Shader Graph

一. 菲涅尔效果-Fresnel Effect Node

菲涅尔效果是根据视角在表面上产生不同反射率的效果，当您接近掠射角时，会反射更多光。菲涅尔效果节点通过计算表面法线和视图方向之间的角度来近似得到，这个角度越大，返回值就越大。这种效果通常用于实现边缘照明，在许多艺术效果中都很常见。

1. 端口

名称	方向	类型	描述
Normal	Input	Vector 3	法线方向。默认绑定到世界空间法线
View Dir	Input	Vector 3	视口方向。默认绑定到世界空间视图方向
Power	Input	Vector 3	幂计算的指数
Out	Output	Float	输出结果

2. 生成的代码示例

以下示例代码表示此节点的一种可能结果。

```
void Unity_FresnelEffect_float(float3 Normal, float3 ViewDir, float Power, out float Out)
{
    Out = pow((1.0 - saturate(dot(normalize(Normal), normalize(ViewDir)))), Power);
}
```

二. 乘节点-Multiply Node

返回输入值 A 乘以输入值 B 的结果。如果两个输入都是向量类型，则输出类型将是与这些输入的评估类型具有相同维度的向量类型。如果两个输入都是矩阵类型，则输出类型将是与这些输入的评估类型具有相同维度的矩阵类型。如果一个输入是向量类型，另一个是矩阵类型，那么输出类型将是一个与输入向量具有相同维度的向量。

1. 端口

名称	方向	类型	描述
A	Input	Dynamic	第一个输入值
B	Input	Dynamic	第二个输入值
Out	Output	Dynamic	幂计算的指数

2. 生成的代码示例

以下示例代码表示该节点的不同可能结果。

1) 向量 * 向量

```
void Unity_Multiply_float4_float4(float4 A, float4 B, out float4 Out)
{
    Out = A * B;
}
```

2) 向量 * 矩阵

```
void Unity_Multiply_float4_float4x4(float4 A, float4x4 B, out float4 Out)
{
```

```
    Out = mul(A, B);
}
```

3) 矩阵 * 矩阵

```
void Unity_Multiply_float4x4_float4x4(float4x4 A, float4x4 B, out float4x4 Out)
{
    Out = mul(A, B);
}
```

三. 加法节点- Add Node

返回两个输入值 A 和 B 的总和。在 PS 中描述的话，A 和 B 就相当于两个图层，使用加法节点就是将两个图层进行了一个组合。

1. 端口

名称	方向	类型	描述
A	Input	Dynamic	输入的第一个值
B	Input	Dynamic	输入的第二个值
Out	Output	Dynamic	输出值

2. 生成的代码示例

```
void Unity_Add_float4(float4 A, float4 B, out float4 Out)
{
    Out = A + B;
}
```

四. 除法-Divide Node

1. 端口

名称	方向	类型	描述
A	Input	Dynamic	输入的第一个值
B	Input	Dynamic	输入的第二个值
Out	Output	Dynamic	输出值

2. 生成的代码示例

```
void Unity_Divide_float4(float4 A, float4 B, out float4 Out)
{
    Out = A / B;
}
```

五. 正切- Tangent Node

1. 端口

名称	方向	类型	描述
In	Input	Dynamic	输入值
Out	Output	Dynamic	输出值

2. 生成的代码示例

```
void Unity_Tangent_float4(float4 In, out float4 Out)
{
    Out = tan(In);
}
```

六. 步节点-Step Node

如果输入 In(B)的值大于或等于输入 Edge(A)的值，则返回 1（白色），否则返回 0（黑色）。也就是说 Step 节点输出的内容非黑即白，不存在中间的过渡，我们可以用 Step 节点进行判断，即对图像进行切割。

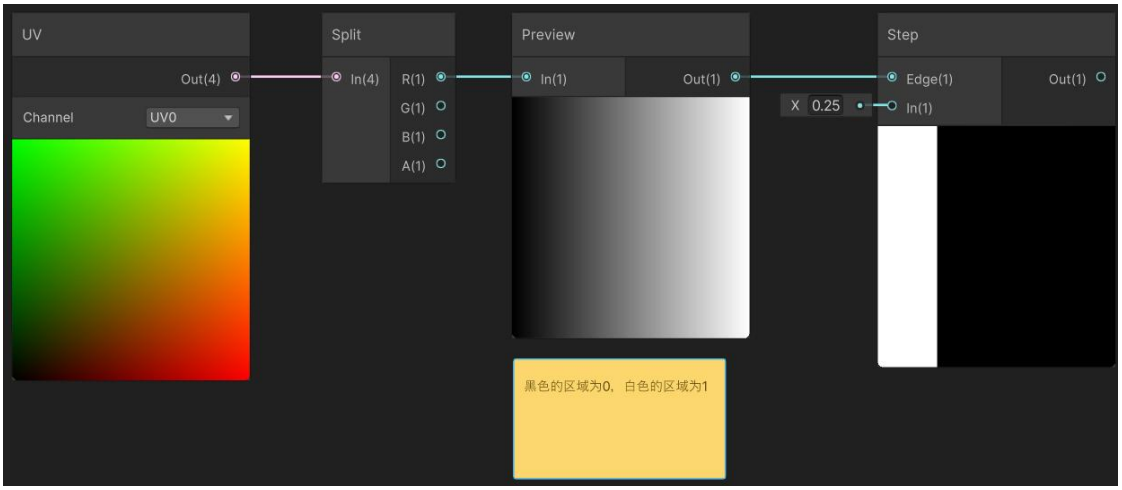

图 1.　　图 4.1 Step 节点可视化

由上图可以看到，输入的值 B 为 0.25，值 A 为一张从左(0)到右(1)的渐变纹理。因为输入的 A 值左四分之一位置的值都小于等于 B 值，因此在输出的时候都被输出为 1（白色）；右四分之三位置的值都大于 B 值，因此在输出的时候都被输出为 0（黑色）。

1. 端口

名称	方向	类型	描述
Edge	Input	Dynamic	Step 值
In	Input	Dynamic	输入值
Out	Output	Dynamic	输出值

2. 生成的代码示例

```
void Unity_Step_float4(float4 Edge, float4 In, out float4 Out)
{
    Out = step(Edge, In);
}
```

七. 平滑步节点-Smoothstep Node

如果输入 In 的值分别介于输入 Edge1 和 Edge2 的值之间，则返回 0 和 1 之间的平滑 Hermite 插值结果。如果输入 In 的值小于输入 Step1 的值，则返回 0；如果大于输入 Step2 的值，则返回 1 。

该节点类似于 Lerp 节点，但有两个显着差异。首先，用户使用此节点指定范围，返回值介于 0 和 1 之间。这可以看作与 Lerp 节点相反。其次，该节点使用平滑 Hermite 插值而不是线性插值。这意味着插值将从开始逐渐加速并逐渐减慢。这对于创建看起来自然的动画、淡入淡出和其他过渡非常有用。

1. 端口

名称	方向	类型	描述
Edge1	Input	Dynamic Vector	最小 Step 值
Edge2	Input	Dynamic Vector	最大 Step 值
In	Input	Dynamic Vector	输入值

Out	Output	Dynamic Vector	输出值

2. 生成的代码示例

```
void Unity_Smoothstep_float4(float4 Edge1, float4 Edge2, float4 In, out float4 Out)
{
    Out = smoothstep(Step1, Step2, In);
}
```

八. 极坐标-Polar Coordinates Node

将输入 UV 的值转换为极坐标。在数学中，极坐标系是一个二维坐标系，这个坐标系中的每个点，由该点距离参考点的距离和距离参考方向的角度确定。

由此产生的效果是，UV 输入的 x 通道将转换为与输入中心值指定的点的距离值，而 UV 的 y 通道将转换为围绕该点的旋转角度值。

这些值可以分别通过输入 Radial Scale（径向比例）和 Length Scale（长度比例）的值进行缩放。

1. 端口

名称	方向	类型	描述
UV	Input	Vector 2	输入的 UV 值
Center	Input	Vector 2	中心参考点
Radial Scale	Input	Float	距离值的比例
Length Scale	Input	Float	角度值的刻度
Out	Output	Vector 2	输出值

2. 生成的代码示例

```
void Unity_PolarCoordinates_float(float2 UV, float2 Center, float RadialScale, float LengthScale, out float2 Out)
{
    float2 delta = UV - Center;
    float radius = length(delta) * 2 * RadialScale;
    float angle = atan2(delta.x, delta.y) * 1.0/6.28 * LengthScale;
    Out = float2(radius, angle);
}
```

九. 重新映射-Remap Node

根据输入 In Min Max 的 x 和 y 分量之间的输入 In 值的线性插值，返回输入 Out Min Max 的 x 和 y 分量之间的值。

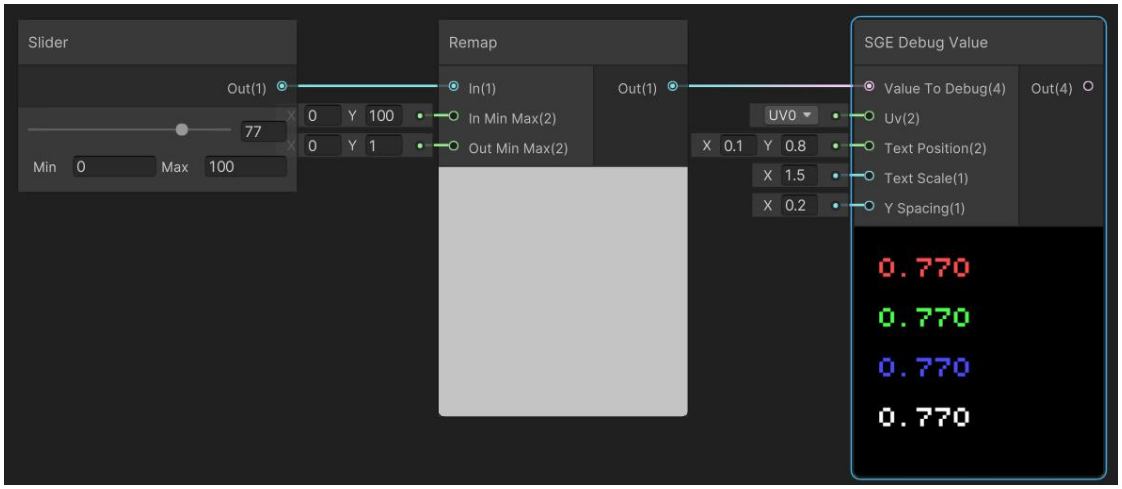

图 2.　　图 6.1 Remap Node 示例

当我们为 Remap 节点输入一个值，其最终的输出值为（输入值-输入最小值）*（输出最大值-输出最小值）/（输入最大值-输入最小值）+输出最小值。

1. 端口

名称	方向	类型	描述
In	Input	Dynamic Vector	输入值
In Min Max	Input	Vector 2	输入插值的最小值和最大值
Out Min Max	Input	Vector 2	输出插值的最小值和最大值
Out	Output	Dynamic Vector	输出值

2. 生成的代码示例

```
void Unity_Remap_float4(float4 In, float2 InMinMax, float2 OutMinMax, out float4 Out)
{
    Out = OutMinMax.x + (In - InMinMax.x) * (OutMinMax.y - OutMinMax.x) / (InMinMax.y - InMinMax.x);
}
```

十. 限制-Clamp Node

Clamp[1]节点的作用就是限制输入值的范围，将输入值限制在设定的最小值和最大值之间。如果在 Shader Graph，创建节点时输入 Limit，出现的节点也是 Clamp 节点，即 Clamp 节点就是起限制的作用。

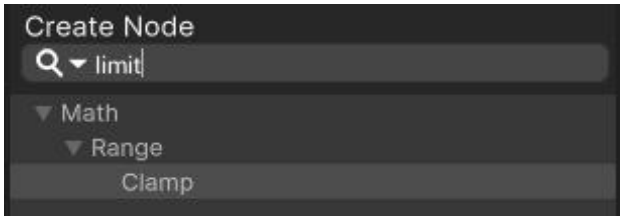

图 3.　　图 6.2 创建节点时输入 Limit

[1] n. 夹具，夹钳；v. （用夹具）夹紧，夹住；

1. 端口

名称	方向	类型	描述
In	Input	Dynamic Vector	输入值
Min	Input	Dynamic Vector	限制范围最小值
Max	Input	Dynamic Vector	限制范围最大值
Out	Output	Dynamic Vector	输出值

2. 生成的代码示例

```
void Unity_Clamp_float4(float4 In, float4 Min, float4 Max, out float4 Out)
{
    Out = clamp(In, Min, Max);
}
```

十一. 时间-Time Node

提供访问各种时间的着色器中的参数。

1. 端口

名称	方向	类型	描述
Time	Output	Float	时间值
Sine Time	Output	Float	正弦时间值
Cosine Time	Output	Float	余弦时间值
Delta Time	Output	Float	当前帧时间值
Smooth Delta	Output	Float	当前帧平滑时间值

Time 端口就如字面意思，代表时间的流逝，Time 端口输出的值会不断增加。

Sine Time 和 Cosine Time 端口，因为分别代表的是正弦时间值和余弦时间值，而正弦和余弦都是一个周期函数，且值都在负 1 到正 1 之间，因此使用这两个端口会具有周期性。

2. 生成的代码示例

```
float Time_Time = _Time.y;
float Time_SineTime = _SinTime.w;
float Time_CosineTime = _CosTime.w;
float Time_DeltaTime = unity_DeltaTime.x;
float Time_SmoothDelta = unity_DeltaTime.z;
```

十二. 瓦片与偏移-Tiling And Offset Node

1. 端口

名称	方向	类型	描述
UV	Input	Vector 2	输入的 UV 值
Tiling	Input	Vector 2	每个通道的瓦片数
Offset	Input	Vector 2	每个通道的偏移量
Out	Output	Vector 2	输出的 UV 值

Tiling 端口是一个二维向量，X 分量代表横向重复多少图，Y 分量代表纵向重复多少图。如下图所示，将分量都设置为了 2，图像的重复数就变成了 4，即 X 分量乘 Y 分量的值。

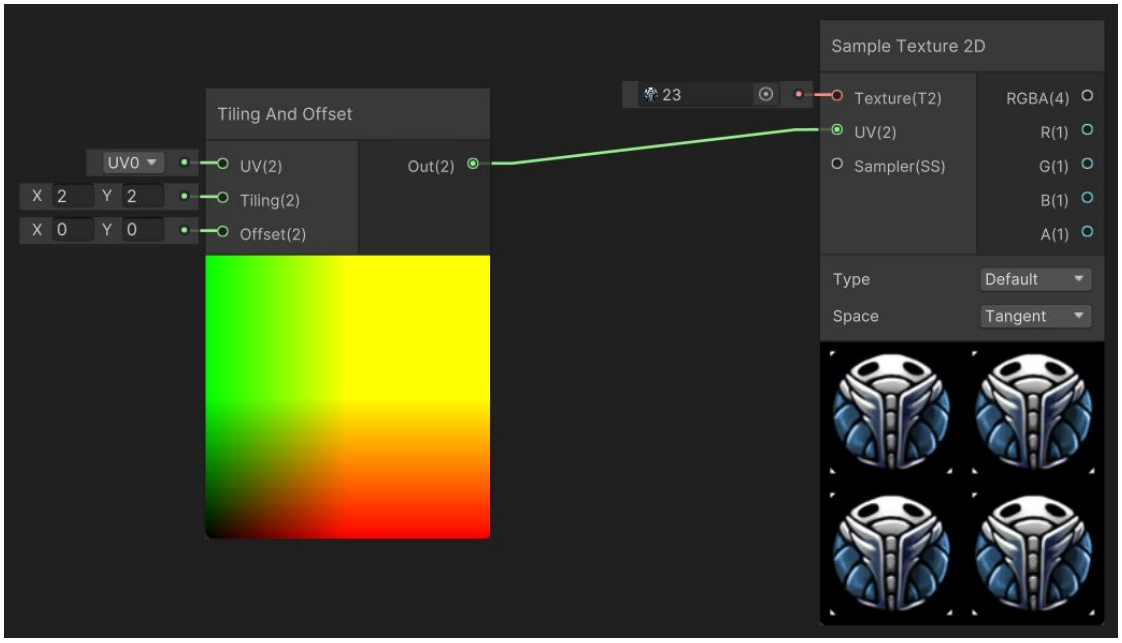

图 4.　　图 9.1 将 Tiling 的分量都设置为 2

注意，要让图像重复，该图像在 Unity 中的设置必须为 Repeat[2]。

图 5.　　设置图像的 Wrap Mode

Offset 端口也是一个二维向量，X 分量代表横向移动多少，Y 分量代表纵向移动多少。

2. 生成的代码示例

```
void Unity_TilingAndOffset_float(float2 UV, float2 Tiling, float2 Offset, out float2 Out)
{
    Out = UV * Tiling + Offset;
}
```

十三.　取反-Neagte Node

返回输入值的反向符号值。正值会变成负值，负值变为正值。

1. 端口

名称	方向	类型	描述
In	Input	Dynamic Vector	输入值
Out	Output	Dynamic Vector	输出值

2. 生成的代码示例

```
void Unity_Negate_float4(float4 In, out float4 Out)
{
    Out = -1 * In;
}
```

[2] v. 重复，重说；复述，跟读（尤指为学习）；
n.（事件的）重演，重现；重播节目；（同类货物的）再次运送；

十四. 简单噪声-Simple Noise Node

1. 端口

名称	方向	类型	描述
UV	Input	Vector 2	输入 UV 值
Scale	Input	Float	噪声比例
Out	Output	Float	输出值

2. 控件

名称	类型	选项	描述
Hash Type	Dropdown（下拉菜单）	Deterministic, LegacySine	选择用于生成噪声的随机数的散列函数

3. 生成的代码示例

```
inline float unity_noise_randomValue (float2 uv)
{
    return frac(sin(dot(uv, float2(12.9898, 78.233)))*43758.5453);
}

inline float unity_noise_interpolate (float a, float b, float t)
{
    return (1.0-t)*a + (t*b);
}

inline float unity_valueNoise (float2 uv)
{
    float2 i = floor(uv);
    float2 f = frac(uv);
    f = f * f * (3.0 - 2.0 * f);

    uv = abs(frac(uv) - 0.5);
    float2 c0 = i + float2(0.0, 0.0);
    float2 c1 = i + float2(1.0, 0.0);
    float2 c2 = i + float2(0.0, 1.0);
    float2 c3 = i + float2(1.0, 1.0);
    float r0 = unity_noise_randomValue(c0);
    float r1 = unity_noise_randomValue(c1);
    float r2 = unity_noise_randomValue(c2);
    float r3 = unity_noise_randomValue(c3);

    float bottomOfGrid = unity_noise_interpolate(r0, r1, f.x);
    float topOfGrid = unity_noise_interpolate(r2, r3, f.x);
    float t = unity_noise_interpolate(bottomOfGrid, topOfGrid, f.y);
    return t;
}
```

```
void Unity_SimpleNoise_float(float2 UV, float Scale, out float Out)
{
    float t = 0.0;

    float freq = pow(2.0, float(0));
    float amp = pow(0.5, float(3-0));
    t += unity_valueNoise(float2(UV.x*Scale/freq, UV.y*Scale/freq))*amp;

    freq = pow(2.0, float(1));
    amp = pow(0.5, float(3-1));
    t += unity_valueNoise(float2(UV.x*Scale/freq, UV.y*Scale/freq))*amp;

    freq = pow(2.0, float(2));
    amp = pow(0.5, float(3-2));
    t += unity_valueNoise(float2(UV.x*Scale/freq, UV.y*Scale/freq))*amp;

    Out = t;
}
```

十五. 旋转-Rotate Node

1. 端口

名称	方向	类型	描述
UV	Input	Vector 2	输入 UV 值
Center	Input	Vector 2	中心点到 旋转 大约
Rotation	Input	Float	要应用的旋转量
Out	Output	Vector 2	输出 UV 值

2. 控件

名称	类型	选项	描述
Unit	Dropdown（下拉菜单）	Radians（弧度），Degrees（度数）	切换旋转值输入单位

3. 生成的代码示例

1) 弧度

```
void Unity_Rotate_Radians_float(float2 UV, float2 Center, float Rotation, out float2 Out)
{
    UV -= Center;
    float s = sin(Rotation);
    float c = cos(Rotation);
    float2x2 rMatrix = float2x2(c, -s, s, c);
    rMatrix *= 0.5;
    rMatrix += 0.5;
    rMatrix = rMatrix * 2 - 1;
    UV.xy = mul(UV.xy, rMatrix);
```

```
    UV += Center;
    Out = UV;
}
```

2) 度数

```
void Unity_Rotate_Degrees_float(float2 UV, float2 Center, float Rotation, out float2 Out)
{
    Rotation = Rotation * (3.1415926f/180.0f);
    UV -= Center;
    float s = sin(Rotation);
    float c = cos(Rotation);
    float2x2 rMatrix = float2x2(c, -s, s, c);
    rMatrix *= 0.5;
    rMatrix += 0.5;
    rMatrix = rMatrix * 2 - 1;
    UV.xy = mul(UV.xy, rMatrix);
    UV += Center;
    Out = UV;
}
```

十六. 幂节点-Power Node

返回值为 A 的 B 次幂，例如输入的 A 值为 2，B 值为 3，则输出的值就为 8。

1. 端口

名称	方向	类型	描述
A	Input	Dynamic Vector	底数
B	Input	Dynamic Vector	指数
Out	Output	Dynamic Vector	输出值

2. 生成的代码示例

```
void Unity_Power_float4(float4 A, float4 B, out float4 Out)
{
    Out = pow(A, B);
}
```

十七. 饱和节点-Saturate Node

返回介于 0 和 1 之间的 In 的值。

1. 端口

名称	方向	类型	描述
In	Input	Dynamic Vector	输入值
Out	Output	Dynamic Vector	输出值

2. 生成的代码示例

```
void Unity_Saturate_float4(float4 In, out float4 Out)
{
    Out = saturate(In);
}
```

十八. 线性插值-Lerp Node

根据 T 值，返回在 A 和 B 之间线性插值的结果。

例如，当输入 T 的值为 0 时，返回值等于输入 A 的值，当它为 1 时，返回值等于输入 B 的值，当它为 0.5 时，返回值是 A 和 B 的中点。

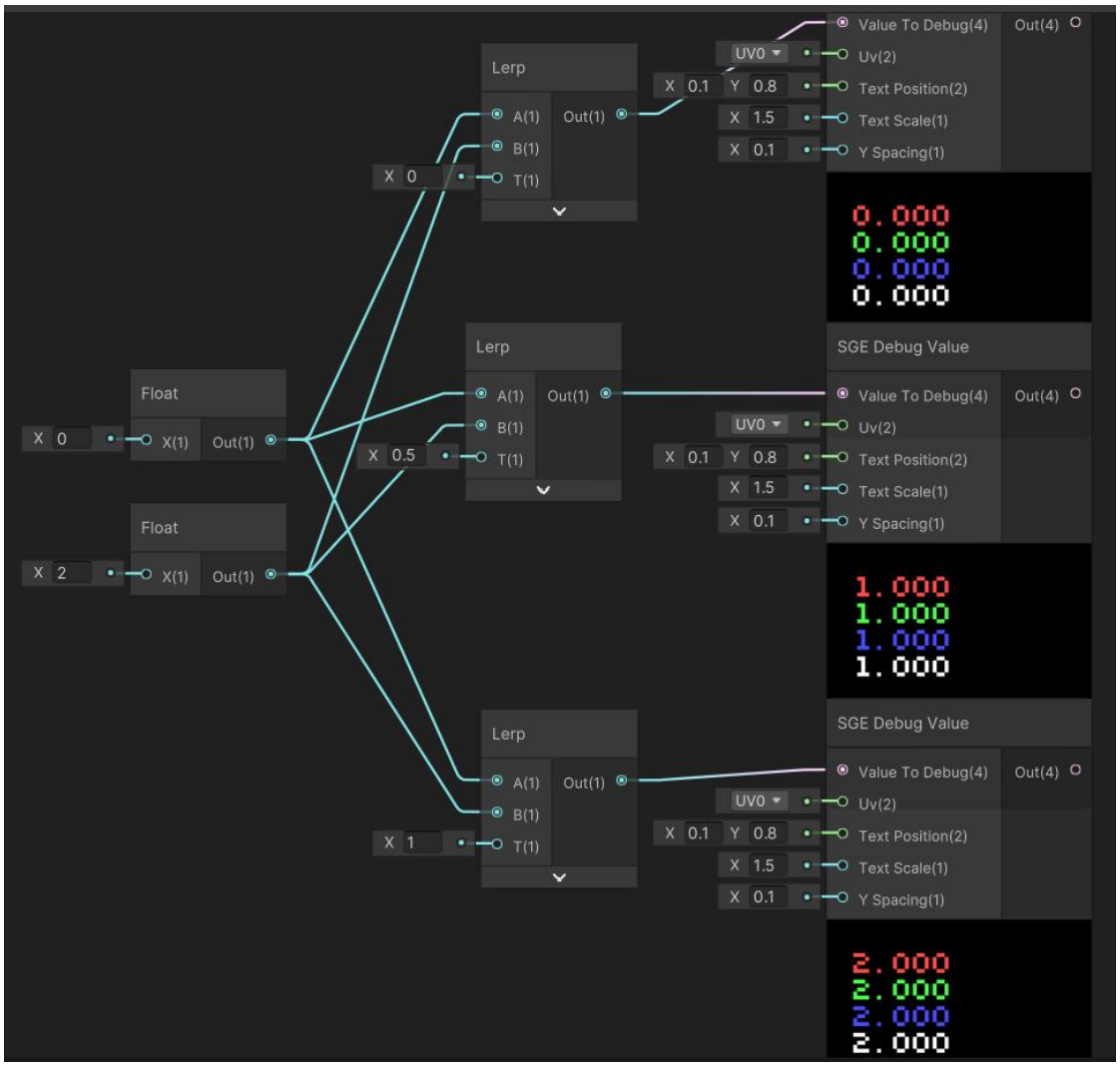

图 6. 线性插值图示

1. 端口

名称	方向	类型	描述
A	Input	Dynamic Vector	第一个输入值
B	Input	Dynamic Vector	第二个输入值
T	Input	Dynamic Vector	时间输入值
Out	Output	Dynamic Vector	输出值

2. 生成的代码示例

```
void Unity_Lerp_float4(float4 A, float4 B, float4 T, out float4 Out)
{
    Out = lerp(A, B, T);
}
```

十九. 反插值- Inverse Lerp Node

返回在 A 到 B 的范围内由输入 T 指定的插值线性参数。逆 Lerp 是 Lerp 节点的逆运算。

它可用于根据 Lerp 的输出值，来确定 Lerp 的 T 值是什么。 与 lerp 正好相反，Inverse Lerp 是计算出 T 值在 A 到 B 之间所占比例。

例如，T 值为 1 的 Lerp 在 0 和 2 之间的值为 0.5。 因此，在 0 和 2 之间且 T 值为 0.5 的 Inverse Lerp 的值为 1。

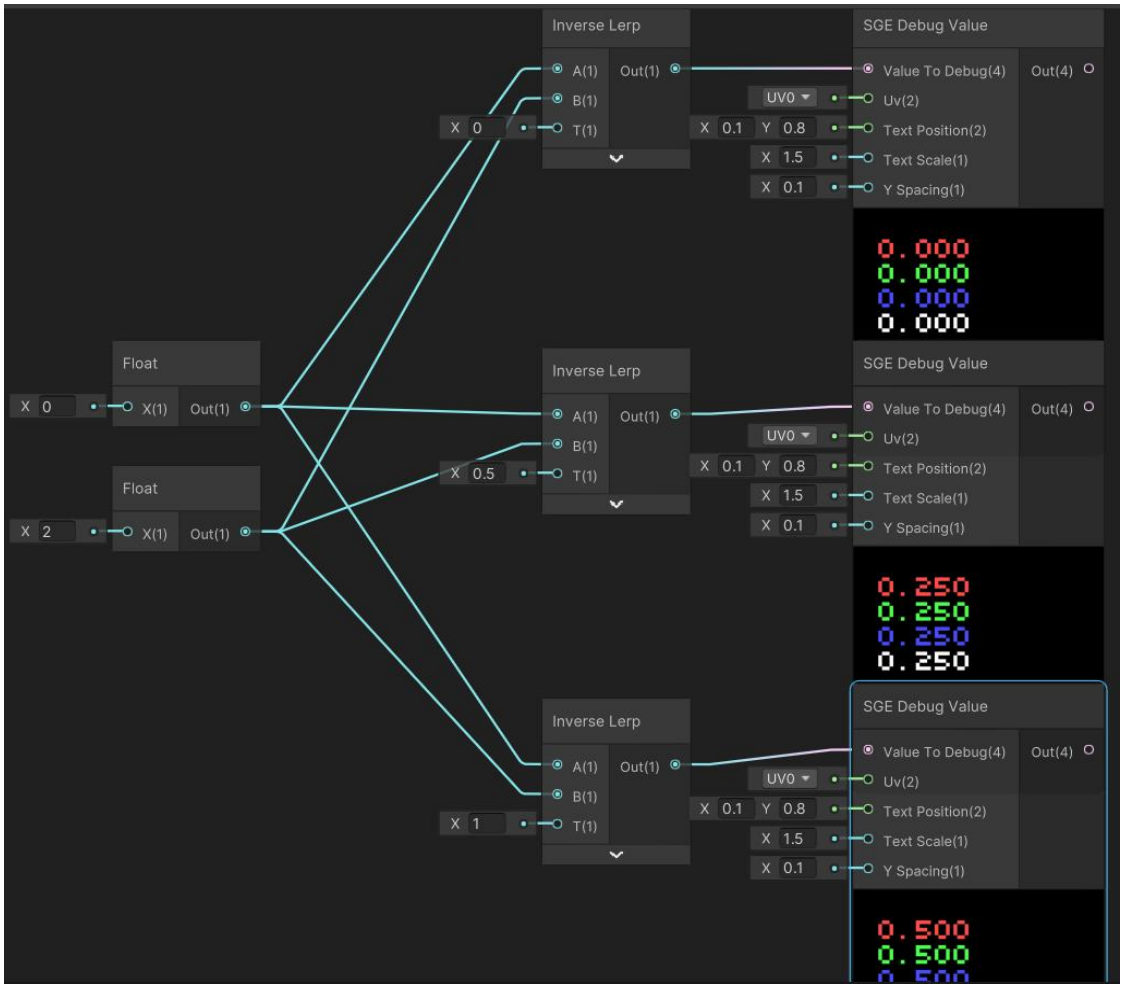

图 7. 反插值图示

如图所示，反插值输出的值就是均匀分布的分布函数所对应的值。

当输入的 A 值为 0，B 值为 1，T 为 1 时，T 所占区间长度的比例为二分之一，当 T 为 0.5 时，所占区间长度的比例为四分之一。

1. 端口

名称	方向	类型	描述
A	Input	Dynamic Vector	第一个输入值
B	Input	Dynamic Vector	第二个输入值
T	Input	Dynamic Vector	时间输入值
Out	Output	Dynamic Vector	输出值

2. 生成的代码示例

```
void Unity_InverseLerp_float4(float4 A, float4 B, float4 T, out float4 Out)
```

```
{
    Out = (T - A)/(B - A);
}
```

二十. 翻书-Flipbook Node

创建一个 翻动提供给输入 UV 的 UV 的书或纹理表动画。工作表上的平铺数量由输入 Width 和 Height 的值定义。当前 Tile 的索引由输入 Tile 的值定义。

该节点可用于创建纹理动画功能，通常用于粒子效果和精灵，方法是向输入 Tile 提供 Time 并输出到 Texture Sampler 的 UV 输入端口。

从 UV 空间的左下角开始，UV 数据通常在 0 到 1 的范围内。这可以通过 UV 预览左下角的黑色值看出。作为翻动书籍通常从左上角开始，默认情况下启用反转 Y 参数。

1. 端口

名称	方向	类型	描述
UV	Input	Vector 2	输入 UV 值
Width	Input	Float	水平瓦片数量
Height	Input	Float	垂直瓦片数量
Tile	Input	Float	当前瓦片索引
Out	Output	Vector 2	输出值

2. 控件

名称	类型	选项	描述
Invert X	Toggle	True, False	如果启用，则图块从右到左迭代
Invert Y	Toggle	True, False	如果启用，则图块从上到下迭代

1) 启用 X 关闭 Y

图片最下层为第一行，最右边为第一列。

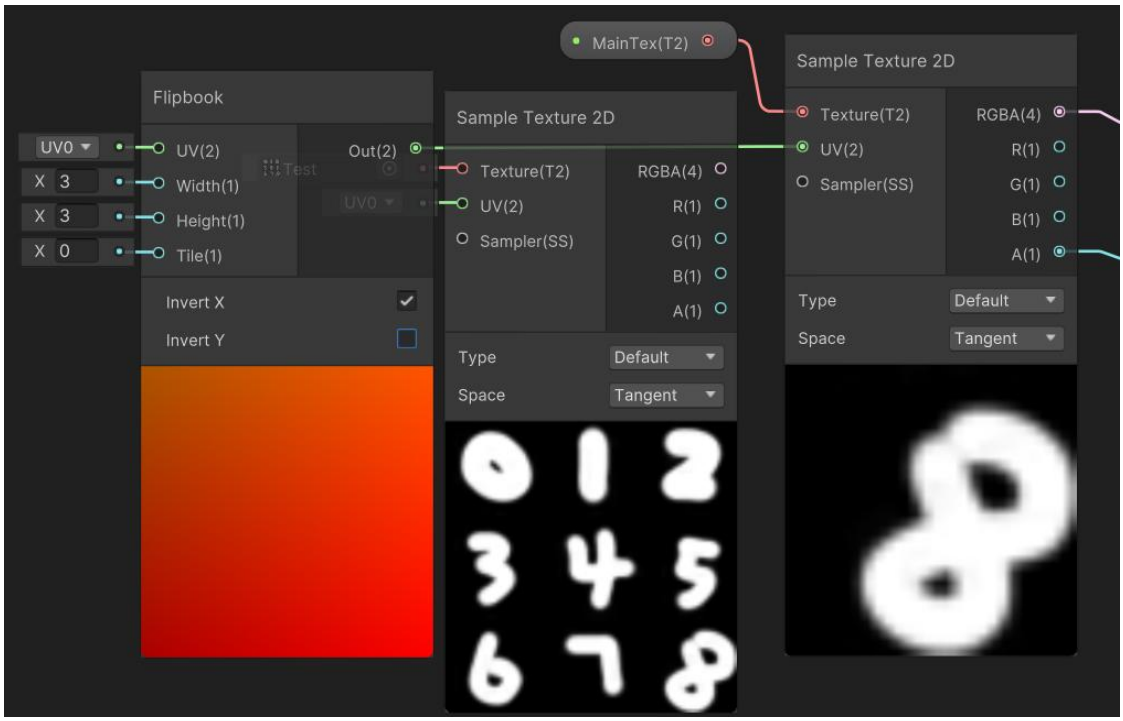

图 8.　　启用 X 关闭 Y

2) 启用 X 和 Y

图片最上层为第一行，最右边为第一列。

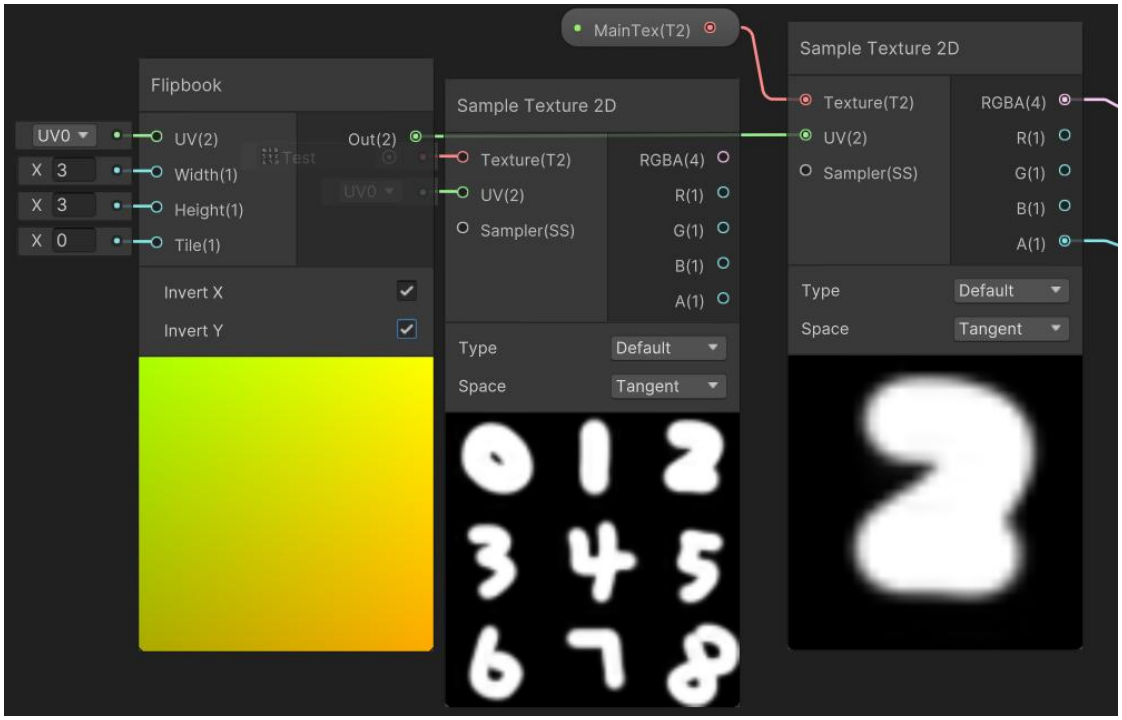

图 9.　　启用 X 和 Y

3. 生成的代码示例

```
float2 _Flipbook_Invert = float2(FlipX, FlipY);
void Unity_Flipbook_float(float2 UV, float Width, float Height, float Tile, float2 Invert, out float2 Out)
{
    Tile = fmod(Tile, Width * Height);
    float2 tileCount = float2(1.0, 1.0) / float2(Width, Height);
    float tileY = abs(Invert.y * Height - (floor(Tile * tileCount.x) + Invert.y * 1));
    float tileX = abs(Invert.x * Width - ((Tile - Width * floor(Tile * tileCount.x)) + Invert.x * 1));
    Out = (UV + float2(tileX, tileY)) * tileCount;
}
```

二十一. 向下取整- Floor Node

返回小于或等于输入值 In 的最大整数值或整数。

1. 端口

名称	方向	类型	描述
In	Input	Dynamic Vector	输入值
Out	Output	Dynamic Vector	输出值

2. 生成的代码示例

```
void Unity_Floor_float4(float4 In, out float4 Out)
{
```

```
    Out = floor(In);
}
```

二十二. 向上取整- Ceiling Node

返回大于或等于输入值的最小整数值或整数。

1. 端口

名称	方向	类型	描述
In	Input	Dynamic Vector	输入值
Out	Output	Dynamic Vector	输出值

2. 生成的代码示例

```
void Unity_Ceiling_float4(float4 In, out float4 Out)
{
    Out = ceil(In);
}
```

二十三. 取余- Modulo Node

返回输入 A 除以输入 B 的余数。

1. 端口

名称	方向	类型	描述
A	Input	Dynamic Vector	第一个输入值
B	Input	Dynamic Vector	第二个输入值
Out	Output	Dynamic Vector	输出值

2. 生成的代码示例

```
void Unity_Modulo_float4(float4 A, float4 B, out float4 Out)
{
    Out = fmod(A, B);
}
```

二十四. 场景深度-Scene Depth Node

使用输入 UV 提供对当前相机深度缓冲区的访问，该输入 UV 应该是标准化的屏幕坐标。

深度缓冲区访问需要在活动渲染管线上启用深度缓冲区。此过程因渲染管道而异。建议您阅读活动渲染管道的文档以获取有关启用深度缓冲区的信息。如果深度缓冲区不可用，此节点将返回中灰色。

该节点执行的 HLSL 代码是按 Render Pipeline 定义的，不同的 Render Pipelines 可能会产生不同的结果。 希望自定义渲染管道支持此节点，需要明确定义它的行为。 如果未定义，此节点将返回 1（白色）。

此节点只能在片元着色器阶段使用。

1. 端口

名称	方向	类型	描述
UV	Input	Vector 4	标准化屏幕坐标
Out	Output	Float	输出值

2. 深度采样模式

名称	描述
Linear01	0 到 1 之间的线性深度值

原生-Raw	原始深度值
Eye	深度转换为眼睛空间单位

3. 生成的代码示例

```
void Unity_SceneDepth_Raw_float(float4 UV, out float Out)
{
    Out = SHADERGRAPH_SAMPLE_SCENE_DEPTH(UV);
}
```

二十五. One Minus Node

返回 1 减去输入值的结果。

1. 端口

名称	方向	类型	描述
In	Input	Dynamic Vector	输入值
Out	Output	Dynamic Vector	输出值

2. 生成的代码示例

```
void Unity_OneMinus_float4(float4 In, out float4 Out)
{
    Out = 1 - In;
}
```

二十六. 位置-Position Node

提供对网格顶点或片段位置的访问，取决于节点所属的图形部分的有效着色器阶段。使用控件的下拉菜单选择输出值的坐标空间。

1. 端口

名称	方向	类型	描述
Out	Output	Dynamic Vector	输出网格顶点/片段的位置。

2. 控件

名称	类型	选项	描述
Space	下拉菜单	Object, View, World, Tangent, Absolute World	选择对应的坐标空间进行位置输出。

1) World and Absolute World

位置节点为我们提供了，World 和 Absolute World 空间这两个下拉选项。绝对世界(Absolute World)选项总是返回所有可编写脚本的渲染管线场景中对象的绝对世界位置。World 选项返回选定的 Scriptable Render Pipeline 的默认世界空间。

高清渲染管线(High Definition Render Pipeline)使用相管相机(Camera Relative)作为其默认世界空间。

通用渲染管线(Universal Render Pipeline)使用绝对世界作为其默认世界空间。

二十七. 绝对值-Absolute Node

返回输入值的绝对值，如果输入值为负则将其变为正数，输入值为正则保持不变。

1. 端口

名称	方向	类型	描述
In	Input	Dynamic Vector	输入值
Out	Output	Dynamic Vector	输出值

2. 生成的代码示例

```
void Unity_Absolute_float4(float4 In, out float4 Out)
{
    Out = abs(In);
}
```

二十八. 变换-Transform Node

将输入值（In）从一个坐标空间转移到另一个坐标空间。

1. 端口

名称	方向	类型	描述
In	Input	Vector 3	输入值
Out	Output	Vector 3	输出值

2. 控件

名称	类型	选项	描述
From	下拉菜单	Object, View, World, Tangent, Absolute World	选择要转换的空间
To	下拉菜单	Object, View, World, Tangent, Absolute World	选择要转换的空间

3. World and Absolute World

世界空间选项使用 Scriptable Render Pipeline 默认世界空间来转换位置坐标。绝对世界空间选项使用绝对世界空间来转换所有 Scriptable Render Pipelines 中的位置坐标。

如果使用变换节点转换不用于位置值的坐标空间，Unity 建议您使用 World space 选项。对不代表位置的值使用绝对世界可能会导致意外行为。

4. 生成的代码示例

1) World -> World

```
float3 _Transform_Out = In;
```

2) World -> Object

```
float3 _Transform_Out = TransformWorldToObject(In);
```

3) World -> Tangent

```
float3x3 tangentTransform_World = float3x3(IN.WorldSpaceTangent, IN.WorldSpaceBiTangent, IN.WorldSpaceNormal);
float3 _Transform_Out = TransformWorldToTangent(In, tangentTransform_World);
```

4) World -> View

```
float3 _Transform_Out = TransformWorldToView(In)
```

5) World -> Absolute World

```
float3 _Transform_Out = GetAbsolutePositionWS(In);
```

6) Object -> World

```
float3 _Transform_Out = TransformObjectToWorld(In);
```

7) Object -> Object

```
float3 _Transform_Out = In;
```

8) Object -> Tangent

```
float3x3 tangentTransform_World = float3x3(IN.WorldSpaceTangent, IN.WorldSpaceBiTangent, IN.WorldSpaceNormal);
float3 _Transform_Out = TransformWorldToTangent(TransformObjectToWorld(In), tangentTransform_World);
```

9) Object -> View

```
float3 _Transform_Out = TransformWorldToView(TransformObjectToWorld(In));
```

10) Object -> Absolute World

```
float3 _Transform_Out = GetAbsolutePositionWS(TransformObjectToWorld(In));
```

11) Tangent -> World

```
float3x3 transposeTangent = transpose(float3x3(IN.WorldSpaceTangent, IN.WorldSpaceBiTangent, IN.WorldSpaceNormal));
float3 _Transform_Out = mul(In, transposeTangent).xyz;
```

12) Tangent -> Object

```
float3x3 transposeTangent = transpose(float3x3(IN.WorldSpaceTangent, IN.WorldSpaceBiTangent, IN.WorldSpaceNormal));
float3 _Transform_Out = TransformWorldToObject(mul(In, transposeTangent).xyz);
```

13) Tangent -> Tangent

```
float3 _Transform_Out = In;
```

14) Tangent -> View

```
float3x3 transposeTangent = transpose(float3x3(IN.WorldSpaceTangent, IN.WorldSpaceBiTangent, IN.WorldSpaceNormal));
float3 _Transform_Out = TransformWorldToView(mul(In, transposeTangent).xyz);
```

15) Tangent -> Absolute World

```
float3x3 transposeTangent = transpose(float3x3(IN.WorldSpaceTangent, IN.WorldSpaceBiTangent, IN.WorldSpaceNormal));
float3 _Transform_Out = GetAbsolutePositionWS(mul(In, transposeTangent)).xyz;
```

16) View -> World

```
float3 _Transform_Out = mul(UNITY_MATRIX_I_V, float4(In, 1)).xyz;
```

17) View -> Object

```
float3 _Transform_Out = TransformWorldToObject(mul(UNITY_MATRIX_I_V, float4(In, 1) ).xyz);
```

18) View -> Tangent

```
float3x3 tangentTransform_World = float3x3(IN.WorldSpaceTangent, IN.WorldSpaceBiTangent, IN.WorldSpaceNormal);
float3 _Transform_Out = TransformWorldToTangent(mul(UNITY_MATRIX_I_V, float4(In, 1) ).xyz, tangentTransform_World);
```

19) View -> View

float3 _Transform_Out = In;

20) View -> Absolute World

float3 _Transform_Out = GetAbsolutePositionWS(mul(UNITY_MATRIX_I_V, float4(In, 1))).xyz;

21) Absolute World -> World

float3 _Transform_Out = GetCameraRelativePositionWS(In);

22) Absolute World -> Object

float3 _Transform_Out = TransformWorldToObject(In);

23) Absolute World -> Tangent

float3x3 tangentTransform_World = float3x3(IN.WorldSpaceTangent, IN.WorldSpaceBiTangent, IN.WorldSpaceNormal);

float3 _Transform_Out = TransformWorldToTangent(In, tangentTransform_World);

24) Absolute World -> View

float3 _Transform_Out = TransformWorldToView(In)

25) Absolute World -> Absolute World

float3 _Transform_Out = In;

二十九. 标准化-Normalize Node

在只关心方向，而不关心大小的情况下，我们需要将输入值进行标准化。输出向量将与输入值具有相同的方向，但长度为 1。

1. 端口

名称	方向	类型	描述
In	Input	Dynamic Vector	输入值
Out	Output	Dynamic Vector	输出值

2. 生成的代码示例

```
void Unity_Normalize_float4(float4 In, out float4 Out)
{
    Out = normalize(In);
}
```

三十. 长度-Length Node

返回长度输入向量的长度，也称为向量的大小。一个向量的长度用勾股定理计算。

一个二维向量的长度（向量的模）计算可以为：$\sqrt{x^2 + y^2}$

1. 端口

名称	方向	类型	描述
In	Input	Dynamic Vector	输入值
Out	Output	Float	输出值

2. 生成的代码示例

```
void Unity_Length_float4(float4 In, out float Out)
{
    Out = length(In);
```

}

三十一. 距离-Distance Node

返回输入 A 和 B 的值之间的欧几里德距离。这对于计算空间中两点之间的距离非常有用，并且通常用于计算有向距离函数(Signed Distance Function)。

1. 端口

名称	方向	类型	描述
A	Input	Dynamic Vector	第一个输入值
B	Input	Dynamic Vector	第二个输入值
Out	Output	Float	输出值

2. 生成的代码示例

```
void Unity_Distance_float4(float4 A, float4 B, out float Out)
{
    Out = distance(A, B);
}
```

三十二. 点乘-Dot Product Node

返回输入值的数量积。对于归一化的输入向量，如果指向完全相同的方向，则返回 1，如果它们指向完全相反的方向，则返回 -1，如果向量垂直，则返回 0。

1. 端口

名称	方向	类型	描述
A	Input	Dynamic Vector	第一个输入值
B	Input	Dynamic Vector	第二个输入值
Out	Output	Float	输出值

2. 生成的代码示例

```
void Unity_DotProduct_float4(float4 A, float4 B, out float Out)
{
    Out = dot(A, B);
}
```

三十三. 小数-Fraction Node

返回输入值的小数部分；大于等于 0 小于 1。

1. 端口

名称	方向	类型	描述
In	Input	Dynamic Vector	输入值
Out	Output	Dynamic Vector	输出值

2. 生成的代码示例

```
void Unity_Fraction_float4(float4 In, out float4 Out)
{
    Out = frac(In);
}
```

三十四. 随机范围-Random Range Node

返回一个基于 Seed 伪随机数，这个数介于输入的最小值(Min)和最大值(Max)之间。

1. 端口

名称	方向	类型	描述
Seed	Input	Vector 2	随机数种子
Min	Input	Float	最小值
Max	Input	Float	最大值
Out	Output	Float	输出值

2. 生成的代码示例

```
void Unity_RandomRange_float(float2 Seed, float Min, float Max, out float Out)
{
    float randomno =  frac(sin(dot(Seed, float2(12.9898, 78.233)))*43758.5453);
    Out = lerp(Min, Max, randomno);
}
```

三十五. 比较-Comparison Node

基于下拉菜单，去比较 A 和 B 的值，这通常用作分支节点的输入。

1. 端口

名称	方向	类型	描述
A	Input	Float	第一个输入值
B	Input	Float	第二个输入值
Out	Output	Boolean	输出值

2. 控件

名称	类型	选项	描述
	下拉菜单	Equal, NotEqual, Less, LessOrEqual, Greater, GreaterOrEqual	比较

3. 生成的代码示例

1) Equal

```
void Unity_Comparison_Equal_float(float A, float B, out float Out)
{
    Out = A == B ? 1 : 0;
}
```

2) NotEqual

```
void Unity_Comparison_NotEqual_float(float A, float B, out float Out)
{
    Out = A != B ? 1 : 0;
}
```

3) Less

```
void Unity_Comparison_Less_float(float A, float B, out float Out)
{
    Out = A < B ? 1 : 0;
}
```

4) LessOrEqual

```
void Unity_Comparison_LessOrEqual_float(float A, float B, out float Out)
{
    Out = A <= B ? 1 : 0;
}
```

5) Greater

```
void Unity_Comparison_Greater_float(float A, float B, out float Out)
{
    Out = A > B ? 1 : 0;
}
```

6) GreaterOrEqual

```
void Unity_Comparison_GreaterOrEqual_float(float A, float B, out float Out)
{
    Out = A >= B ? 1 : 0;
}
```

三十六. 指数-Exponential Node

输入值 In 为指数的幂。指数的底数 base 可以从节点上的 Base 下拉菜单中选择。

1. 端口

名称	方向	类型	描述
In	Input	Dynamic Vector	输入值
Out	Output	Dynamic Vector	输出值

2. 控件

名称	类型	选项	描述
Base	下拉菜单	BaseE, Base2	Base E：返回 e 的输入 In 的幂 Base 2：返回 2 的输入 In 的幂

3. 生成的代码示例

1) BaseE

```
void Unity_Exponential_float4(float4 In, out float4 Out)
{
    Out = exp(In);
}
```

2) Base2

```
void Unity_Exponential2_float4(float4 In, out float4 Out)
{
    Out = exp2(In);
}
```

三十七. 符号-Sign Node

对于每个组件，如果输入 In 的值小于零，则返回 -1，如果等于零，则返回 0，如果大

于零，则返回 1。

1. 端口

名称	方向	类型	描述
In	Input	Dynamic Vector	输入值
Out	Output	Dynamic Vector	输出值

2. 生成的代码示例

```
void Unity_Sign_float4(float4 In, out float4 Out)
{
    Out = sign(In);
}
```

三十八. 返回整数部分-Truncate

返回输入 In 值的整数或整数部分。 例如，给定输入值 1.7，此节点将返回值 1.0。

1. 端口

名称	方向	类型	描述
In	Input	Dynamic Vector	输入值
Out	Output	Dynamic Vector	输出值

2. 生成的代码示例

```
void Unity_Truncate_float4(float4 In, out float4 Out)
{
    Out = trunc(In);
}
```

三十九. 是否是正面——Is Front Face

如果当前正在渲染正面，则返回 true，如果渲染的是背面则为 false。

如果我们没有勾选 ShaderGraph 中的渲染双面选项，那么这个节点将始终为 true。

注意：该节点只能在片元着色器阶段使用。

1. 端口

名称	方向	类型	描述
Out	Output	Boolean	输出值

四十. 屏幕分辨率——Screen Node

用于访问屏幕分辨率。

1. 端口

名称	方向	类型	描述
Width	Output	Float	屏幕横向像素数
Height	Output	Float	屏幕纵向像素数

2. 生成的代码示例

```
float _Screen_Width = _ScreenParams.x;
float _Screen_Height = _ScreenParams.y;
```

四十一. 快捷键

Ctrl+C 复制

Ctrl+V 粘贴

Ctrl+D 直接创建副本

Ctrl+G 将选中内容编组
Delete 删除
Space 创建节点