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 3 - SPECULAR HIGHLIGHTS

Now that we have our nice Lambertian lighting model working, it is time we took this a step further and added some specular highlights. Specular highlights make a model look more shiny such as metals, plastics or something that is wet.
Once we are done we will move our entire lighting model from the vertex program to the fragment program to give us much higher quality lighting.
Finally we will then go on to add in point lights to our shader.

HOW DOES IT WORK?

Well it's quite simple really, the specular highlight is always halfway between our view direction and the light direction, this is why it appears the highlight moves along the surface. We can get the position of the highlight by reflecting the light direction based on the normal and view directions. Sounds complex, looks complex, but is actually rather easy.

THE CODE

There is one line of code that will make this all work:

```
float3 specularReflection = max(0.0, dot(normalDirection, lightDirection)) * atten * _LightColor0.rgb * _
SpecColor.rgb * pow(max(0.0,dot(reflect(-lightDirection, normalDirection), viewDirection)), _Shininess);
```

It just happens to be a really long line. Let's build it from scratch.

```
float3 specularReflection = reflect(lightDirection, normalDirection);
```

First we need to reflect the light direction based on the surface normal, the reflect function does just that, takes the first

ray and reflects it across a specified axis, in this case we use the normal direction. This just results in some weird colors becuase it's just a vector at this point, we need to use a dot product to get our white higlight based on this reflected vector and the view direction.

```
float3 specularReflection = dot(reflect(-lightDirection, normalDirection), viewDirection);
```

Great, so using the dot product we get a nice moving highlight, except it's on the wrong side! this is why we flip the light direction to get the specular highligh on the correct side, now let's stop the highlight wrapping around.

```
float3 specularReflection = max(0.0,dot(reflect(-lightDirection, normalDirection), viewDirection));
```

That was easy, we throw in a good old max function to limit the dot product to positive values. Now let's add some control over the intensity of the specular highlight.

```
float3 specularReflection = pow(max(0.0,dot(reflect(-lightDirection, normalDirection), viewDirection)), _
Shininess);
```

We bring in the pow() function, the pow() function will take the first number to the power of the second, for example pow(a,8) = a*a*a*a*a*a*a*a.
Adding a Shininess variable will do nicely here.
Next we want the specular highlight to fade as it get's closer to the edge.

```
float3 specularReflection = max(0.0, dot(normalDirection, lightDirection))  * pow(max(0.0,dot(reflect(-
lightDirection, normalDirection), viewDirection)), _Shininess);
```

We can use the same dot product from the lambert function to control how much it fades, remember 1 is full and 0 is none.
Finally, let's colour it.

```
float3 specularReflection = max(0.0, dot(normalDirection, lightDirection)) * atten * _LightColor0.rgb * _
SpecColor.rgb * pow(max(0.0,dot(reflect(-lightDirection, normalDirection), viewDirection)), _Shininess);
```

By multiplying it with the light color, and a specular color variable we get a very nice highlight indeed. We also use that attenuation variable that will be important later.

FINAL CODE

```
Shader "unityCookie/introduction/3a - Specular - vertex" {
   Properties {
      _Color ("Color", Color) = (1.0,1.0,1.0,1.0)
      _SpecColor ("Specular Color", Color) = (1,1,1,1)
      _Shininess ("Shininess", Float) = 10
   }
   SubShader {
      Pass {
         CGPROGRAM
         #pragma vertex vert
         #pragma fragment frag

         //user defined variables
         uniform float4 _Color;
         uniform float4 _SpecColor;
         uniform float _Shininess;

         //Unity defined Variables
         uniform float4 _LightColor0;

         //Base Input Structs
         struct vertexInput{
            float4 vertex : POSITION;
            float3 normal : NORMAL;
```

```
        };
        struct vertexOutput {
            float4 pos : SV_POSITION;
            float4 col : COLOR;
        };

        //vertex function
        vertexOutput vert(vertexInput v){

            vertexOutput o;

            float3 normalDirection =  normalize( float3( mul( float4(v.normal, 0.0), _World2Object).rgb ) );
            float3 viewDirection = normalize( float3( float4( float4(_WorldSpaceCameraPos, 1.0) - mul(_
Object2World, v.vertex) ).rgb ) );
            float3 lightDirection;
            float atten = 1.0; //attenuation = 1 for directional lights

            //ambient Light
            float3 ambientLight = UNITY_LIGHTMODEL_AMBIENT.rgb;
            lightDirection = normalize(float3(_WorldSpaceLightPos0.rgb));

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

            float3 lightFinal = ambientLight + diffuseReflection + specularReflection;

            o.col = float4(lightFinal * float3(_Color.rgb), 1.0);
            o.pos = mul(UNITY_MATRIX_MVP, v.vertex);
            return o;
        }

        //fragment function
        float4 frag(vertexOutput i) : COLOR
        {
            return i.col;
        }
        ENDCG
    }
  }
  //fallback commented out during development
  //Fallback “Diffuse”
}
```

Great so that is how we make a specular highlight for our shader, next let's take a look at moving our lighting model to the pixel shader for nice and smooth lighting.

PER PIXEL LIGHTING

Per pixel lighting is fairly straightforward, it's pretty much just copying our lighting model from the vertex program to the fragment program, except we also need to output the worldspace vertex position, and the normal direction from the vertex shader to the fragment shader.

THE STRUCTS

```
struct vertexInput{
    float4 vertex : POSITION;
    float3 normal : NORMAL;
};
struct vertexOutput {
    float4 pos : SV_POSITION;
```

```
    float4 posWorld : TEXCOORD0;
    float3 normalDir : TEXCOORD1;
};
```

Fairly straightforward, we just add in the posWorld and normalDir to our vertexOutput.

THE VERTEX SHADER

```
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);
    return o;
}
```

Well that was easier wasn't it, we removed the lighting model, and added in powWorld and normalDir.
By multiplying the _Object2World built in unity matrix by the vertex position we get the world space vertex position, and by multiplying our vertex normals by the _World2Object matrix we get our normals for the fragment shader.
Very similar to what we did within our per vertex lighting previously, except we are passing the values to the fragment shader to work with.

THE FRAGMENT SHADER

```
float4 frag(vertexOutput i) : COLOR
{
    float3 normalDirection = normalize(i.normalDir);
    float3 viewDirection = normalize(_WorldSpaceCameraPos.xyz - float3(i.posWorld.xyz));
    float3 lightDirection;
    float atten = 1.0;

    float3 ambientLight = UNITY_LIGHTMODEL_AMBIENT.rgb;
    lightDirection = normalize(float3(_WorldSpaceLightPos0.xyz));

    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);

    float3 lightFinal = ambientLight + diffuseReflection + specularReflection;

    return float4(lightFinal * float3(_Color.rgb) , 1.0);
}
```

Almost identical isn't it.
Because we calculated our normals in our vertex shader, we only need to normalize it (squish into 0-1 values), likewise we can replace the second part of the viewDirection calculation and replace it with the posWorld we calculated in the vertex shader.

THE FINAL CODE

```
Shader "unityCookie/introduction/3b - Specular - pixel" {
    Properties {
        _Color ("Color", Color) = (1.0,1.0,1.0,1.0)
        _SpecColor ("Specular Color", Color) = (1,1,1,1)
        _Shininess ("Shininess", Float) = 10
    }
    SubShader {
        Pass {
            Tags { "LightMode" = "ForwardBase" }
            CGPROGRAM
```

```
            #pragma vertex vert
            #pragma fragment frag

            //user defined variables
            uniform float4 _Color;
            uniform float4 _SpecColor;
            uniform float _Shininess;

            //Unity defined Variables
            uniform float4 _LightColor0;

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

            //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);

               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 = 1.0;

               float3 ambientLight = UNITY_LIGHTMODEL_AMBIENT.rgb;
               lightDirection = normalize(float3(_WorldSpaceLightPos0.xyz));

               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);

               float3 lightFinal = ambientLight + diffuseReflection + specularReflection;

               return float4(lightFinal * float3(_Color.rgb) , 1.0);
            }
         }
         ENDCG
      }
      //fallback commented out during development
      //Fallback “Diffuse”
   }
```

So that’s it, we now have some smooth pixel lighting on our shader, let’s move on to rim lighting.