Workshop: Shaders in Unity

Authors: Maarten Bobbeldijk, Karim Tawfik and Peter-paul van der Mark

Can you tell me something about...

- Textures

- Materials

- Shaders

Table of content

What are materials, Shaders & Textures?

- **Materials**
  Definitions of how a surface should be rendered, including references to textures used, tiling information, colour tints and more.

- **Shaders**
  Small scripts that contain the mathematical calculations and algorithms for calculating the colour of each pixel rendered.

- **Textures**
  Simply said, they are bitmap images.

  A Material may contain references to textures, so that the Material's shader can use the textures while calculating the surface colour of an object

More in detail of shaders

Shaders are small scripts that contain the mathematical calculations and algorithms for calculating the colour of each pixel rendered, based on the lighting input and the Material configuration.

Examples:

- Foliage
- Reflections
- Creating a cartoony look in the game
- Special effect, like: night-, heat- or x-ray vision

Standard Unity shaders

Not every shader needs to be created from scratch, Unity itself has a set of shaders by default.

The Standard Shader is designed with hard surfaces in mind and can deal with most real-world materials like stone, glass, ceramics, brass, silver or rubber.

With the Standard Shader, a large range of shader types (such as such as Diffuse, Specular, Bumped Specular, Reflective) are combined into a single shader intended to be used across all material types. Which gives a realistic, consistent and believable distribution of light and shade across all models that use the shader.

What kind of shaders varieties are there?

Surface Shaders
Surface Shaders are your best option if your Shader needs to be affected by lights and shadows, to make it easy to write, a lot of code is generated for you. Unity has implemented these shaders directly in the pipeline. Written in Cg/HLSL.

Vertex and Fragment Shaders
Vertex and Fragment Shaders are required if your Shader doesn't need to interact with lighting, or if you need some very exotic effects, however, these shaders require more programming. Written in Cg/HLSL.

Fixed Function Shaders
Fixed Function Shaders are legacy Shader syntax for very simple effects. It is entirely written in a language called ShaderLab

Shaderlab structure

- Every shader in Unity is written in the Shaderlab structure
- Describes which properties will be used and which ones you can change in the material editor
- The actual shader code, is written in CGPROGRAM (HLSL/Cg shading language)
- If the used shader code isn't available in combination with the hardware API (like Dx11), what is the fallback?
- Each shader file always represents ONE shader.

Simple shaderlab code example

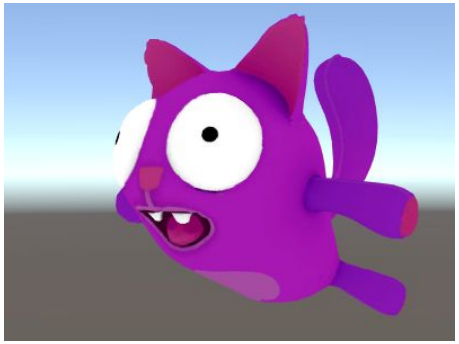
```
Shader "MyShader" {
    Properties {
        _MyTexture ("My Texture", 2D) = "white" { }
        // Place other properties like colors or vectors here as well
    }
    SubShader {
        // here goes your
        // - Surface Shader or
        // - Vertex and Fragment Shader or
        // - Fixed Function Shader
    }
    SubShader {
        // Place a simpler "fallback" version of the SubShader above
        // that can run on older graphics cards here
    }
}
```

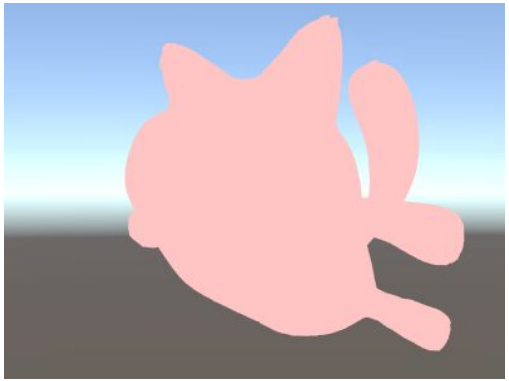
```
Shader "Unlit/SingleColor"
{
    Properties
    {
        // Color property for material inspector, default to white
        _Color ("Main Color", Color) = (1,1,1,1)
    }
    SubShader
    {
        Pass
        {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag

            // vertex shader
            // this time instead of using "appdata" struct, just spell inputs manually,
            // and instead of returning v2f struct, also just return a single output
            // float4 clip position
            float4 vert (float4 vertex : POSITION) : SV_POSITION
            {
                return mul(UNITY_MATRIX_MVP, vertex);
            }

            // color from the material
            fixed4 _Color;

            // pixel shader, no inputs needed
            fixed4 frag () : SV_Target
            {
                return _Color; // just return it
            }
            ENDCG
        }
    }
}
```





Shader compilation details

On shader import time, Unity does not compile the whole shader. This is because majority of shaders have a lot of variants inside, and compiling all of them, for all possible platforms, would take a very long time.

At import time, only do minimal processing of the shader (surface shader generation etc.).
At player build time, all the “not yet compiled” shader variants are compiled, so that they are in the game data even if the editor did not happen to use them.

However, this does mean that a shader might have an error in there, which is not detected at shader import time. However, this can be resolved by using the ‘Compile and show code’ pop-up menu in the shader inspector.

The shaders will be cached in the folder *Library/ShaderCache* and can become quite large. It’s always safe to delete these caches when needed.

Time for some specific shader examples

With the focus on surface shaders.

Simple

```
Shader "Example/Diffuse Simple" {
    SubShader {
      Tags { "RenderType" = "Opaque" }
      CGPROGRAM
      #pragma surface surf Lambert
      struct Input {
          float4 color : COLOR;
      };
      void surf (Input IN, inout SurfaceOutput o) {
          o.Albedo = 1;
      }
      ENDCG
    }
    Fallback "Diffuse"
}
```

Texture

```
Shader "Example/Diffuse Texture" {
  Properties {
    _MainTex ("Texture", 2D) = "white" {}
  }
  SubShader {
    Tags { "RenderType" = "Opaque" }
    CGPROGRAM
    #pragma surface surf Lambert
    struct Input {
        float2 uv_MainTex;
    };
    sampler2D _MainTex;
    void surf (Input IN, inout SurfaceOutput o) {
        o.Albedo = tex2D (_MainTex, IN.uv_MainTex).rgb;
    }
    ENDCG
  }
  Fallback "Diffuse"
}
```

Rim Lighting

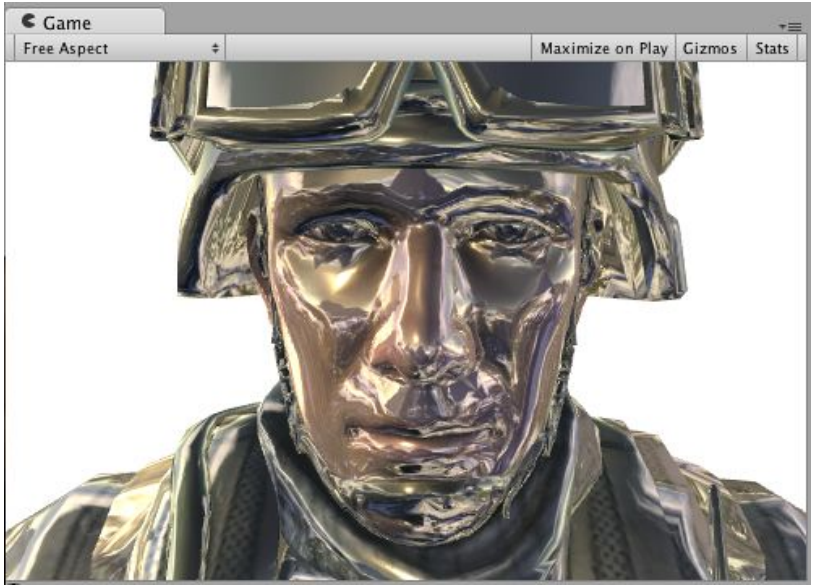
```
Shader "Example/Rim" {
  Properties {
    _MainTex ("Texture", 2D) = "white" {}
    _BumpMap ("Bumpmap", 2D) = "bump" {}
    _RimColor ("Rim Color", Color) = (0.26,0.19,0.16,0.0)
    _RimPower ("Rim Power", Range(0.5,8.0)) = 3.0
  }
  SubShader {
    Tags { "RenderType" = "Opaque" }
    CGPROGRAM
    #pragma surface surf Lambert
    struct Input {
        float2 uv_MainTex;
        float2 uv_BumpMap;
        float3 viewDir;
    };
    sampler2D _MainTex;
    sampler2D _BumpMap;
    float4 _RimColor;
    float _RimPower;
    void surf (Input IN, inout SurfaceOutput o) {
        o.Albedo = tex2D (_MainTex, IN.uv_MainTex).rgb;
        o.Normal = UnpackNormal (tex2D (_BumpMap, IN.uv_BumpMap));
        half rim = 1.0 - saturate(dot (normalize(IN.viewDir), o.Normal));
        o.Emission = _RimColor.rgb * pow (rim, _RimPower);
    }
    ENDCG
  }
  Fallback "Diffuse"
}
```

Final Color Modifier

```
Shader "Example/Tint Final Color" {
  Properties {
    _MainTex ("Texture", 2D) = "white" {}
    _ColorTint ("Tint", Color) = (1.0, 0.6, 0.6, 1.0)
  }
  SubShader {
    Tags { "RenderType" = "Opaque" }
    CGPROGRAM
    #pragma surface surf Lambert finalcolor:mycolor
    struct Input {
        float2 uv_MainTex;
    };
    fixed4 _ColorTint;
    void mycolor (Input IN, SurfaceOutput o, inout fixed4 color)
    {
        color *= _ColorTint;
    }
    sampler2D _MainTex;
    void surf (Input IN, inout SurfaceOutput o) {
        o.Albedo = tex2D (_MainTex, IN.uv_MainTex).rgb;
    }
    ENDCG
  }
  Fallback "Diffuse"
}
```

Cubemap Reflection

```
Shader "Example/WorldRefl" {
  Properties {
    _MainTex ("Texture", 2D) = "white" {}
    _Cube ("Cubemap", CUBE) = "" {}
  }
  SubShader {
    Tags { "RenderType" = "Opaque" }
    CGPROGRAM
    #pragma surface surf Lambert
    struct Input {
        float2 uv_MainTex;
        float3 worldRefl;
    };
    sampler2D _MainTex;
    samplerCUBE _Cube;
    void surf (Input IN, inout SurfaceOutput o) {
        o.Albedo = tex2D (_MainTex, IN.uv_MainTex).rgb * 0.5;
        o.Emission = texCUBE (_Cube, IN.worldRefl).rgb;
    }
    ENDCG
  }
  Fallback "Diffuse"
}
```



Slices via World Space Position

```
Shader "Example/Slices" {
  Properties {
    _MainTex ("Texture", 2D) = "white" {}
    _BumpMap ("Bumpmap", 2D) = "bump" {}
  }
  SubShader {
    Tags { "RenderType" = "Opaque" }
    Cull Off
    CGPROGRAM
    #pragma surface surf Lambert
    struct Input {
        float2 uv_MainTex;
        float2 uv_BumpMap;
        float3 worldPos;
    };
    sampler2D _MainTex;
    sampler2D _BumpMap;
    void surf (Input IN, inout SurfaceOutput o) {
        clip (frac((IN.worldPos.y+IN.worldPos.z*0.1) * 5) - 0.5);
        o.Albedo = tex2D (_MainTex, IN.uv_MainTex).rgb;
        o.Normal = UnpackNormal (tex2D (_BumpMap, IN.uv_BumpMap));
    }
    ENDCG
  }
  Fallback "Diffuse"
}
```

Assignment / Demo 45 Mins

- Programming a shader with the class.
- Download the project ……
- You get 6 basic shaders play around with them and place them in the correct order.
- Make 3 shaders by combining the basic shaders and place them on the correct position.

Tips for writing high-performance shaders

Complex mathematical operations
Transcendental mathematical functions (such as pow, exp, log, cos, sin, tan) are quite resource-intensive, so avoid using them where possible. Consider using lookup textures as an alternative to complex math calculations if applicable.

Floating point precision
While the precision (float vs half vs fixed) of floating point variables is largely ignored on desktop GPUs, it is quite important to get a good performance on mobile GPUs. See the Shader Data Types and Precision page for details.

End of presentation*

* Clapping is appreciated.