<https://blog.csdn.net/puppet_master/article/details/54000951>

描边效果有几种实现方式，其实边缘光效果与描边效果有些类似，适当调整边缘光效果，其实也可以达到凸显要表达的对象的意思，边缘光的实现最为简单，只是在计算的时候增加了一次计算法线方向与视线方向的夹角计算，用1减去结果作为系数乘以一个边缘光颜色就达到了边缘光的效果，是性能最好的一种方法。

边缘光效果虽然简单，但是有很大的局限性，边缘光效果只是在当前模型本身的光照计算时调整了边缘位置的颜色值，并没有达到真正的“描边”，而我们需要的描边效果，一般都是在正常模型的渲染状态下，在模型外面扩展出一个描边的效果，既然要让模型的形状有所改变（向外扩一点），那么肯定就和vertex shader有关系了。而我们的描边效果，肯定就是要让模型更“胖”一点，能够把我们原来的大小包裹住；微观一点来看，一个面，如果我们让它向外扩展，也就是这个面的法线所指向的方向，那么就让这个面朝着法线的方向平移一点，再围观一点来看，对于顶点来说，也就是我们的vertex shader真正要写的内容了，我们正常计算顶点的时候，传入vertex会经过mvp变换，最终传给fragment shader，那么我们就可以在这一步让顶点沿着法线的方向稍微平移一些。

要实现卡通渲染有很多方法，其中之一就是使用**基于色调的着色技术**。在实现中，我们往往会使用漫反射系数对一张一维纹理进行采样，以控制漫反射的色调。卡通风格的高光效果也和我们之前的光照不同，在卡通风格中，模型的高光往往是一块块分界明显的纯色区域。除了光照模型不同外，卡通风格通常还需要在物体边缘部分绘制轮廓，在前面中我们介绍使用屏幕后处理技术对屏幕图像进行描边，在这里，我们将会介绍基于模型的描边方法，这种实现方法更加简单，而且在很多情况下也能得到不错的效果。

Shader "Unlit/ToonShader"

{

Properties

{

\_Color ("Color Tint", Color) = (1, 1, 1, 1)

\_MainTex ("Texture", 2D) = "white" {}

// 漫反射色调的渐变纹理

\_Ramp ("Ramp Texture", 2D) = "white" {}

// 轮廓线宽度

\_Outline ("Outline", Range(0, 1)) = 0.1

// 轮廓线颜色

\_OutlineColor ("Outline Color", Color) = (0, 0, 0, 1)

// 高光反射颜色

\_Specular ("Specular", Color) = (1, 1, 1, 1)

// 高光反射使用的阈值

\_SpecularScale ("Specular Scale", Range(0, 0.1)) = 0.01

}

SubShader

{

Name "OutLine"

Pass

{

Cull Front

CGPROGRAM

#pragma vertex vert

#pragma fragment frag

#include "UnityCG.cginc"

struct v2f

{

float4 pos : SV\_POSITION;

float2 uv : TEXCOORD0;

};

float \_Outline;

fixed4 \_OutlineColor;

v2f vert(appdata\_base v)

{

v2f o;

float4 pos = mul(UNITY\_MATRIX\_MV, v.vertex);

float3 normal = mul((float3x3)UNITY\_MATRIX\_IT\_MV, v.normal);

// 尽可能让背面扁平化，避免凹多边形背面扩张后的顶点挡住正面的面片

normal.z = -0.5;

pos = pos + float4(normalize(normal), 0) \* \_Outline;

o.pos = mul(UNITY\_MATRIX\_P, pos);

return o;

}

fixed4 frag(v2f i) : SV\_Target

{

return fixed4(\_OutlineColor.rgb, 1.0);

}

ENDCG

}

Pass

{

Tags { "LightMode" = "ForwardBase" }

Cull Back

CGPROGRAM

#pragma vertex vert

#pragma fragment frag

#pragma multi\_compile\_fwdbase

#include "UnityCG.cginc"

#include "AutoLight.cginc"

#include "Lighting.cginc"

struct v2f

{

float4 pos : SV\_POSITION;

float2 uv : TEXCOORD0;

float3 worldNormal : TEXCOORD1;

float3 worldPos : TEXCOORD2;

SHADOW\_COORDS(3)

};

sampler2D \_MainTex;

float4 \_MainTex\_ST;

sampler2D \_Ramp;

float \_Outline;

fixed4 \_OutlineColor;

fixed4 \_Color;

fixed4 \_Specular;

float \_SpecularScale;

v2f vert(appdata\_base v)

{

v2f o;

UNITY\_INITIALIZE\_OUTPUT(v2f, o);

o.pos = UnityObjectToClipPos(v.vertex);

o.uv = TRANSFORM\_TEX(v.texcoord, \_MainTex);

o.worldNormal = UnityObjectToWorldNormal(v.normal);

o.worldPos = mul(unity\_ObjectToWorld, v.vertex).xyz;

TRANSFER\_SHADOW(o);

return o;

}

fixed4 frag(v2f i) : SV\_Target

{

fixed3 worldNormal = normalize(i.worldNormal);

fixed3 worldLightDir = normalize(UnityWorldSpaceLightDir(i.worldPos));

fixed3 worldViewDir = normalize(UnityWorldSpaceViewDir(i.worldPos));

fixed3 worldHalfDir = normalize(worldLightDir + worldViewDir);

fixed4 c = tex2D(\_MainTex, i.uv);

fixed3 albedo = c.rgb \* \_Color.rgb;

fixed3 ambient = UNITY\_LIGHTMODEL\_AMBIENT.xyz \* albedo;

UNITY\_LIGHT\_ATTENUATION(atten, i, i.worldPos);

fixed diff = dot(worldNormal, worldLightDir);

diff = (diff \* 0.5 + 0.5) \* atten;

fixed3 diffuse = \_LightColor0.rgb \* albedo \* diff; // tex2D(\_Ramp, float2(diff, diff)).rgb;

fixed spec = dot(worldNormal, worldHalfDir);

fixed3 w = fwidth(spec);

fixed3 specular = \_Specular.rgb \* lerp(0, 1, smoothstep(-w, w, spec + \_SpecularScale - 1)) \* step(0.0001, \_SpecularScale);

return fixed4(ambient + diffuse + specular, 1.0);

}

ENDCG

}

}

FallBack "Diffuse"

}

