UnityShader 编程手册

整理：瓦间草
联系：657917230@qq.com
日期：2017 年 9 月

未完待续

目录

0 可以做除数。shader 内部貌似做了排除。

各个输出通道的数据类型

输出通道	数据类型	描述
Albedo	fixed3	基本色
Normal	fixed3	tangent 空间法线
Emission	half3	自发光，其实用辐射度这个词更准确
Metallic	half	0 代表金属感弱，1 代表强
Smoothness	half	0 代表粗糙，1 代表镜面
Occlusion	half	遮蔽，默认是 1
Alpha	fixed	不透明度，0 透，1 不透

常用函数

若无说明，a，b，n 皆可为任意纬度数字数组类型。

常数篇

e 自然常数

unityshader 当中并无此常数，可以近似写为:`float 2.718281828459`

π 圆周率

可以近似写为:`float 3.141592654`

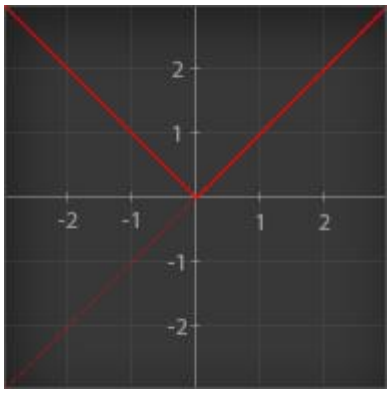
phi 黄金分割数

可以近似写为:`float 1.61803398875`

函数篇：

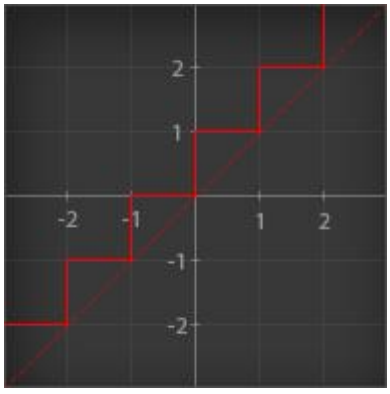
abs(n);　//绝对值函数

返回 n 的绝对值。



ceil(n);　//取整，超则上；

只要 n 大于(*.0)，则进一。




```
float3 emissive = ceil(_col.rgb);
```

clamp(n , a , b);　//限制函数

限制 n 取值范围在 a 和 b 之间。n < a 则 n = a。n > b 则 n = b 。

cross(vector3 a , vector3 b)

输出 a 和 b 的叉乘。本质上;它输出一个垂直于两个输入矢量的矢量

distance(a , b); //距离

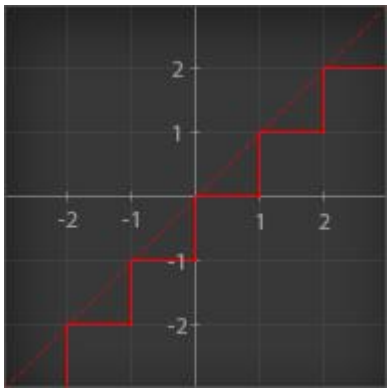
计算距离，即 a 与 b 之间的差的绝对值。

exp(n); //计算以 e 为底的 n 的对数

exp2(n); //计算以 2 为底的 n 的指数

floor(n); //取整，不足不上

输入 n，以整数（*.0）为基线四舍五入，结果返回整数。

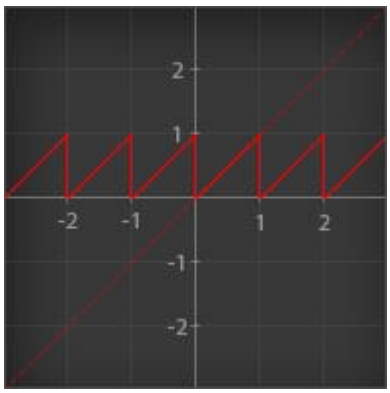


fmod(a , b); //余数

返回 a 除以 b 的余数

frac(n); //获取小数

返回 n 的小数部分。



lerp(a , b , n); //插值，按权重混合

非常常用的命令。n 为权重，取值范围[0,1]，越接近 0，则 a 的比重越大。越接近 1，则 b 的比重越大。a，b 可以为三维数组或贴图。
其实现混合原理是：(a + b) / 2 。
算法：

```
lerp( a , b , n ){
    n = saturate( n );
    col =   ( a * ( n – 1 ) + b * n ) / 2 ;
    return col;
}
```

log(n); //对数函数

计算以自然常数 e 为底，n 的对数值，返回 n 等于 e 的几次方。

log2(n); //对数函数

计算以 2 为底，n 的对数值，返回 n 等于 2 的几次方。若 2^a = n，则 a=log2(n)。

延伸知识点：

log(x * y) = log(x) + log(y)；
log(x / y) = log(x) - log(y)；

log10(n); //对数函数

计算以 10 为底，n 的对数值，返回 n 等于 10 的几次方。若 10^a = n，则 a=log10(n)。

max(a , b); //最大值

返回两者中的最大值；a，b 可以为任意数值。也可以将三维数组与一维、二维数组对比。

fixed finalcol = max(0.1,0.2); //得到结果为 0.2

fixed3 finalcol = max(max(_col1.rgb,_col1.g),_col1.b);
//得到结果相当于 max(max(_col1.rgb,_col1.ggg), _col1.bbb);
//图像 rgb 最亮的和最亮的 g 最亮的 b 比大小，留下来的部分。

fixed3 col = tex2D(_MainTex, i.uv);
fixed light= max(max(col.r,col.g),col.b;
fixed4 finalcol = fixed4(light,light,light,1);
//得到的是 MainTex 主贴图的亮度图。（ps：RGB 色彩的亮度就是 RGB 数值最大的那个数）

min(a , b); //最小值

返回两者中的最小值；a，b 可以为任意数值。也可以将三维数组与一维、二维数组对比。

normalize(n); //向量归一化

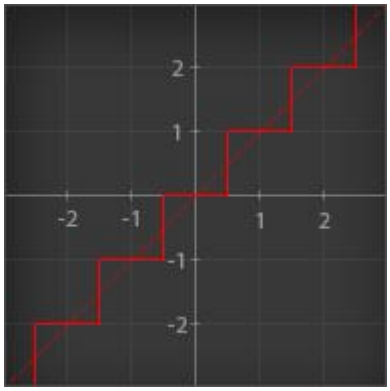
输出输入向量的规范化版本。本质上;将向量的长度设置为 1，同时保持相同的方向。

pow(a , b); //乘方函数

返回 a 的 b 次方。

round(n); //取整，四舍五入

满半（*.5）进一。



saturate(n); //防溢出函数

用于片元着色器，输入大于 1 的数则取 1，输入小于 0 的数则取 0。等价于 clamp(n,a,b);
程序解释：

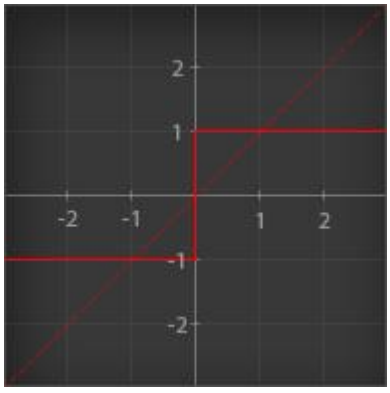
```
function saturate(a){
    if (a>1)
        a = 1
    if (a<0)
        a = 0
    return a
    }
```

Shader 举例：

```
fixed4 frag (v2f i) : SV_Target
{
      fixed4 col = tex2D(_MainTex, i.uv) + 0.5;
      fixed4 finalcol = saturate(col);
      return finalcol;
}
```

sign(n); //符号函数

输入任意数 n，如果 n 大于 0 返回 1；n 等于 0 返回 0；n 小于 0 返回-1。



smoothstep（ a ， b ， n ）//限制函数，平滑

和 lerp 类似，在 a（最小值）和 b（最大值）之间的插值，并在限制处渐入渐出。shader 当中渐入渐出效果并不明显。

sqrt(n); //开平方根

输出 n 的平方根。

step(a , b); //输出比较结果。

当 a>b 则输出 0，当 a<=b 则输出 1。

trunk(n); //截掉小数以取整

实用代码片段

通过色相，纯度，明度，控制颜色。

设_H，_S，_V 分别为 fixed 型数值，取值范围在（0，1）。

```
float3 emissive = (lerp(float3(1,1,1),saturate(3.0*abs(1.0-
2.0*frac(_h+float3(0.0,-1.0/3.0,1.0/3.0)))-1),_s)*_v);
```

通过一维数组控制色相变化

```
float3 emissive = saturate(3.0*abs(1.0-
2.0*frac(_s+float3(0.0,-1.0/3.0,1.0/3.0)))-1);
```

噪波函数

```
float2 noise_skew = i.uv0 + 0.2127+i.uv0.x*0.3713*i.uv0.y;
float2 noise_rnd = 4.789*sin(489.123*(noise_skew));
float noise = frac(noise_rnd.x*noise_rnd.y*(1+noise_skew.x));
```

限制色阶

finalcol 输出颜色，_col 输入颜色，_n 色阶数量。
这个限制会导致边缘非常的硬，如果软化就不要使用 floor 函数，需要单独处理。

```
  float finalcol = floor(_col * _n) / (_n - 1);
```

重新映射

将原先的色彩范围映射给新的色彩范围。
_col 原始图像； _in_min 输入最小值；_in_max 输入最大值；_out_min 输出最小值；_out_max 输出最大值；

```
 float3 emissive = (_out_min + ( (_col.rgb - _in_min) * (_out
_max - _out_min) ) / (_in_max - _in_min));
```

获取色相，明度，纯度

```
float4 RGB_HSV_p = lerp(float4(float4(_col.rgb,0.0).zy, RGB_H
SV_k.wz), float4(float4(_col.rgb,0.0).yz, RGB_HSV_k.xy), step
(float4(_col.rgb,0.0).z, float4(_col.rgb,0.0).y));
float4 RGB_HSV_q = lerp(float4(RGB_HSV_p.xyw, float4(_col.rgb
,0.0).x), float4(float4(_col.rgb,0.0).x, RGB_HSV_p.yzx), step
(RGB_HSV_p.x, float4(_col.rgb,0.0).x));
float RGB_HSV_d = RGB_HSV_q.x - min(RGB_HSV_q.w, RGB_HSV_q.y)
;
float RGB_HSV_e = 1.0e-10;
float3 RGB_HSV = float3(abs(RGB_HSV_q.z + (RGB_HSV_q.w - RGB_
HSV_q.y) / (6.0 * RGB_HSV_d + RGB_HSV_e)), RGB_HSV_d / (RGB_H
SV_q.x + RGB_HSV_e), RGB_HSV_q.x);;
```

减小饱和度

输入颜色_col，控制_val 取值范围[-1 , 1]，

```
float3 emissive = lerp(_col.rgb,dot(_col.rgb,float3(0.3,0.59,
```

```
0.11)),_val);
```