唐老狮系列教程

利用深度法线纹理实现边缘检测效果的基本原理

WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY



唐老狮系列教程-深度法线纹理边缘检测基本原理

主要讲解内容

WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY



唐老狮系列教程-深度法线纹理边缘检测基本原理

主要讲解内容

1. 知识回顾 边缘检测屏幕期处理效果
2. 为什么要基于深度+法线纹理实现边缘检测
3. 利用深度+法线纹理实现 边缘检测的基本原理

WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY



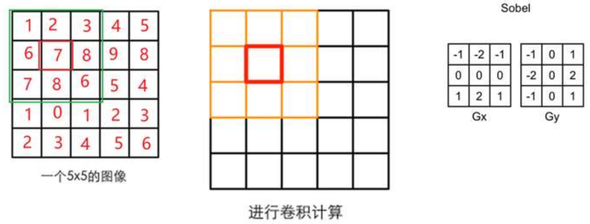
唐老狮系列教程-深度法线纹理边缘检测基本原理

知识回顾 边缘检测屏幕后处理效果

WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY

唐老狮系列教程-深度法线纹理边缘检测基本原理

边缘检测屏幕后处理效果



1. 边缘检测效果是什么

   是一种用于突出图像中的边缘，使物体的轮廓更加明显的图像处理技术

   利用 Shader 代码自动给屏幕图像进行描边处理

2. 边缘检测效果的基本原理

   2-1 得到 当前像素以及其 上下左右、左上左下、右上右下共9个像素的灰度值

   2-2 用这9个灰度值和 Sobel算子 进行卷积计算得到梯度值 G = abs(Gx) + abs(Gy)

   2-3 最终颜色 = lerp（原始颜色，描边颜色，梯度值）

3. 如何得到当前像素周围8个像素位置

   利用 float4 纹理名_TexelSize 纹素 信息得到当前像素周围8个像素位置

WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY



唐老狮系列教程-深度法线纹理边缘检测基本原理

为什么要基于深度法线纹理实现边缘检测

WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY

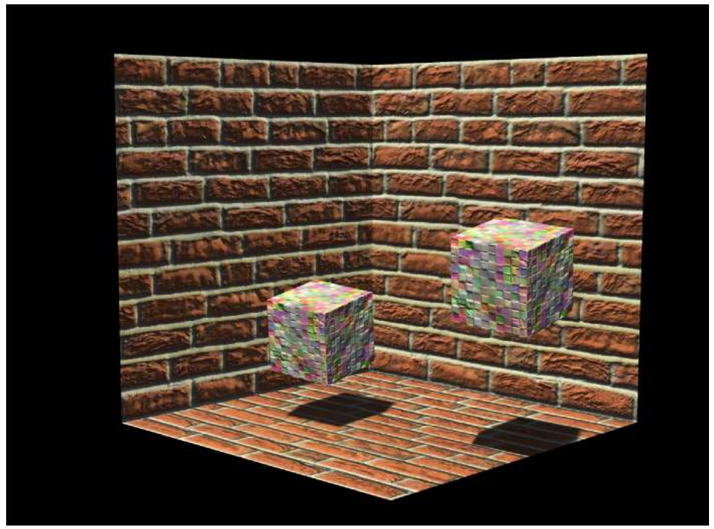
唐老狮系列教程-深度法线纹理边缘检测基本原理

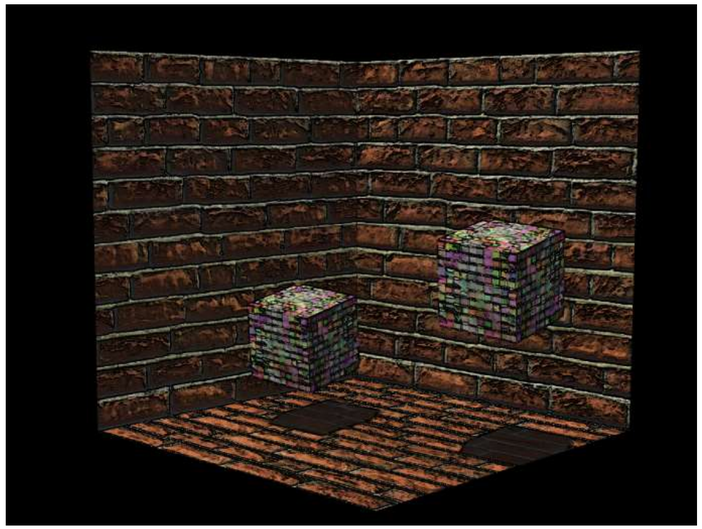
为什么要基于深度法线纹理实现边缘检测

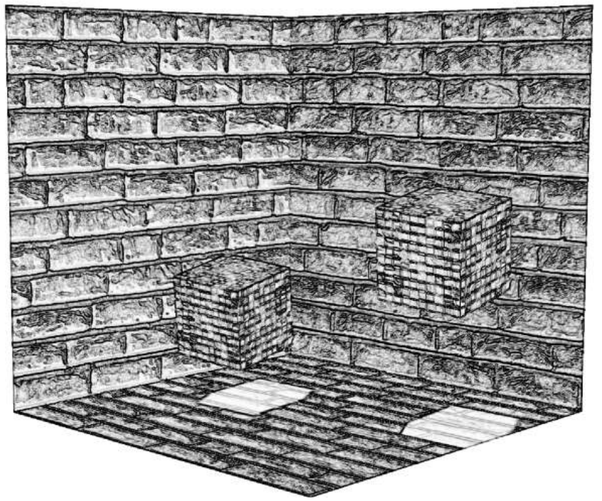
我们之前**利用Sobel算子基于像素灰度值计算出来的边缘检测效果**，其实总体上来说不太理想。

因为这种**计算方式依赖于像素的颜色(灰度值)变化来识别边缘，会受到物体纹理和阴影颜色等因素影响。**

所以这种制作方式往往不能准确的反应出物体的真实轮廓！







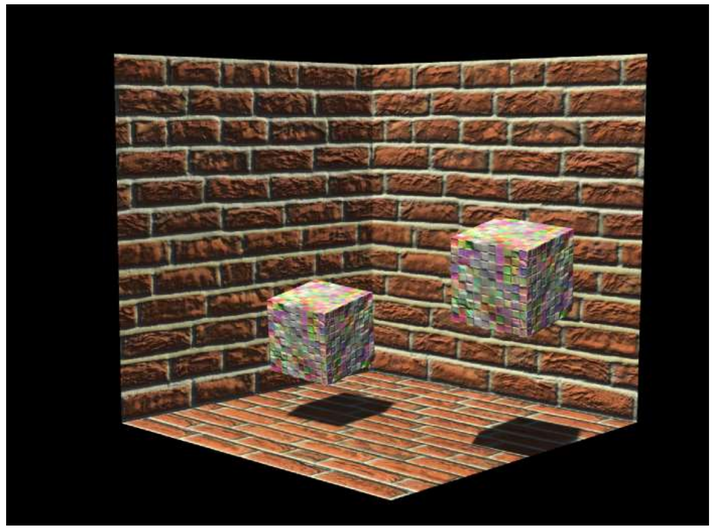
WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY

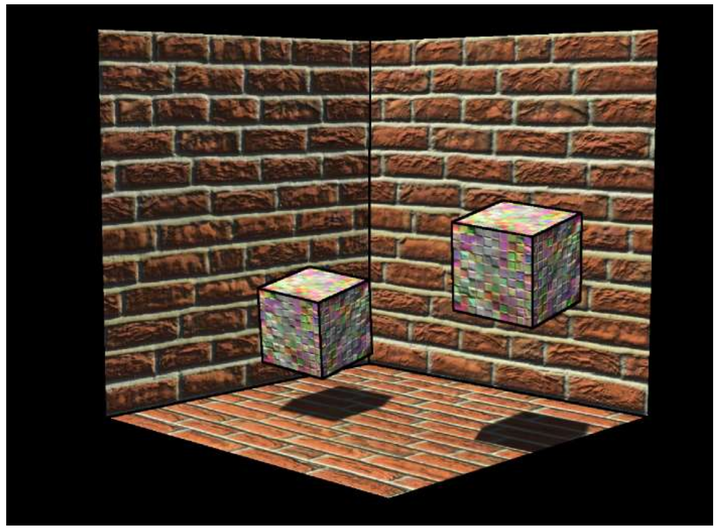
唐老狮系列教程-深度法线纹理边缘检测基本原理

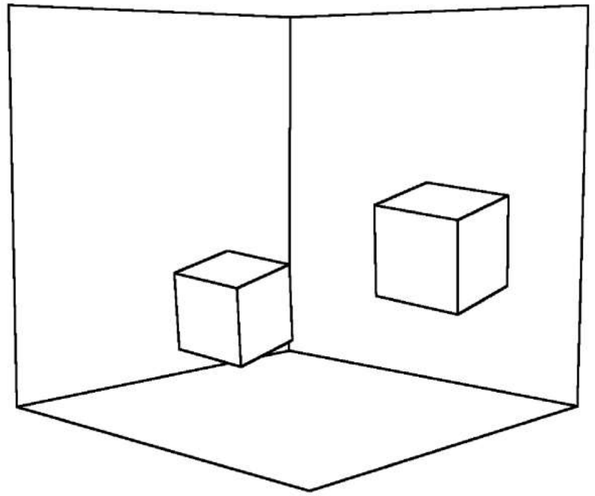
为什么要基于深度法线纹理实现边缘检测

因此我们才会来学习**基于深度+法线纹理来实现边缘检测屏幕后期处理效果**，

通过这种方式实现的边缘检测，**不会受纹理和光照的影响，只会根据渲染物体的模型信息（深度、法线）来进行判断检测，这样检测出来的边缘更加的可靠！**







WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY

唐老狮系列教程-深度法线纹理边缘检测基本原理

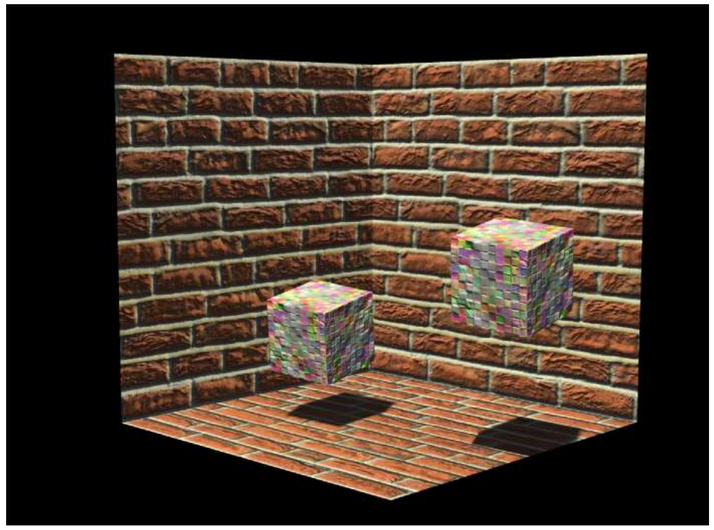
为什么要基于深度法线纹理实现边缘检测

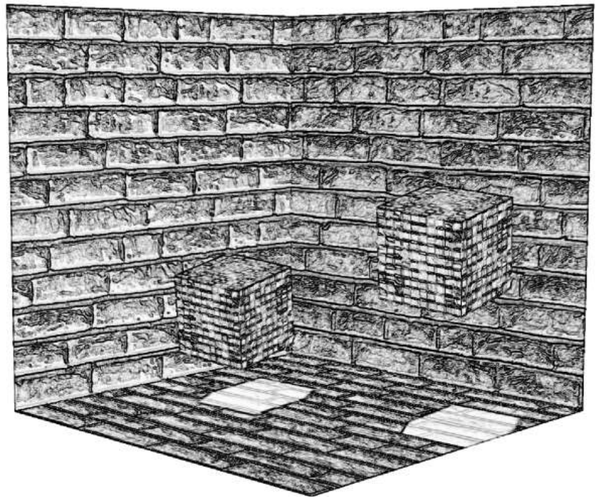
注意：

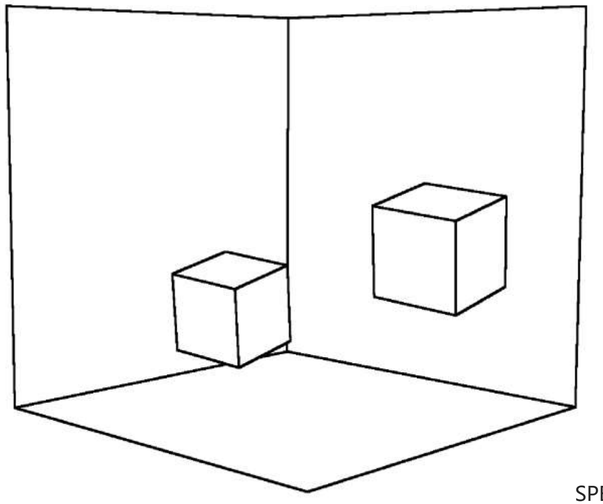
对于2D图片，基于灰度值的边缘检测更合适

对于3D场景，基于深度+法线纹理的边缘检测更合适

主要原因是2D图片中的深度和法线信息往往是一致的，不存在差异性







WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY



唐老狮系列教程-深度法线纹理边缘检测基本原理

利用深度+法线纹理实现边缘检测的基本原理

WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY

唐老狮系列教程-深度法线纹理边缘检测基本原理

利用深度+法线纹理实现边缘检测的基本原理

一句话概括它的基本原理：

基于Roberts(罗伯茨)交叉算子，通过比较对角线上的的像素的深度和法线值，判断是否在边缘上。

关键点：

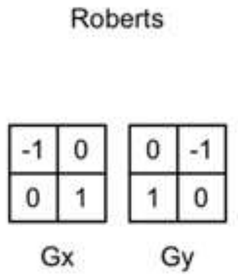
1.如何得到对角线上的像素

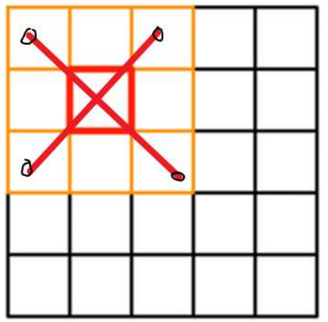
2.如何进行深度和法线值的比较

3.如何决定是否在边缘

注意点：

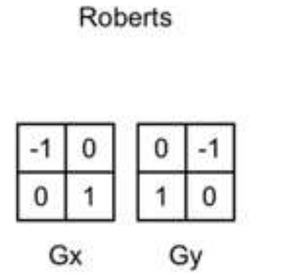
不会进行卷积运算





唐老狮系列教程-深度法线纹理边缘检测基本原理

利用深度+法线纹理实现边缘检测的基本原理

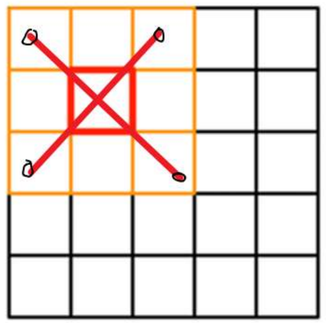


1.如何得到对角线上的像素

在顶点着色器中，利用纹素进行uv坐标偏移计算

并且我们可以添加一个可控的 采样偏移距离变量 _SampleDistance

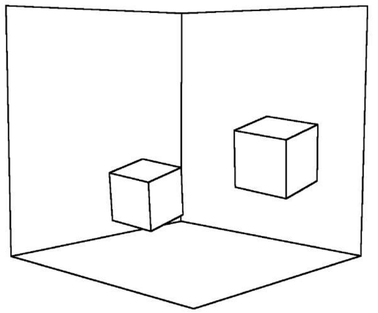
它可以用来决定描边的粗细，值越大描边越粗



原理是

采样离中心像素越近，检测的变化更细微，深度和法线值变化小，边缘会较细

采样离中心像素越远，检测的范围较大，深度和法线值变化大，边缘会较粗



```
half2 uv = v.texcoord;
o.uv[0] = uv;

o.uv[1] = uv + _MainTex_TexelSize.xy * half2(1,1) * _SampleDistance;
o.uv[2] = uv + _MainTex_TexelSize.xy * half2(-1,-1) * _SampleDistance;
o.uv[3] = uv + _MainTex_TexelSize.xy * half2(-1,1) * _SampleDistance;
o.uv[4] = uv + _MainTex_TexelSize.xy * half2(1,-1) * _SampleDistance;
```

WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY

唐老狮系列教程-深度法线纹理边缘检测基本原理

利用深度+法线纹理实现边缘检测的基本原理

2.如何进行深度和法线的比较

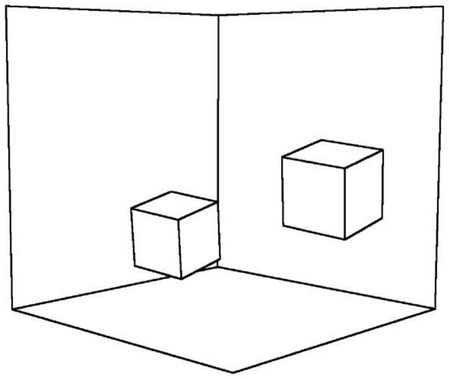
在片元着色器中，利用顶点着色器中得到的uv坐标，

在深度+法线纹理中进行采样，得到深度值和法线值。

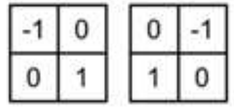
再求出对角线上对角的两个像素的 深度值差 和 法线值差。

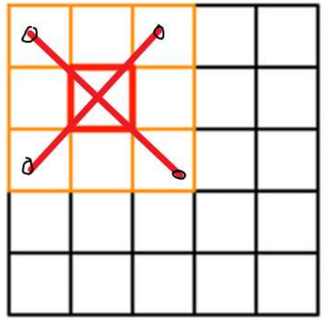
如果其中一个的差值大于了自定义的阈值

那么我们认为该像素点在物体的边缘上，否则不是边缘



Roberts

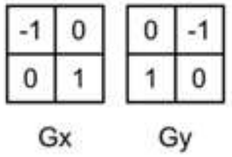




唐老狮系列教程-深度法线纹理边缘检测基本原理

利用深度+法线纹理实现边缘检测的基本原理

Roberts



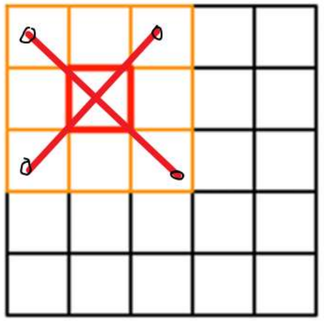
3.如何决定是否在边缘上

根据刚才进行的深度和法线值的比较，

只要其中之一（深度或者法线）满足了大于自定义阈值的条件，

那么该像素就在边缘上，就使用描边颜色进行颜色处理

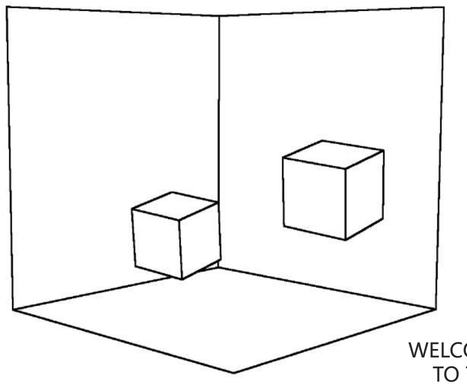
关键计算规则如下



```
half CheckSame(half4 point1, half4 point2) {
    //得到点一点儿的法线xy(不需要解码)和深度值
    half2 point1Normal = point1.xy;
    float point1Depth = DecodeFloatRG(point1.zw);
    half2 point2Normal = point2.xy;
    float point2Depth = DecodeFloatRG(point2.zw);

    //法线差异计算
    half2 diffNormal = abs(point1Normal - point2Normal) * 自定义法线敏感度变量;
    //是否在同一法线区间
    int isSameNormal = (diffNormal.x + diffNormal.y) < 0.1;
    // 深度差异计算
    float diffDepth = abs(point1Depth - point2Depth) * 自定义深度敏感度变量;
    // 是否在同一深度区间
    int isSameDepth = diffDepth < 0.1 * point1Depth;

    // 返回值
    // 1 - 法线和深度基本相似
    // 0 - 不相似 证明中心像素在边缘
    return isSameNormal * isSameDepth ? 1.0 : 0.0;
}
```



WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY

唐老狮系列教程-深度法线纹理边缘检测基本原理

利用深度+法线纹理实现边缘检测的基本原理

一句话概括它的基本原理：

基于Roberts(罗伯茨)交叉算子，通过比较对角线上的的像素的深度和法线值，判断是否在边缘上。

关键点：

1.得到对角线上的像素

2.进行深度和法线值的比较

3.决定是否在边缘

注意点：

不会进行卷积运算

```
half2 uv = v.texcoord;
o.uv[0] = uv;

o.uv[1] = uv + _MainTex_TexelSize.xy * half2(1,1) * _SampleDistance;
o.uv[2] = uv + _MainTex_TexelSize.xy * half2(-1,-1) * _SampleDistance;
o.uv[3] = uv + _MainTex_TexelSize.xy * half2(-1,1) * _SampleDistance;
o.uv[4] = uv + _MainTex_TexelSize.xy * half2(1,-1) * _SampleDistance;
```

```
half CheckSame(half4 point1, half4 point2) {
    //得到点一点儿的法线xy(不需要解码)和深度值
    half2 point1Normal = point1.xy;
    float point1Depth = DecodeFloatRG(point1.zw);
    half2 point2Normal = point2.xy;
    float point2Depth = DecodeFloatRG(point2.zw);

    //法线差异计算
    half2 diffNormal = abs(point1Normal - point2Normal) * 自定义法线敏感度变量;
    //是否在同一法线区间
    int isSameNormal = (diffNormal.x + diffNormal.y) < 0.1;
    // 深度差异计算
    float diffDepth = abs(point1Depth - point2Depth) * 自定义深度敏感度变量;
    // 是否在同一深度区间
    int isSameDepth = diffDepth < 0.1 * point1Depth;

    // 返回值
    // 1 - 法线和深度基本相似
    // 0 - 不相似 证明中心像素在边缘
    return isSameNormal * isSameDepth ? 1.0 : 0.0;
}
```

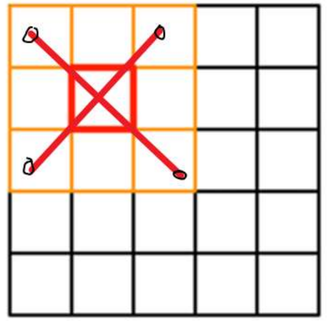
Roberts

-1	0
0	1

Gx

0	-1
1	0

Gy



WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY



唐老狮系列教程-深度法线纹理边缘检测基本原理

总结

WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY



唐老狮系列教程-深度法线纹理边缘检测基本原理

总结

1. 知识回顾 边缘检测屏幕后处理效果

   是一种用于突出图像中的边缘，使物体的轮廓更加明显的图像处理技术

   利用 Shader 代码自动给屏幕图像进行描边处理，以前我们是基于Sobel算子

   利用中心像素周围像素的灰度值参与卷积运算来决定边缘的

2. 为什么要基于深度+法线纹理实现边缘检测

   以前的方式主要用来处理2D图像，并且得到的效果也很一般

   为了得到更好的效果所以在3D场景中我们通过结合

   深度+法线纹理来实现边缘检测屏幕后期处理效果

唐老狮系列教程-深度法线纹理边缘检测基本原理

总结

3. 利用深度+法线纹理实现 边缘检测的基本原理

基于Roberts(罗伯茨)交叉算子，通过比较对角线上的的像素的深度和法线值，判断是否在边缘上。

关键点：

1. 得到对角线上的像素

2. 进行深度和法线值的比较

3. 决定是否在边缘

注意点：

不会进行卷积运算

```
half2 uv = v.texcoord;
o.uv[0] = uv;

o.uv[1] = uv + _MainTex_TexelSize.xy * half2(1,1) * _SampleDistance;
o.uv[2] = uv + _MainTex_TexelSize.xy * half2(-1,-1) * _SampleDistance;
o.uv[3] = uv + _MainTex_TexelSize.xy * half2(-1,1) * _SampleDistance;
o.uv[4] = uv + _MainTex_TexelSize.xy * half2(1,-1) * _SampleDistance;
```

```
half CheckSame(half4 point1, half4 point2) {
    //得到点一点儿的法线xy(不需要解码)和深度值
    half2 point1Normal = point1.xy;
    float point1Depth = DecodeFloatRG(point1.zw);
    half2 point2Normal = point2.xy;
    float point2Depth = DecodeFloatRG(point2.zw);

    //法线差异计算
    half2 diffNormal = abs(point1Normal - point2Normal) * 自定义法线敏感度变量;
    //是否在同一法线区间
    int isSameNormal = (diffNormal.x + diffNormal.y) < 0.1;
    // 深度差异计算
    float diffDepth = abs(point1Depth - point2Depth) * 自定义深度敏感度变量;
    // 是否在同一深度区间
    int isSameDepth = diffDepth < 0.1 * point1Depth;

    // 返回值
    // 1 - 法线和深度基本相似
    // 0 - 不相似 证明中心像素在边缘
    return isSameNormal * isSameDepth ? 1.0 : 0.0;
}
```

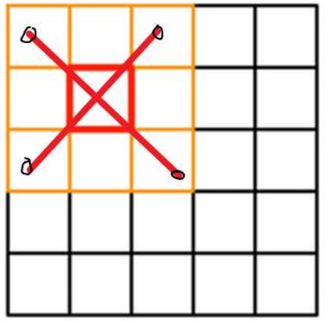
Roberts

-1	0
0	1

Gx

0	-1
1	0

Gy



WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY



唐老狮系列教程

Thank

谢谢您的聆听

WELCOME
TO THE
UNITY
SPECIALTY COURSE
STUDY