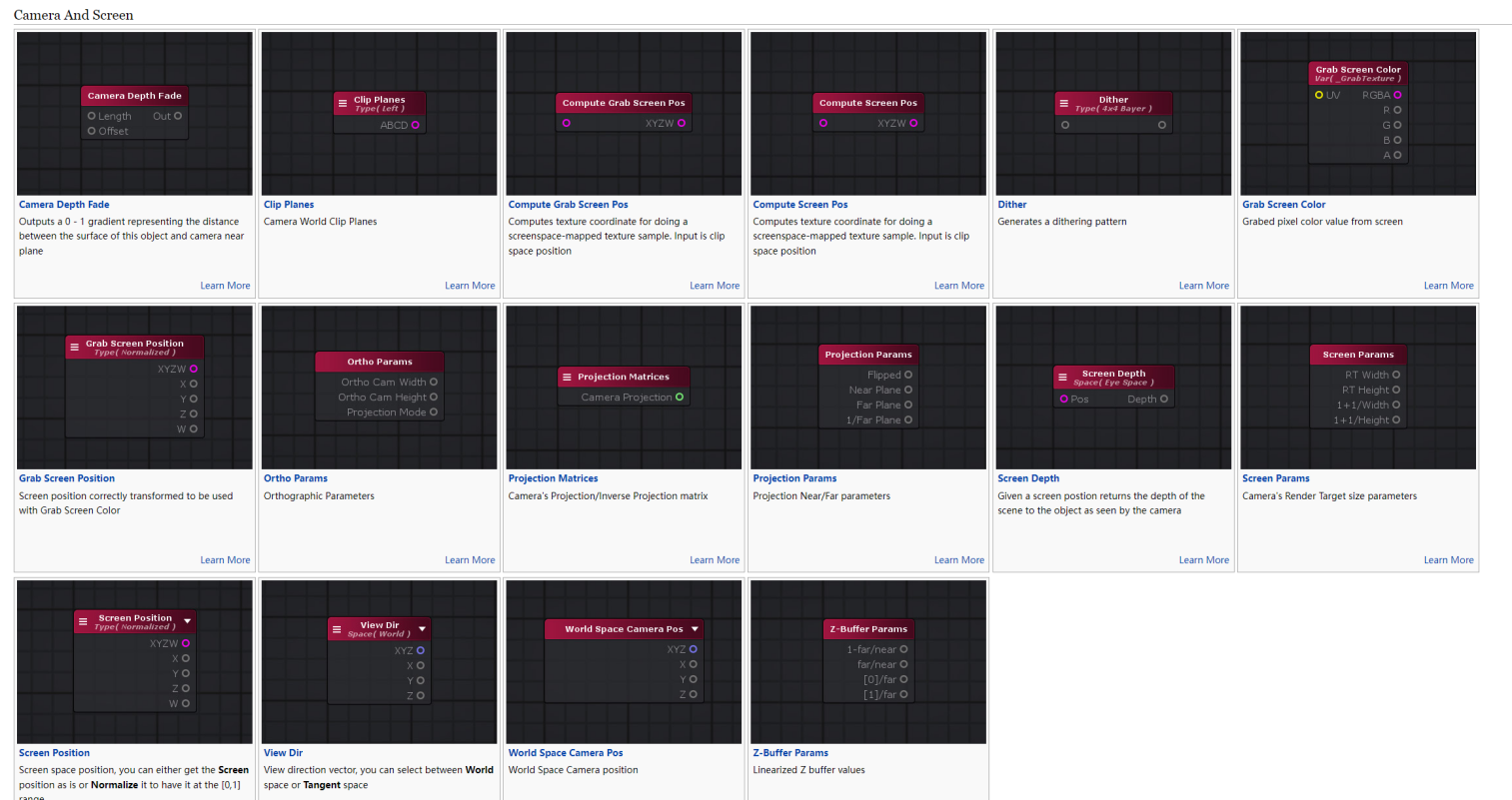
# ASE NODE

## 1 Camera And Screen

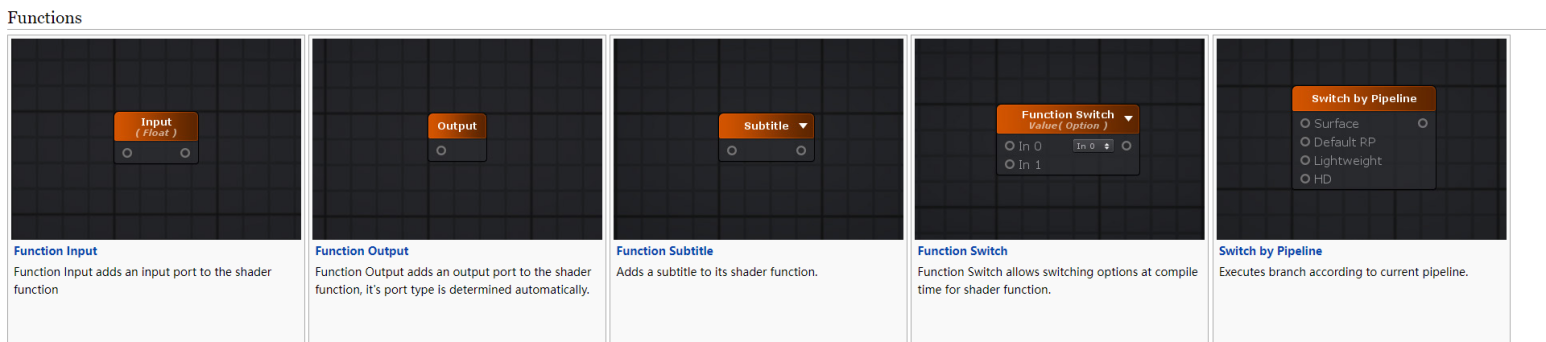


|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 中文 | 英文 | 概念 | 掌握程度 |
|  | 相机深度淡入淡出节点 | Camera Depth Fade | 用来隐藏摄像机面向的半透明对象与不透明对象相交时出现的不美观接缝 | IMG_256 |
|  | 裁切面 | Clip Plane | Clip Plane 节点输出由其[Plane](http://wiki.amplify.pt/index.php?title=Unity_Products:Amplify_Shader_Editor/Clip_Planes" \l "paramPlane)选项选择的相机视锥体平面世界空间方程。 | 用两个面对球体裁切 |
|  |  | Compute Grab Screen Pos | 计算抓取屏幕位置节点将剪辑空间中的位置转换为标准化屏幕空间纹理坐标 |  |
|  |  | Compute Screen Pos | 计算屏幕位置节点将剪辑空间中的位置转换为**屏幕空间纹理坐标**。然后，这些坐标可以直接在[纹理样本](http://wiki.amplify.pt/index.php?title=Unity_Products:Amplify_Shader_Editor/Texture_Sample" \o "Unity 产品：Amplify 着色器编辑器/纹理示例)上用作其 UV 坐标，用于执行屏幕空间映射纹理样本。 |  |
|  | 抖动节点 | Dither | Dither 节点根据其[Pattern](http://wiki.amplify.pt/index.php?title=Unity_Products:Amplify_Shader_Editor/Dither" \l "paramPattern)属性创建屏幕空间抖动图案。 |  |
|  | **[抓取屏幕颜色](http://wiki.amplify.pt/index.php?title=Unity_Products:Amplify_Shader_Editor/Grab_Screen_Color" \o "Unity 产品：Amplify Shader Editor/抓取屏幕颜色)** | Grab Screen Color | 从屏幕抓取像素颜色值 |  |
|  | 抓取屏幕位置 | Grab Screen Position | 屏幕位置已正确转换以与抓取屏幕颜色一起使用 |  |
|  | 正交参数 | Ortho Params | 正交相机的参数，宽高等 |  |
|  | 投影矩阵 | Projection Matrices | 相机的投影，逆投影矩阵 |  |
|  | 投影参数 | Projection Params | 投影近/远参数 |  |
|  | 屏幕深度 | Screen Depth | 给定屏幕位置，返回相机所看到的对象的场景深度 |  |
|  | 屏幕参数 | Screen Params | 相机的渲染目标尺寸参数 |  |
|  | 屏幕位置 | Screen Position | 屏幕空间位置，您可以按原样获取**屏幕位置，也可以对其进行标准化**以使其处于 [0,1] 范围内 |  |
|  | 视图方向 | View Dir | 视图方向向量，可以在**世界**空间或**切线**空间之间选择 |  |
|  | **[世界空间相机定位](http://wiki.amplify.pt/index.php?title=Unity_Products:Amplify_Shader_Editor/World_Space_Camera_Pos" \o "Unity 产品：Amplify Shader Editor/World Space Camera Pos)** | World Space Camera Pos | 世界空间相机位置 |  |
|  | **[Z 缓冲区参数](http://wiki.amplify.pt/index.php?title=Unity_Products:Amplify_Shader_Editor/Z-Buffer_Params" \o "Unity 产品：Amplify Shader Editor/Z-Buffer Params)** | Z-Buffer Params | 线性化 Z 缓冲区值 |  |

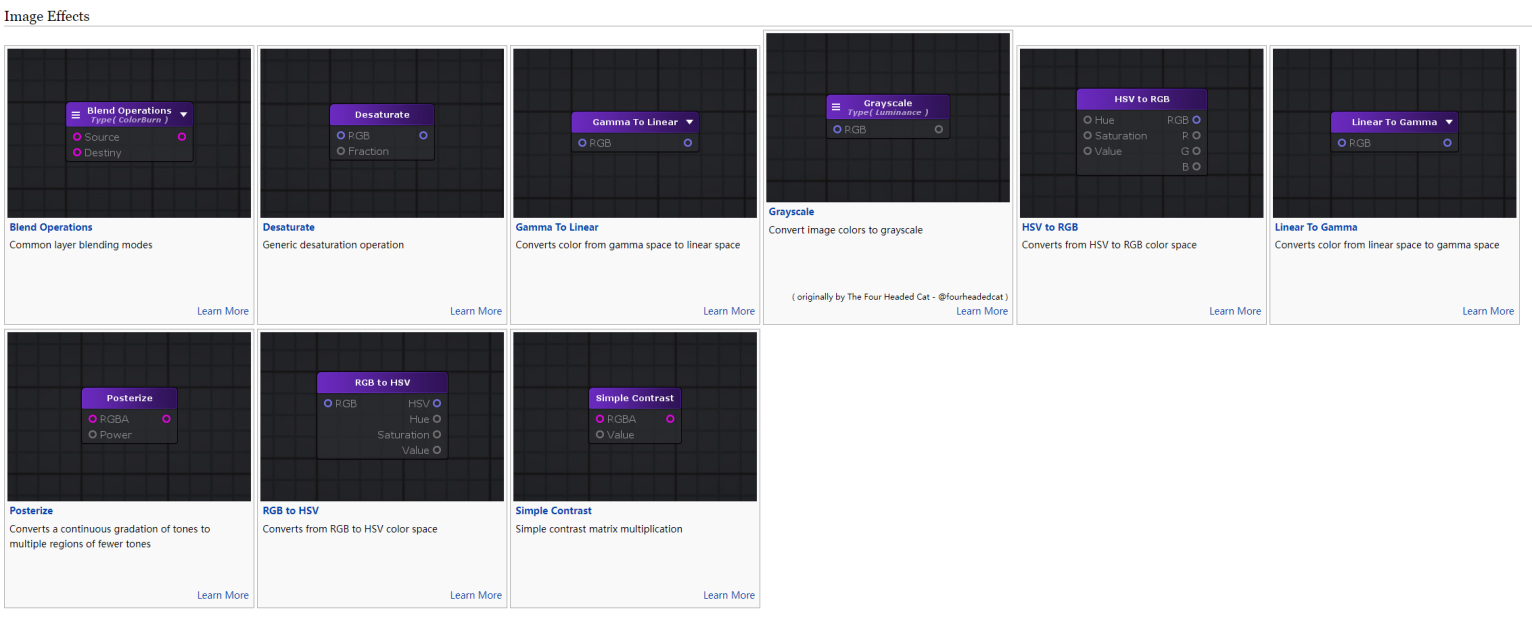
## 2 Constants And Properties



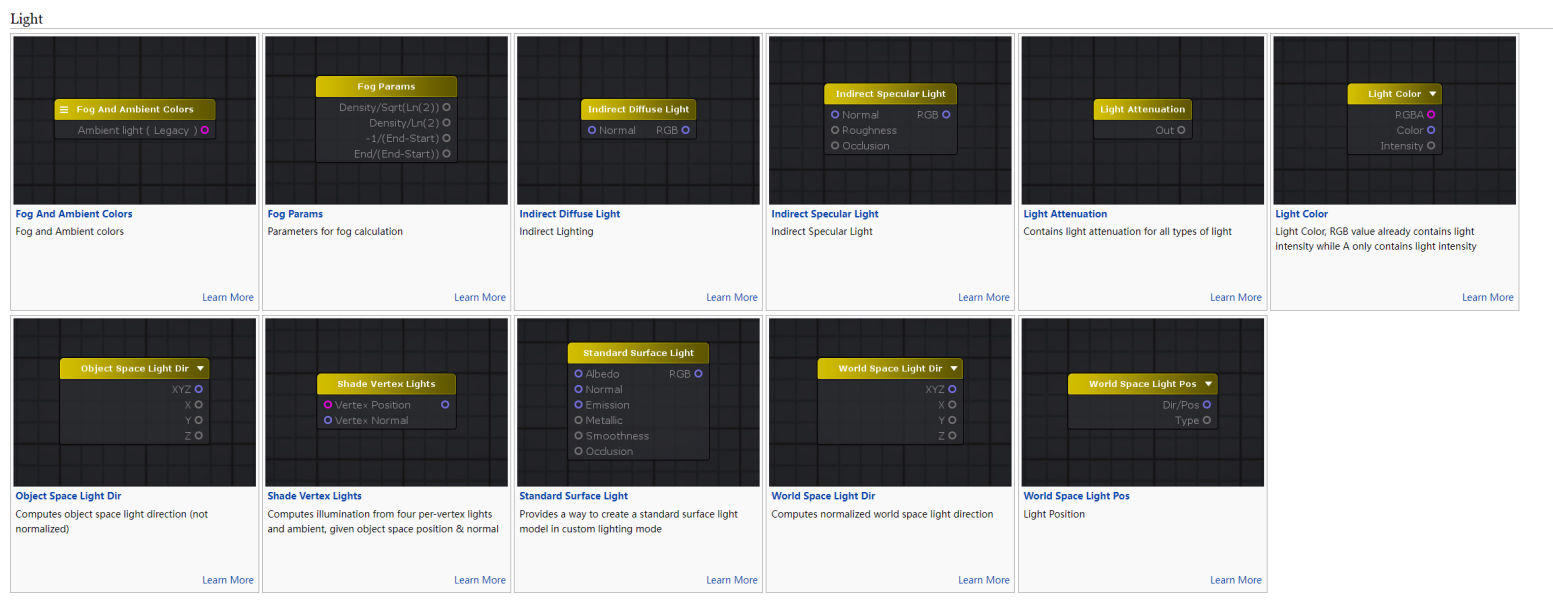
## 3 Functions



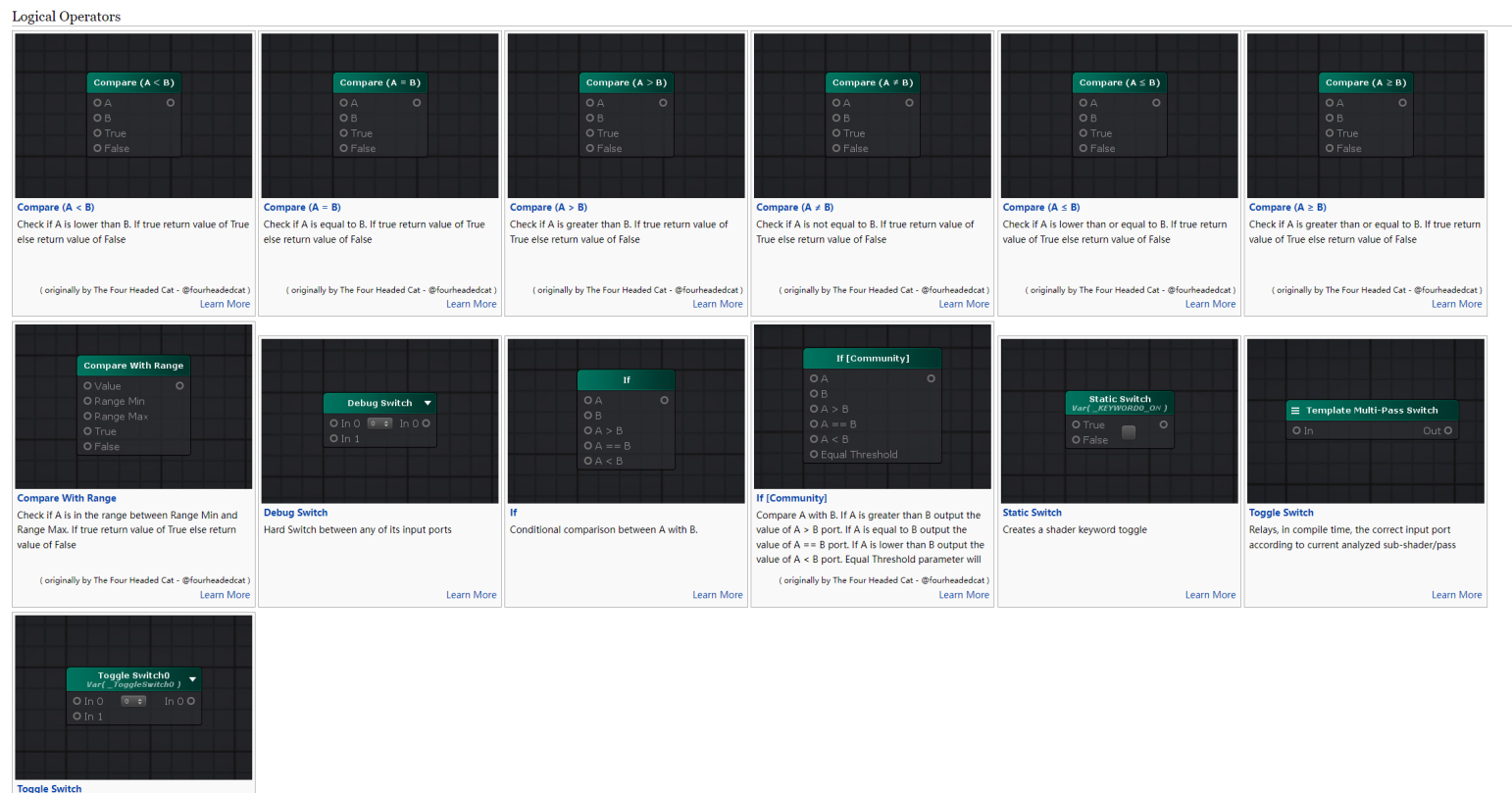
## 4 Image Effects



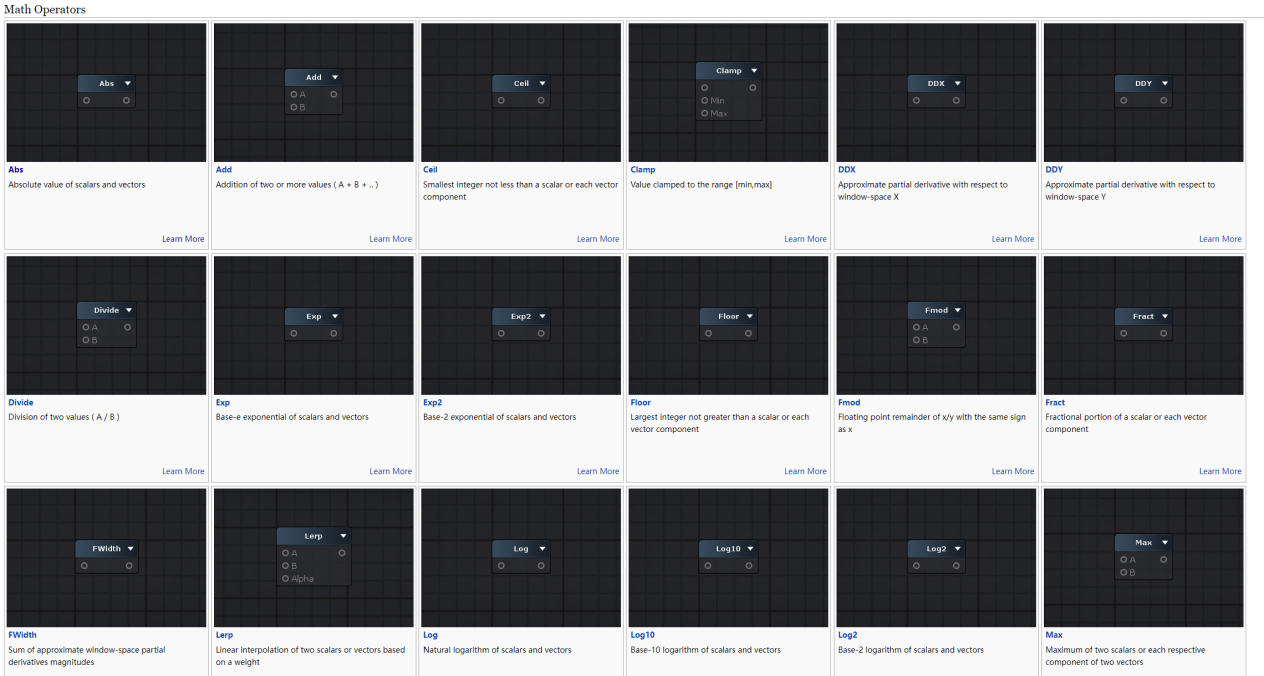
## 5 Light



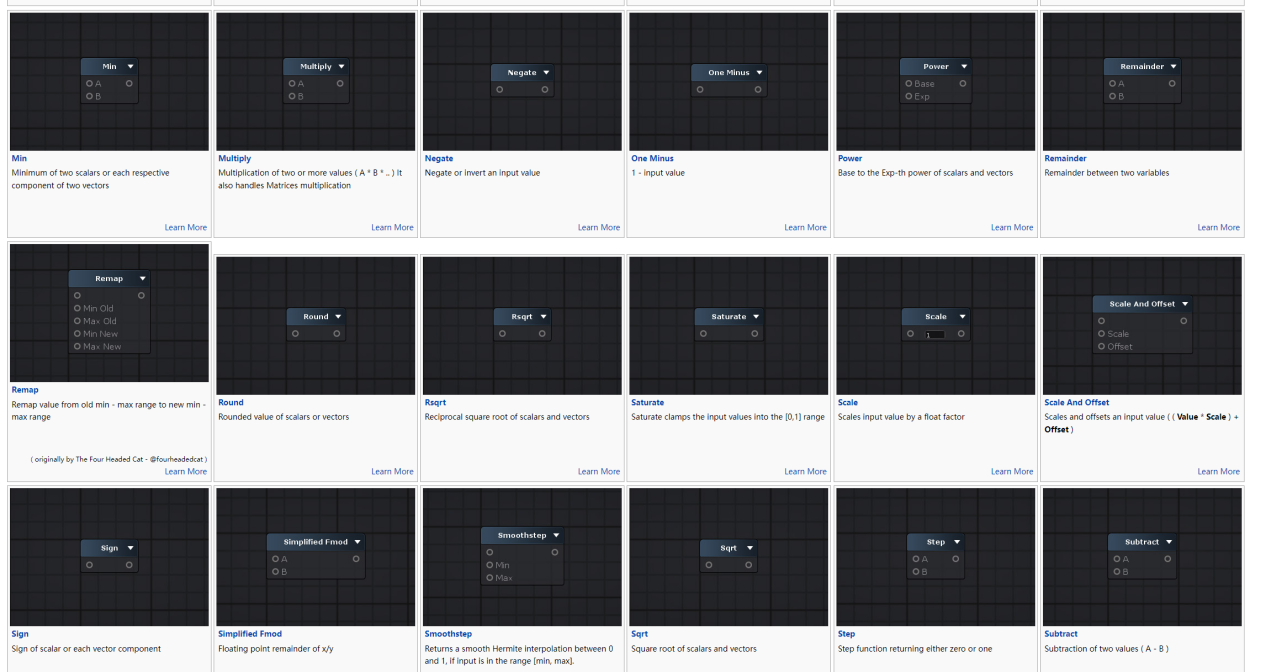
## 6 Logical Operators

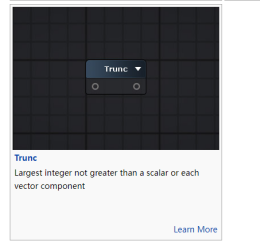


## 7 Math Operators



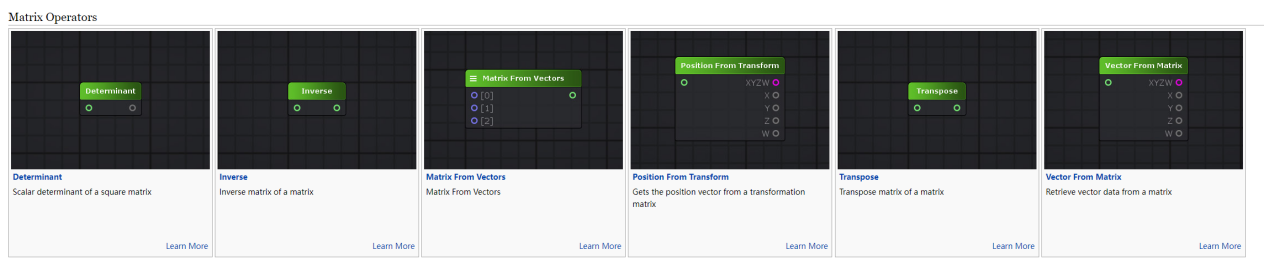
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 中文 | 英文 | 概念 | 作用 | 掌握程度 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  | 插值 | Lerp | 使用表达式(( 1 - A ) \* I + B \* I )在 A 和 B 之间生成一个新的混合值。插值器值的范围在 [0 - 1] 之间，如果I = 0则返回A，如果I = 1则返回B。 | 1 边缘光  2 混合贴图(比如草地和岩石) |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |



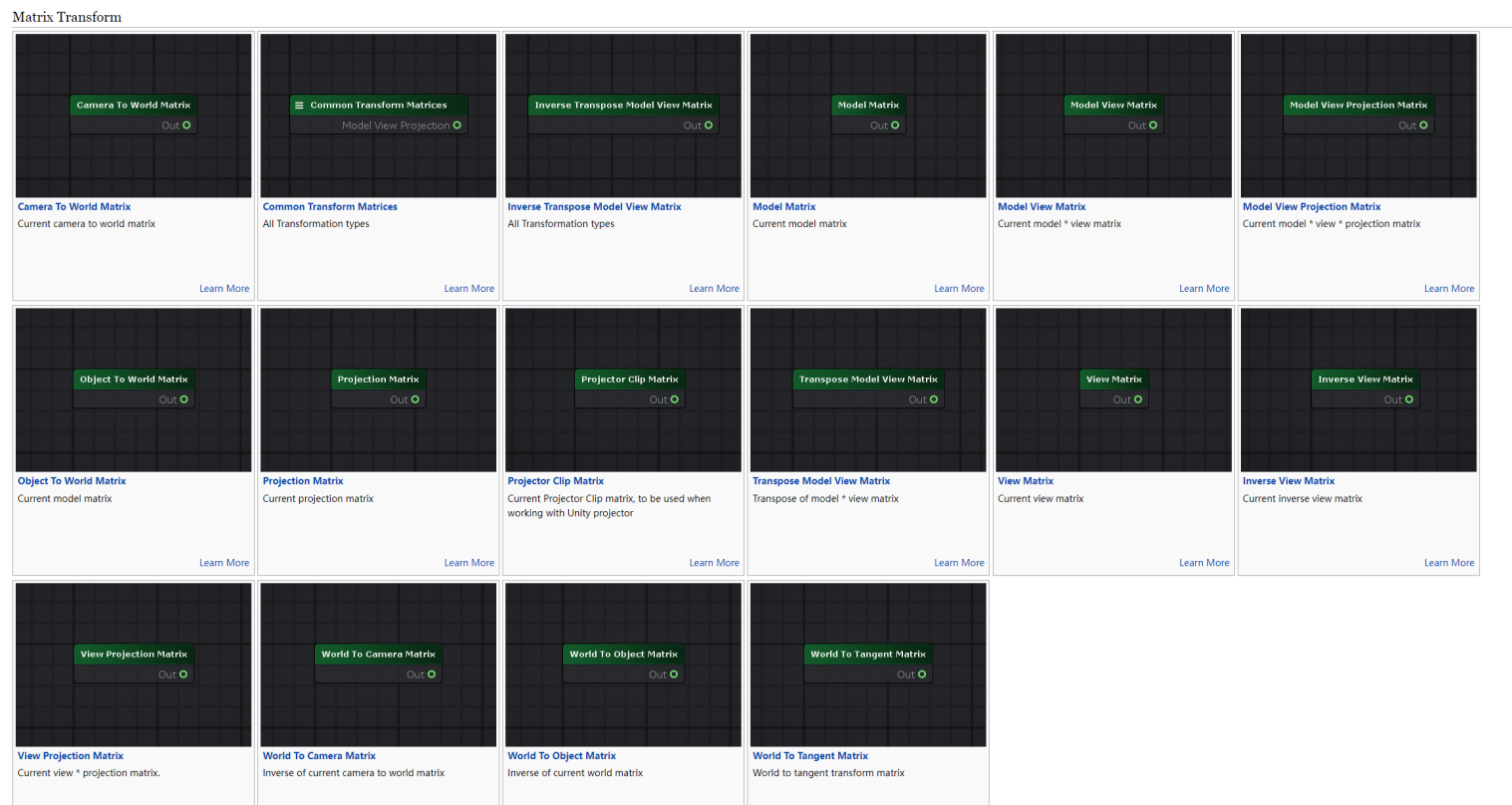


|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 中文 | 英文 | 概念 | 作用 | 掌握程度 |
|  |  |  |  |  |  |
|  | 乘法 | Multiply | 可以使几个输入按照通道进行相乘 | 1 纹理叠加 |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  | 步骤 | Step | 对比后输出值 | 0: This value is returned if B < A  1: This value is returned if B >= A |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

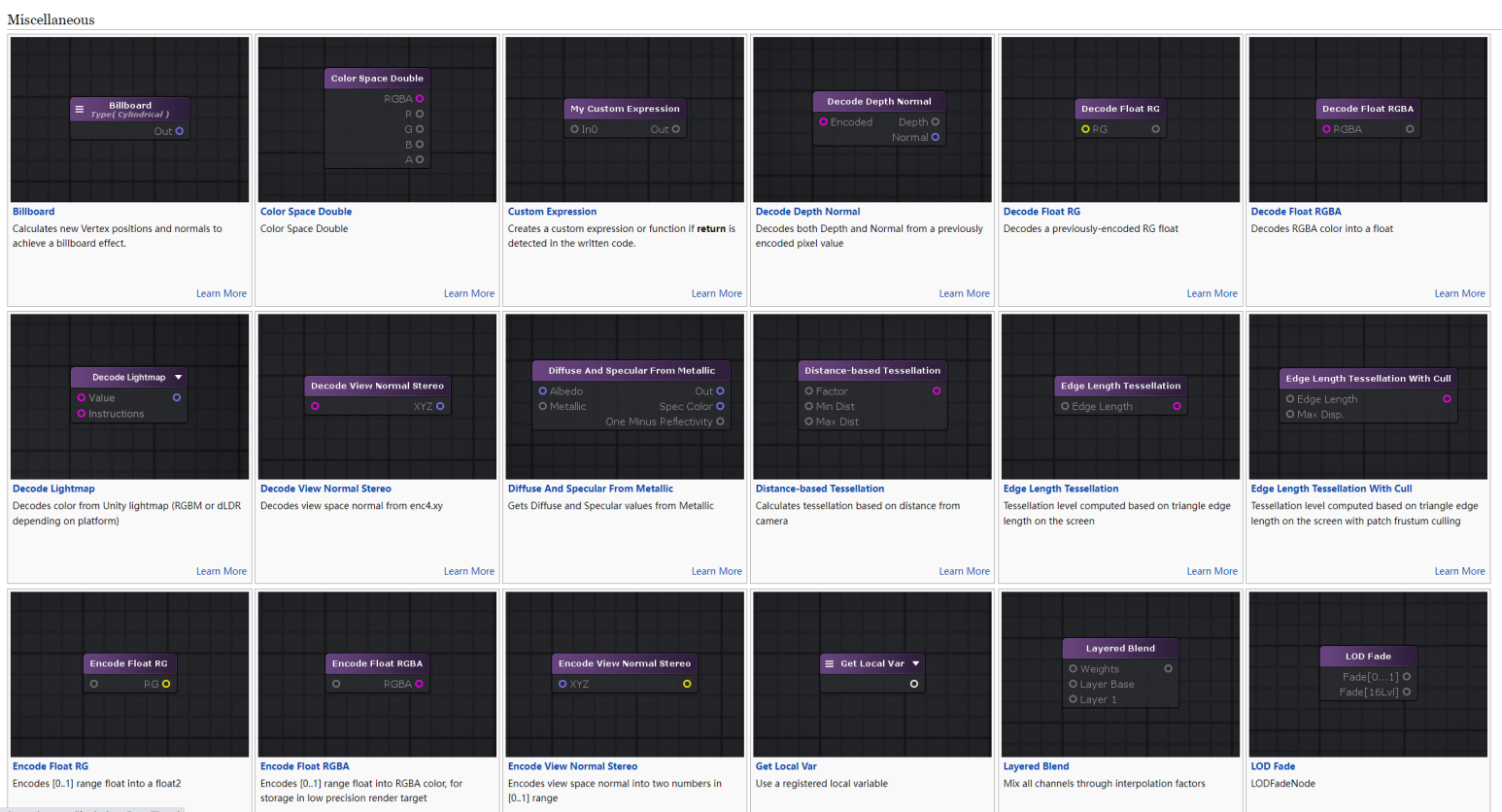
## 8 Matrix Operators

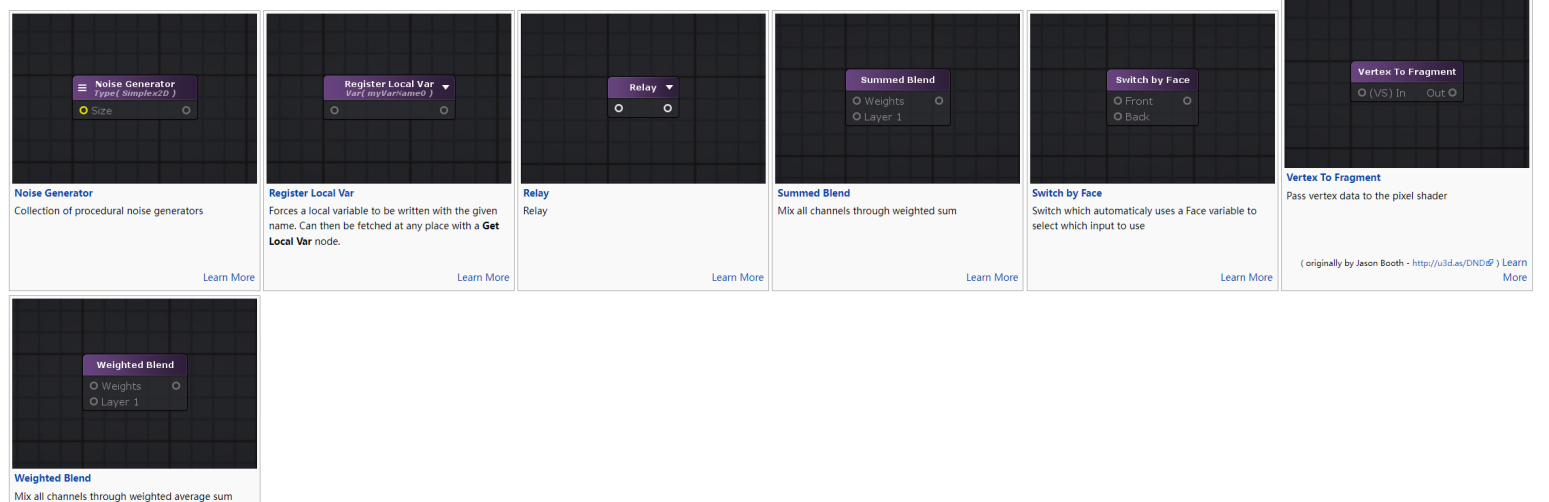


## 9 Matrix Transform

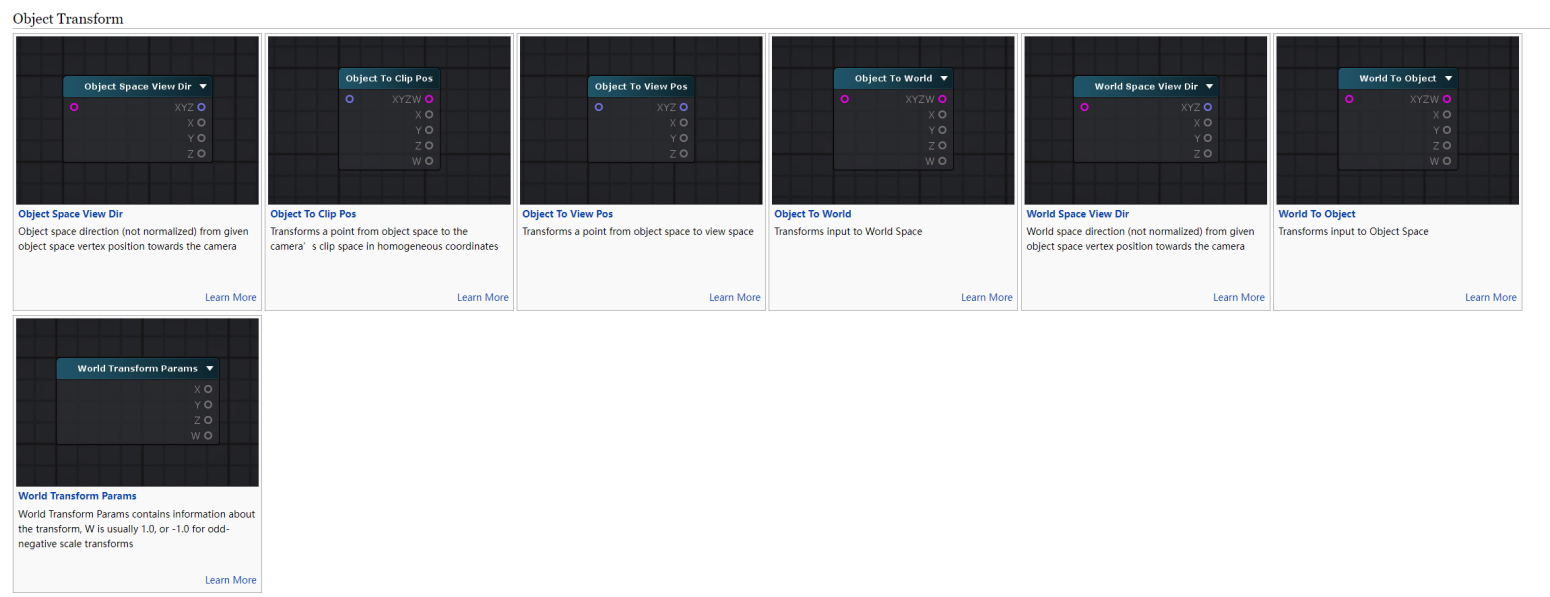


## 10 Miscellaneous

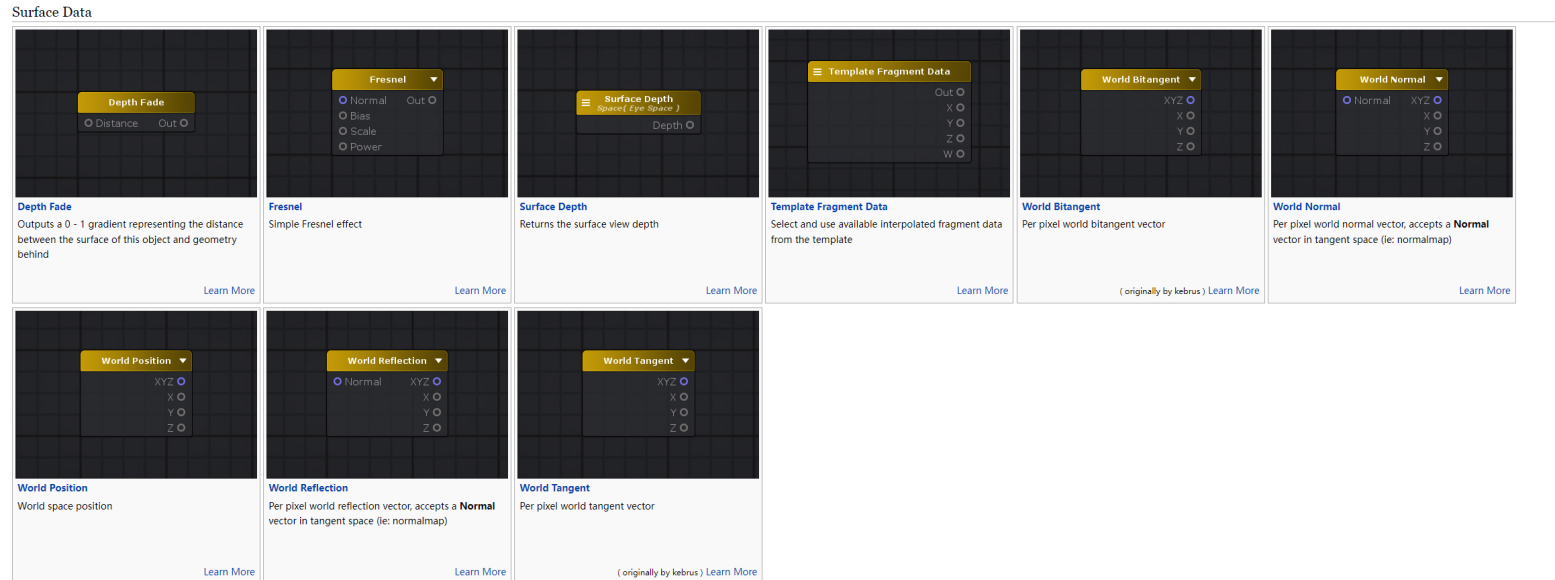




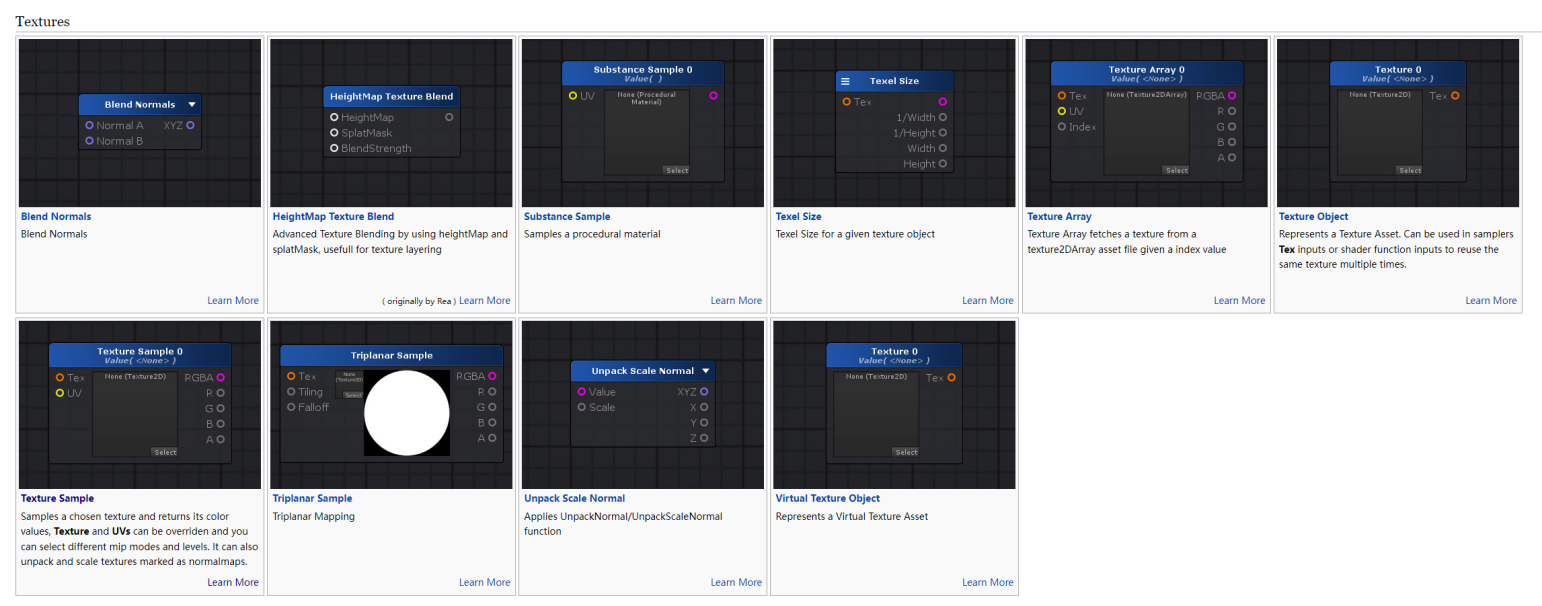
## 11 Object Transform



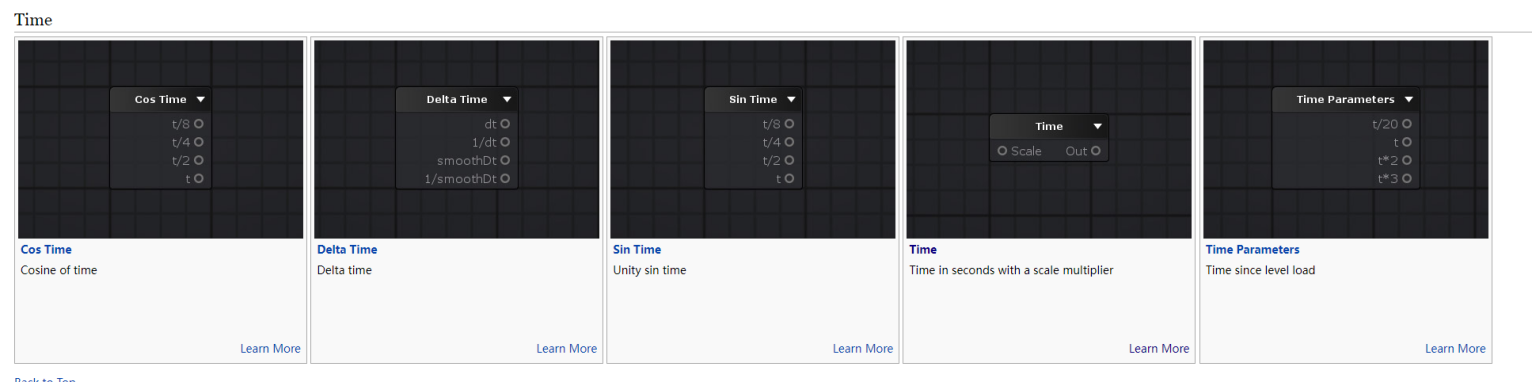
## 12 Surface Data



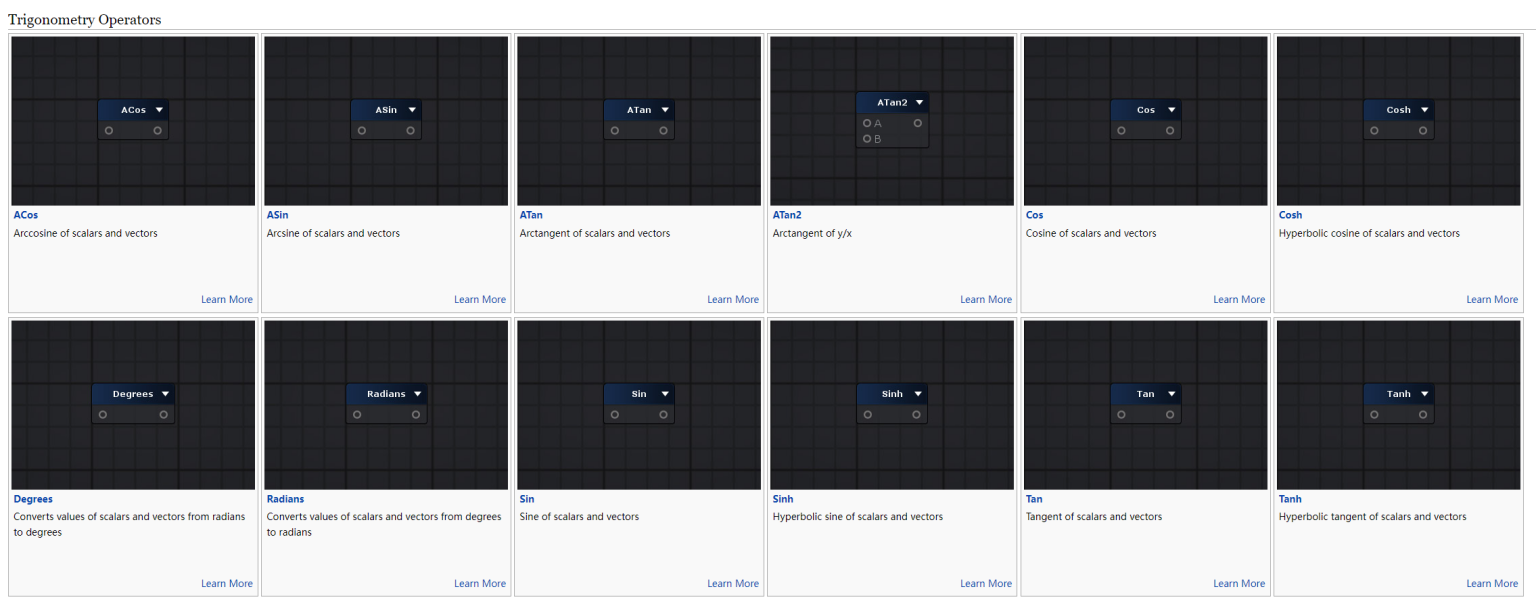
## 13 Textures



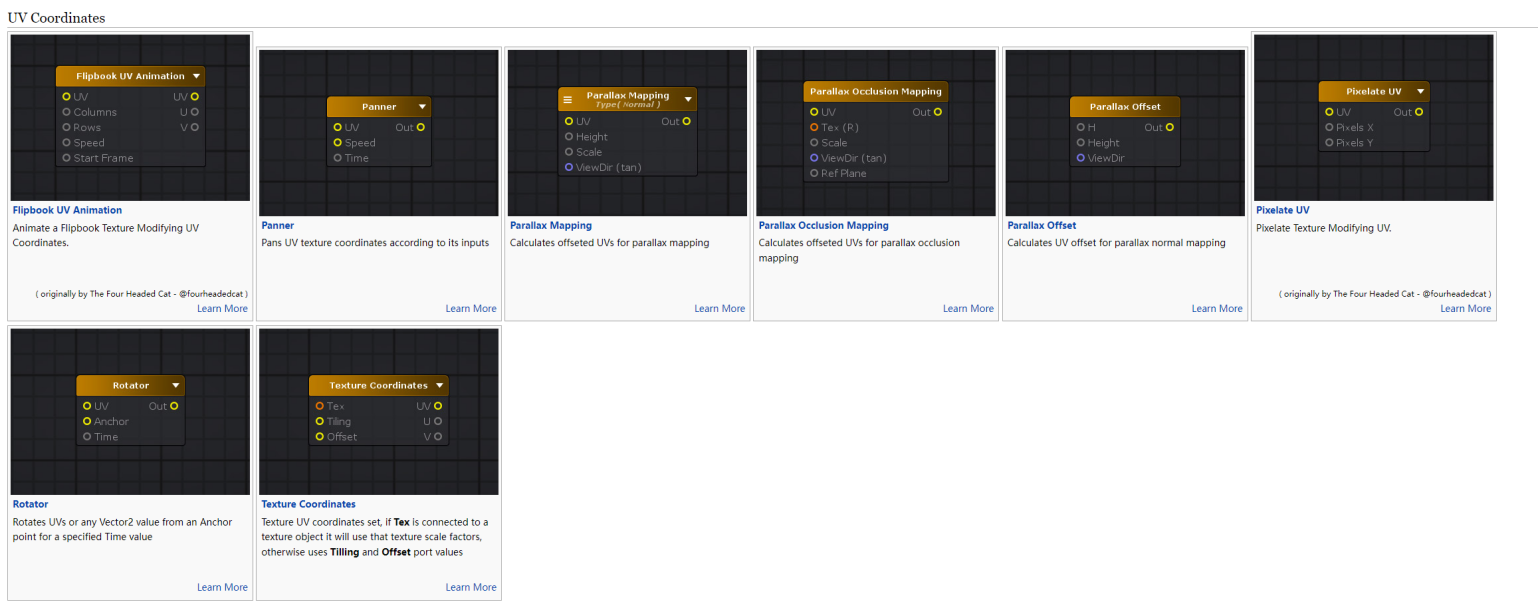
## 14 Time



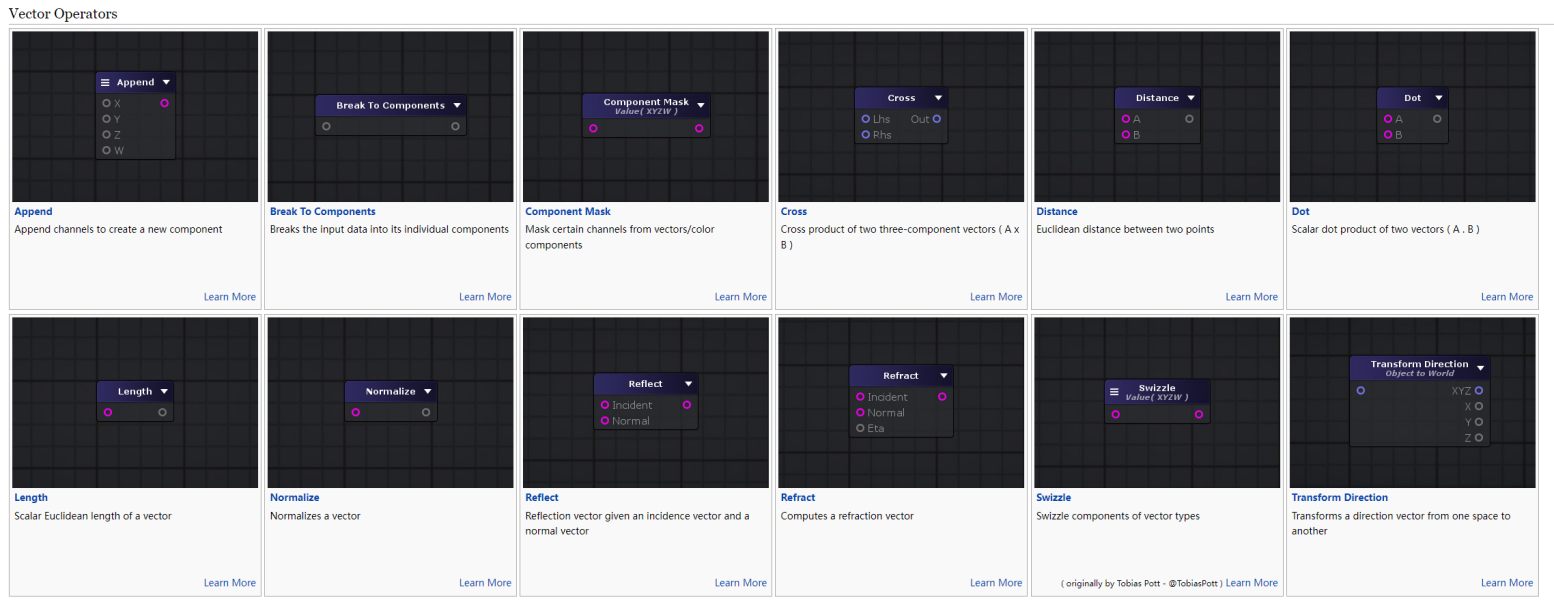
## 15 Trigonometry Operators



## 16 UV Coordinates

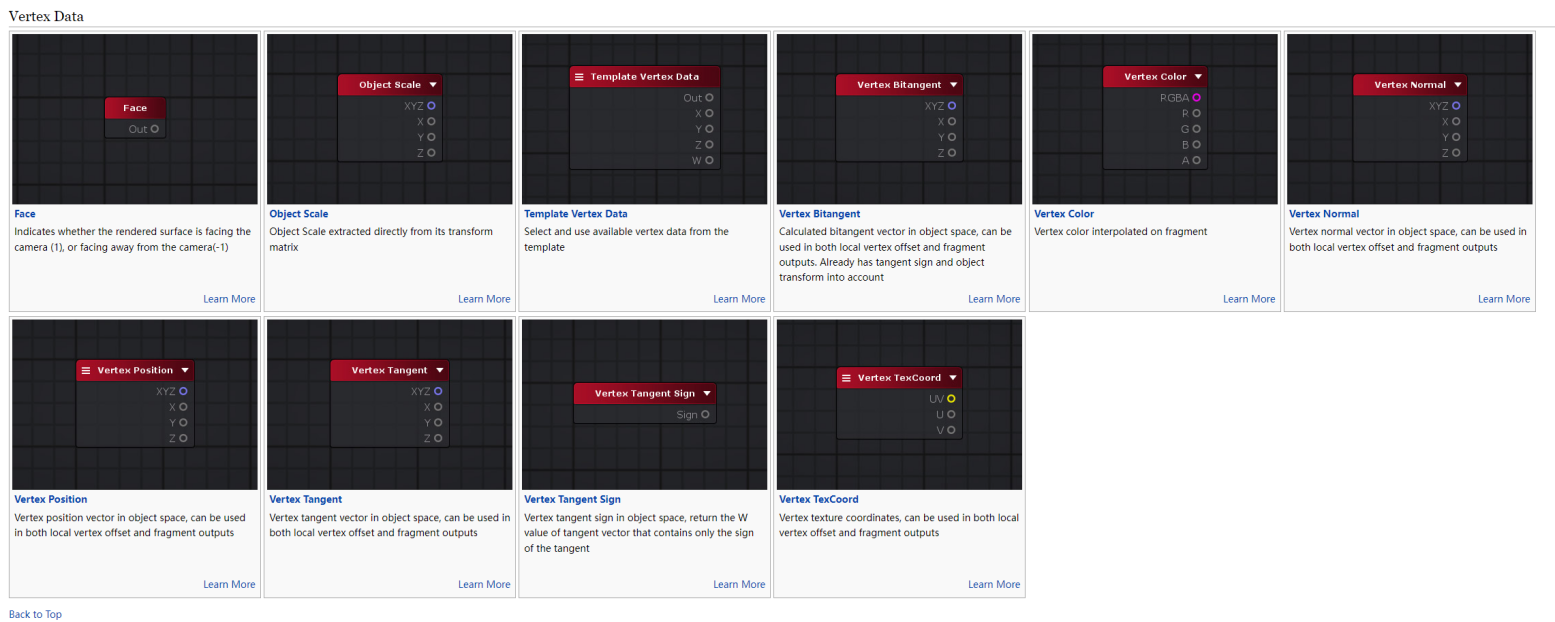


## 17 Vector Operators



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 中文 | 英文 | 概念 | 作用 | 掌握程度 |
|  | 追加/组合 | Append | 追加/组合多个输入向量 | 1 抽取UV坐标  2 组合向量 |  |
|  |  |  |  |  |  |
|  | 通道蒙版 | Component mask | 只选取部分通道 | 选取部分通道---比如RGBA 只选取RGB |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  | 点乘 | Dot | 判断向量是否垂直，也可以判断是否有抑制 | 1 判断向量是否垂直  2 也可以判断是否有抑制 |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## 18 Vertex Data



# Ase输出属性

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 中文 | 英文 | 概念 |
|  | 反射率/自身颜色 | Albedo |  |
|  | 法线 | Normal |  |
|  | 自发光/辐射 | Emission |  |
|  | 金属 | Metallic |  |
|  | 光滑度 | Smoothness |  |
|  | 环境光遮罩 | Ambient Occlusion |  |
|  | 透射 | Transmission |  |
|  |  | Translucency |  |
|  | 折射 | Refraction |  |
|  | 不透明 | Opacity |  |
|  | 不透明蒙版 | Opacity Mask |  |
|  | 自定义光照 | Custom Lighting |  |
|  |  | Local Vetex Offset |  |
|  |  | Local Vetex Normal |  |
|  |  | Tessellation |  |
|  |  | Debug |  |
|  |  |  |  |

### RenderQueue

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 中文 | 英文 | 概念 |
|  |  | BackGround | **1000** This render queue is rendered before any others. 这个队列通常被最先渲染（比如 天空盒）。 |
|  |  | Geometry | **2000**这是默认的渲染队列。它被用于绝大多数对象。不透明几何体使用该队列。 |
|  |  | AlphaTest | **2450** Alpha tested geometry uses this queue. 需要开启透明度测试的物体。Unity5以后从Geometry队列中拆出来,因为在所有不透明物体渲染完之后再渲染会比较高效。 |
|  |  | GeometryLast | Last render queue that is considered “opaque”. 所有Geometry和AlphaTest队列的物体渲染完后 |
|  |  | Transparent | **3000** This render queue is rendered after Geometry and AlphaTest, in back-to-front order. 所有Geometry和AlphaTest队列的物体渲染完后，再按照从后往前的顺序进行渲染,任何使用了透明度混合的物体都应该使用该队列（例如玻璃和粒子效果） |
|  |  | Overlay | **4000** This render queue is meant for overlay effects. 该队列用于实现一些叠加效果，适合最后渲染的物体（如镜头光晕）。 |

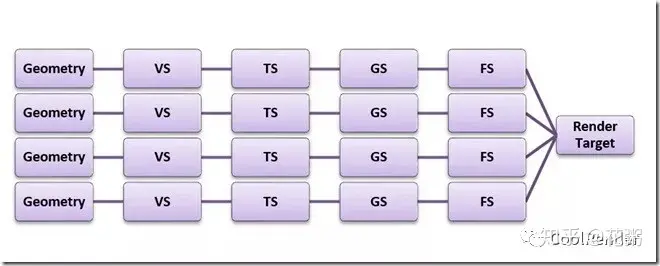
### Shader type

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 中文 | 英文 | 概念 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

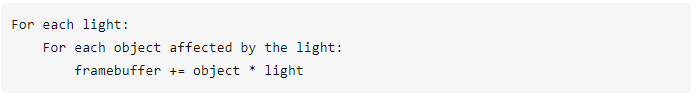
# 渲染概念

### 1 前向渲染

前向渲染中如果有10个物体，物体有不一样的材质，那么我们需要对每个物体进行着色



Forward Rendering是绝大数引擎都含有的一种渲染方式。要使用Forward Rendering，一般在Vertex Shader或Fragment Shader阶段对**每个顶点或每个像素进行光照计算**，并且是对**每个光源**进行计算产生最终结果。下面是Forward Rendering的核心伪代码[1]。



#### 优缺点：

很明显，对于Forward Rendering，光源数量对计算复杂度影响巨大，所以比较适合户外这种光源较少的场景。

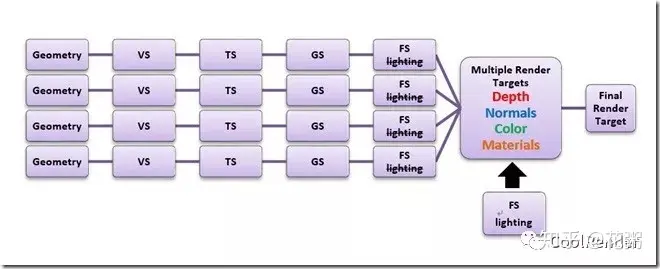
1.多在vertex shader中进行光照处理，因为有一个几何体有10000个顶点，那么对于n个光源，至少要在vertex shader中计算10000n次。而对于在fragment shader中进行处理，这种消耗会更多，因为对于一个普通的1024x768屏幕，将近有8百万的像素要处理。所以如果顶点数小于像素个数的话，尽量在vertex shader中进行光照。

2.如果要在fragment shader中处理光照，我们大可不必对每个光源进行计算时，把所有像素都对该光源进行处理一次。因为每个光源都有其自己的作用区域。比如点光源的作用区域是一个球体，而平行光的作用区域就是整个空间了。对于不在此光照作用区域的像素就不进行处理。但是这样做的话，CPU端的负担将加重，因为要计算作用区域。

3.对于某个几何体，光源对其作用的程度是不同，所以有些作用程度特别小的光源可以不进行考虑。典型的例子就是Unity中只考虑重要程度最大的4个光源。

### 2 延迟渲染

而在延迟渲染中，10个物体，我们只需要2个Pass就可以了

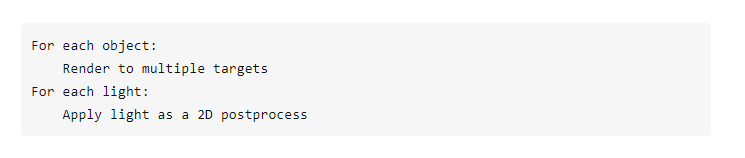


两个阶段：

1. 几何处理阶段(Geometry Pass)。**渲染所有的几何/颜色数据到G-buffer，**这个阶段中，我们获取对象的各种几何信息，并将第二步所需的各种数据储存（也就是渲染）到多个G-buffer中；
2. 2、光照处理阶段(Lighting Pass)。使用G-buffer计算场景的光照。，在这个pass中，我们只需渲染出一个屏幕大小的二维矩形，使用第一步在G-buffer中存储的数据对此矩阵的每一个片段计算场景的光照；光照计算的过程还是和正向渲染以前一样，只是现在我们需要从对应的G-buffer而不是顶点着色器(和一些uniform变量)那里获取输入变量了。

Deferred Rendering（延迟渲染）顾名思义，就是将光照处理这一步骤延迟一段时间再处理。具体做法就是将光照处理这一步放在已经**三维物体生成二维图片之后**进行处理。也就是说将物空间的光照处理**放到了像空间**进行处理。

要做到这一步，需要一个重要的辅助工具——**G-Buffer**。G-Buffer主要是用来存储每个像素对应的Position，Normal，Diffuse Color和其他Material parameters。根据这些信息，我们就可以在像空间中对每个像素进行光照处理。下面是Deferred Rendering的核心伪代码。



#### 优缺点：

Deferred Rendering的最大的优势就是将光源的数目和场景中物体的数目在复杂度层面上完全分开。也就是说场景中不管是一个三角形还是一百万个三角形，最后的复杂度不会随光源数目变化而产生巨大变化。

**缺点**也是显而易见，为了存储每个像素对应的Position，Normal，Diffuse Color和其他Material parameters，**需要使用G-Buffer,随着场景的复杂增加G-Buffer会越来越大，增幅也是不能忽视的，**而且存取G-Buffer耗费的带宽也是不能忽视的。

### 3 Blend Mode

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 英语 | 中文 | 概念 |
|  | Alpha Blend | 透明度混合 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 渲染特效

## 遮罩

## 硬边溶解

使用Step 结合 蒙版制作

## 光边溶解

1 使用R或者其它通道当做透明度值

2 使用step将通道值和float值比较调节R通道的值，获得一个溶解遮罩

3 使用2相同的方式获得一个遮罩，不过要比上面的大一点，

4 3减2这时候可以得到边缘遮罩。然后给边缘遮罩上色，得到上色后的边缘图

5 混合主纹理和边缘图，使用lerp，alpha使用边缘遮罩

6 主alpha使用大的遮罩

注意：Blend Mode 要选 Alpha Blend。并添加到透明queue中

# 中英文字典

|  |  |  |
| --- | --- | --- |
| 序号 | 英文 | 中文 |
|  | dissolve | 使溶解 |
|  | diffusion | 漫反射 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |