转单位