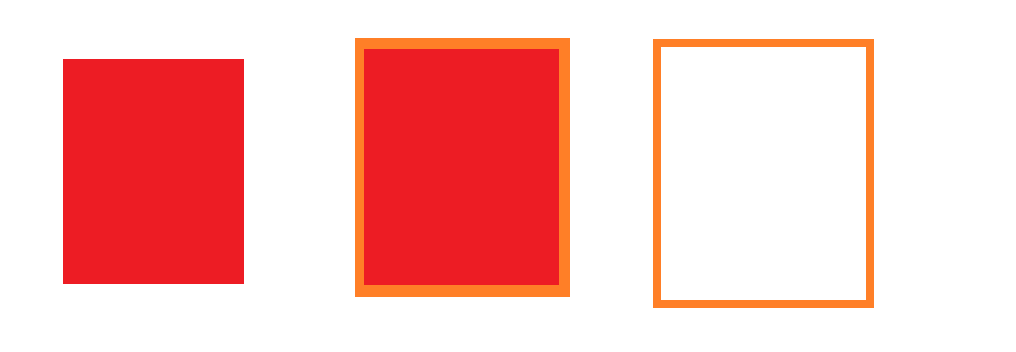
# **后期处理实现高亮描边**

作者：韩宇飞

链接：http://gad.qq.com/article/detail/287441

因为项目需要希望实现gao'liang'b高亮描边的效果，一开始先尝试了火炬之光使用的光照的高亮描边。就是计算法线和Camera的夹角来计算光照，越靠近边缘的地方法线和Camera夹角越大顶点越亮。但是感觉效果不好，正好当时星际2发布了，里面的后期处理描边效果非常好，原理也不复杂因此也在项目中实现了一下。

原理很简单，在RT上渲染出人物的片，然后通过高斯模糊进行模糊和扩散，然后再通过模板挖空留下边的效果，最后在和正常渲染结果的RT叠加。原理如下：



**暗黑3以及轩辕的后处理描边效果：**





**高亮描边流程：**

1．在场景物件的渲染阶段，渲染并写入高亮物体的模版值到高亮效果的输入RT上。

1. 首先在渲染角色前，打开后期处理高亮描边并将高亮物体设置不渲染，这么做是因为写模版值时需要考虑深度，如果被其他人物遮挡的部分则不能写入模版值

1. 然后在角色物体全部渲染完后，，添加模版写入属性（CStencilPropertySwaper）并渲染高亮物体，其中特效和半透明物件不加模版写入属性不描边。伪代码如下：

m\_pkStencilProperty->SetStencilOn(TRUE);

m\_pkStencilProperty->SetStencilFunction(TEST\_ALWAYS);

m\_pkStencilProperty->SetStencilPassAction(ACTION\_REPLACE);

m\_pkStencilProperty->SetStencilFailAction(ACTION\_KEEP);

m\_pkStencilProperty->SetStencilPassZFailAction(ACTION\_KEEP);

m\_pkStencilProperty->SetStencilReference(0x1);

1. 最后还原高亮物体的模版属性。

2．在后期处理阶段，拷贝模版缓冲，并完成后期处理描边，通过宏DEVIATION可以调整扩散范围，也就是粗细，越小越细，MULTIPLITER可以调整模糊程度，越小越不模糊

1）复制深度模版缓冲，为了使用抗拒齿渲染场景时必须使用默认RT，所以要拷贝默认的模版缓冲

pktDefaultDepthStencilBuffer->Copy(m\_pkTargets[RRT\_FULL\_1]->GetRenderTargetGroup()->GetDepthStencilBuffer(), NULL, NULL, COPY\_FILTER\_LINEAR);

2）使用输入RT的深度模版缓冲绘制描边色RT，渲染到RRT\_FULL\_1（正常大小），结果就是人物形状的片。伪代码如下：

StencilEnable = true;

StencilFunc = equal;

StencilRef = 0x1;

StencilPass = keep;

//-----------------------------------------------------------------------------

// Pixel Shader: PS\_HighLightObject

// Desc: 填充高亮颜色.

//-----------------------------------------------------------------------------

float4 PS\_HighLightObject() : COLOR

｛

    return float4(0.7f, 0.7f, 0.5f, 0.0f);

｝

3）降采样，渲染到RRT\_DOWN\_0，主要是为了优化，对1/4大小RT模糊。伪代码如下：

//-----------------------------------------------------------------------------

// Pixel Shader: PS\_DownFilter

// Desc: Perform a high-pass filter and on the source texture and scale down.

//-----------------------------------------------------------------------------

float4 PS\_DownFilter(in float2 \_f2Tex : TEXCOORD0) : COLOR0

｛

    float4 f4Color = 0;

    for (int i = 0; i < 16; i++)

  ｛

      f4Color += tex2D(g\_BaseTexutre, \_f2Tex + TexelCoordsDownFilter[i].xy);

  ｝

  return f4Color / 16;

｝

4）混合采样16遍第张贴图（横向高斯模糊），使用模版测试渲染到RRT\_DOWN\_1，结果就是挖掉人物部分只剩轮廓边的贴图。伪代码如下：

//-----------------------------------------------------------------------------

// Pixel Shader: Bloom

// Desc: Blur the source image along one axis using a gaussian

//       distribution. Since gaussian blurs are separable, this shader is

//       called twice; first along the horizontal axis, then along the

//       vertical axis.

//-----------------------------------------------------------------------------

float4 Bloom(in float2 kScreenPosition : TEXCOORD0) : COLOR

｛

    float4 kSample = 0.0f;

    float4 kColor = 0.0f;

    float2 kSamplePosition;

    // Perform a one-directional gaussian blur

    for (int iSample = 0; iSample < 15; iSample++)

    ｛

        kSamplePosition = kScreenPosition + gakSampleOffsets[iSample];

        kColor = tex2D(BasePointClampSampler, kSamplePosition);

        kSample += gakSampleWeights[iSample] \* kColor;

    ｝

    return kSample;

｝

5）混合采样16遍第张贴图（横向高斯模糊），使用模版测试渲染回RRT\_DOWN\_0，结果就是挖掉人物部分只剩轮廓边的贴图

6）升采样，渲染到RRT\_FULL\_1，还原正常大小并挖去中间轮廓只剩下边。伪代码如下：

StencilEnable = true;

StencilFunc = notequal;

StencilRef = 0x1;

StencilPass = keep;

//-----------------------------------------------------------------------------

// Pixel Shader: PS\_UpFilter

// Desc: Perform a high-pass filter and on the source texture and scale down.

//-----------------------------------------------------------------------------

float4 PS\_UpFilter(in float2 \_f2Tex : TEXCOORD0) : COLOR

｛

    float4 f4Color =  tex2D(g\_BaseTexutre, \_f2Tex);

    return f4Color;

｝

7）最后采样第0张贴图（后期处理的输入，就是最终渲染结果）和采样第1张贴图（边），然后混合相加，渲染到最终RenderTarget，就是带边的场景了。伪代码如下：

//-----------------------------------------------------------------------------

// Pixel Shader: PS\_HighLightBlend

// Desc: 在最终渲染结果上添加描边.

//-----------------------------------------------------------------------------

float4 PS\_HighLightBlend(in float2 \_f2Tex : TEXCOORD0) : COLOR

｛

    float4 f4Color1 = tex2D(g\_BaseTexutre, \_f2Tex);

    float4 f4Color2 = tex2D(g\_ShaderTexure, \_f2Tex);

    return f4Color1 + f4Color2;

｝



**问题：**

虽然功能很快实现了，但是真正应用的时候还是花了一些时间优化和遇到很多问题。

开启抗锯齿之后深度缓冲不能共享导致的问题。通过使用StretchRect来拷贝DepthStencilBuffer，来解决开了AA之后描边的问题。伪代码如下：

LPDIRECT3DSURFACE9 lpBackSurfaceSx9 = NULL;

HRESULT hr = lpDevice9->GetDepthStencilSurface(&lpBackSurfaceSx9);

lpDevice9->StretchRect( lpBackSurfaceSx9, NULL, ((NiDX92DBufferData\*)m\_spDepthStencilBuffer->GetRendererData())->GetSurface(), NULL, D3DTEXF\_LINEAR );

实现了多种颜色描边，并解决了多种颜色描边在不同电脑上显示不连续等bug

偶现描边导致屏幕变白的BUG。因为在没有描边物体时跳过描边物体渲染流程，但这个判断加在了判断是否打开描边后期处理的流程之前。所以虽然关闭了描边物体的渲染流程但是没有关闭描边的后期处理流程。当游戏正好在描边状态时，系统自动降低配置关闭描边效果后，会导致后期处理无限循环扩大，也就是边界无限扩大。通过在判断没有描边物体中断描边物体的渲染流程时同时也关闭描边的后期处理流程解决

描边闪烁。因为是beginrendertarget放到了BeginOffscreen之前了,应该放在之间

Alphatest的问题,这是因为xx渲染颜色时写入模版值的，这个时候因为要用alpha做暗边，所以没有考虑alphatest。最终的方案是先话正常物件,再画高亮物件同时写模版值,最后再画一遍写颜色值这时不通过深度比较而是通过模版值比较，因为多重采样的情况下因为z精度的问题边会乱

在设置模版采样之后必须要还原渲染状态。否则自己写的Shader会一直开启Stencil

因为在copy深度模版缓冲之后清空了Zbuf，在有的电脑上会连Stencil一起清掉。最终解决的办法是在去掉情况zbuf，在渲染不通描边物体颜色时不使用copy不精确的z（多重采样zcopy会不精确，关掉画颜色时ztest和zwrite）而是使用模版比较（模版值没有小数问题）

解决了描边时设备丢失崩溃的问题。这是因为设备丢失时copy深度buff导致，通过在设备丢失时关闭后期处理，在设备丢失好以后打开后期处理解决