# LEARN THE FUNDAMENTALS OF SHADER WRITING IN UNITY IN THIS COMPLETE TUTORIAL SERIES.

In this Unity tutorial series we will be giving you a basic introduction to shader scripting, showing you how to write your own vertex/fragment shaders from scratch in unity.
I have learnt the hard way how to write shaders through books and experimentation, trial and error. The goal of this series is to take everything I have learned and structure it into easy to follow, ordered lessons that work in unity 4, with clear instructions on not only how to write clean and efficient code, but also how to debug and optimize your shaders. Effectively this is the series I wish I had when I started learning CG.

In this beginner section we will be covering how to deal with colors and textures, how to write complete lighting models from scratch and even delve into normal mapping.

This text tutorial is accompanied by a complete video. I recommend watching the video and referring to the text as reference.

Be sure to read the tutorial carefully, and post any questions you may have if you get stuck, as there is a lot to cover. I have referenced some material from the following books throughout this series:
http://en.wikibooks.org/wiki/Cg_Programming/Unity
http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter01.html
http://http.developer.nvidia.com/GPUGems/gpugems_pref02.html
http://http.developer.nvidia.com/GPUGems2/gpugems2_frontmatter.html
http://http.developer.nvidia.com/GPUGems3/gpugems3_pref02.html

I certainly recommend reading those books as they are fantastic and cover a lot more than what you will find here, what I have done is broken everything down into easy to follow, ordered lessons that will work in unity 4.

NOTE: This tutorial requires very basic knowledge of scripting such as variables and functions. If you have no scripting experience I recommend you first watch our Introduction to Scripting tutorial in the getting started section.

## PART 6 - TEXTURE MAPS

Our shader is cool and all, but to make it actually useful, we need to hook up some texture maps. In this lesson we will take our full rim lighting shader and add a diffuse texture map to it. Do note that we will no longer be using multiple lights to keep the code nice and short, I do encourage you to add it in however.

## HOW DOES IT WORK?

It's pretty straightforward actually, we need to first add in an image property so we can load in the image, next we sample it to bring it into a workable format, and line it up with the texture coordinates. Finally we multiply the color by the lighting and hey presto, a textured shader.
Let's begin.

## THE CODE

Let's start with the property:

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

We use the 2D property with a default color of white so we don't have to load one in. We also make sure we call it _MainTex, it's not important now, but if we want to include some more advanced built in features later on unity expects it to be called _MainTex.
Next we need to sample the texture:

```
uniform sampler2D _MainTex;
uniform float4 _MainTex_ST;
```

So this is new, the sampler2D will convert the texture into something we can work with. We also need to grab the tiling

and offset coordinates (the options next to the image in the inspector), we do this by creating another float4 in this way:

```
float4 [_TextureName]_ST;
```

Unity will automatically grab the coordinates for that texture for you.
Now what? Well we want to grab the uv map and bring them from the vertex data through the vertex program and into the fragment program. The uv maps are stored in TEXCOORD0 and TEXCOORD1.

Let's go ahead and take the first UV map through our structs.

## THE STRUCTS

```
struct vertexInput{
    float4 vertex : POSITION;
    float3 normal : NORMAL;
    float4 texcoord : TEXCOORD0;
};
struct vertexOutput {
    float4 pos : SV_POSITION;
    float4 tex : TEXCOORD0;
    float4 posWorld : TEXCOORD1;
    float3 normalDir : TEXCOORD2;
};
```

Fairly straightforward, we assign UV map 1 (TEXCOORD0) to texcoord so we can grab it in the vertex program, and the we will assign it over to tex so we can grab it in the fragment program. Here I will just remind you that the texcoords are important for the vertex input, but the order does not matter for the vertex output.

Now there is only one line to add to the vertex program:

```
o.tex = v.texcoord;
```

That's right, just passing straight through, we don't need to do any fancy work here.

## TEXTURE HANDLING

Once again, we only have one line to add to the fragment program.

```
float4 tex = tex2D(_MainTex, _MainTex_ST.xy * i.tex.xy + _MainTex_ST.zw);
```

Cool, let's explain this.
tex2D() is like our texture unwrap function, it takes an input image '_MainTex' and and unwraps it to the supplied uv coordinates 'i.tex.xy'
Wait, but what about the rest of that?
Well, we need to multiply the uv coordinates by the tiling '_MainTex_ST.xy' and then add the offset '_MainTex_ST.zw'. which is fairly straightforward isn't it.

But now we have an error! Cannot compile for flash!
This is inconvenient but we have reached the limits of the flash renderer, to remove this error you can either set up another subshader for flash, or remove the rim lighting. In my case I chose to simply exclude compiling for flash with the following line:

```
#pragma exclude_renderers flash
```

And we are done, check out the final shader code below.

## FINAL CODE

```
Shader "unityCookie/introduction/6 - Texture Map - pixel" {
    Properties {
        _Color ("Color Tint", Color) = (1.0,1.0,1.0,1.0)
        _MainTex ("Diffuse Texture", 2D) = "white" {}
        _SpecColor ("Specular Color", Color) = (1,1,1,1)
        _Shininess ("Shininess", Float) = 10
        _RimColor ("Rim Color", Color) = (1,1,1,1)
```

```
    _RimPower ("Rim Power",Range(0.1,10)) = 3.0
}
SubShader {
    Pass {
        CGPROGRAM
        #pragma vertex vert
        #pragma fragment frag
        #pragma exclude_renderers flash

        //user defined variables
        uniform sampler2D _MainTex;
        uniform float4 _MainTex_ST;
        uniform float4 _Color;
        uniform float4 _SpecColor;
        uniform float _Shininess;
        uniform float4 _RimColor;
        uniform float _RimPower;

        //Unity defined Variables
        uniform float4 _LightColor0;

        //Base Input Structs
        struct vertexInput{
            float4 vertex : POSITION;
            float3 normal : NORMAL;
            float4 texcoord : TEXCOORD0;
        };
        struct vertexOutput {
            float4 pos : SV_POSITION;
            float4 tex : TEXCOORD0;
            float4 posWorld : TEXCOORD1;
            float3 normalDir : TEXCOORD2;
        };

        //vertex function
        vertexOutput vert(vertexInput v){

            vertexOutput o;

            o.posWorld = mul(_Object2World, v.vertex);
            o.normalDir = normalize( float3( mul( float4(v.normal, 0.0),_World2Object ).xyz ) );
            o.pos = mul(UNITY_MATRIX_MVP, v.vertex);
            o.tex = v.texcoord;
            return o;
        }

        //fragment function
        float4 frag(vertexOutput i) : COLOR
        {
            float3 normalDirection = normalize(i.normalDir);
            float3 viewDirection = normalize(_WorldSpaceCameraPos.xyz - float3(i.posWorld.xyz));
            float3 lightDirection;
            float atten;

            if (0.0 == _WorldSpaceLightPos0.w) //directional Light
            {
                atten = 1.0;
                lightDirection = normalize(float3(_WorldSpaceLightPos0.xyz));
            }
            else{
                float3 vertexToLightSource = float3(_WorldSpaceLightPos0.xyz - float3(i.posWorld.xyz));
                float distance = length(vertexToLightSource);
                atten = 1.0/distance;
                lightDirection = normalize(vertexToLightSource);
```

```
            }

            //Lighting
            float3 ambientLight = UNITY_LIGHTMODEL_AMBIENT.rgb;
            float3 diffuseReflection = atten * float3(_LightColor0.rgb) * max(0.0, dot(normalDirection,
lightDirection));
            float3 specularReflection = max(0.0, dot(normalDirection, lightDirection)) * atten * float3(_LightColor0.
rgb) * float3(_SpecColor.rgb) * pow(max(0.0,dot(reflect(-lightDirection, normalDirection), viewDirection)),
_Shininess);

            //Rim Lighting
            half rim = 1 - saturate(dot(normalize(viewDirection), normalDirection));
            float3 rimLighting = max(0.0,dot(normalDirection, lightDirection) * _RimColor * pow(rim, _RimPower));

            float3 lightFinal = ambientLight + diffuseReflection + specularReflection + rimLighting;

            //Texture Maps
            float4 tex = tex2D(_MainTex, _MainTex_ST.xy * i.tex.xy + _MainTex_ST.zw);

            return float4(tex.rgb * lightFinal * _Color.rgb, 1.0);
        }
        ENDCG
    }
  }
  //fallback commented out during development
  //Fallback "Diffuse"
}
```

That's great, but enough of this easy shader writing, in the next lesson we will get to tackle normal maps, which can be a real challenge.