

2019

1 2 3
4 5 6
7 8 9
10 11 12

2020

1 2 3
4 5 6
7 8 9
10 11 12

Title

Unity Shader

Keyword

UnityShader为我们提供了非常便利的工具，也就是Shaderlab，一种用于在Unity平台编写着色器的语言。

ShaderLab有效的组织了不同类型的文件到Unity中，例如纹理，顶点着色器，片元着色器等。而在一般的OpenGL或者DirectX当中，这些都是分离的，需要通过OpenGL或者DX的API一步步的加载。所以ShaderLab对这些操作进行了非常高度的封装，用户只需要考虑着色器的实现细节即可。

从设计上来说，ShaderLab类似于CgFX和Direct3D Effects (.FX) 语言，他们都定义了要显示一个材质所需要的所有东西，而不仅仅是着色器代码。

Unity Shader 结构

```
Shader "ShaderName"{
    Properties{
        //在此处定义你的着色器需要调整的参数，纹理等
    }
    SubShader{
        //着色器1号，针对显卡A
    }
    SubShader{
        //着色器2号，针对显卡B，功能同着色器1号
    }
    Fallback "Diffuse" //上述着色器都不管用，就使用Unity默认的表面着色器
}
```

1.shader名字

```
Shader "MyShaders/TestShader"
```

2.材质属性，Properties

Properties

```
{
```

```
    _Color ("Color", Color) = (1, 1, 1, 1)
    _Specular ("Specula", Color) = (1, 1, 1, 1)
    _Gloss ("Gloss", Range(8.0, 256)) = 20
}
```

Summary



Title

Keyword

Numbers and Sliders

```
name ("display name", Range (min, max)) = number  
name ("display name", Float) = number  
name ("display name", Int) = number
```

Colors and Vectors

```
name ("display name", Color) = (number,number,number,number)  
name ("display name", Vector) = (number,number,number,number)
```

Textures

```
name ("display name", 2D) = "defaulttexture" {}  
name ("display name", Cube) = "defaulttexture" {}  
name ("display name", 3D) = "defaulttexture" {}
```

3. SubShader 语句块

每个UnityShader文件可以包含多个SubShader语义块，但是最少要有一个。当Unity需要加载这个UnityShader时，会扫描所有的SubShader语义块，选择第一个能够在目标平台运行的SubShader，如果都不支持，会选用Fallback语义指定的Shader。

```
SubShader{  
    // optional  
    [Tags]  
    // optional  
    [RenderSetup]  
    Pass{  
        }  
        //other passes  
    }
```

标签 (Tags)

渲染状态 (RenderSetup)

Pass可以有多个

Summary

Title

标签 (Tags)

Keyword

Syntax

```
Tags { "TagName1" = "Value1" "TagName2" = "Value2" }
```

表 3.3

SubShader 的标签类型

标签类型	说 明	例 子
Queue	控制渲染顺序，指定该物体属于哪一个渲染队列，通过这种方式可以保证所有的透明物体可以在所有不透明物体后面被渲染（详见第 8 章），我们也可以自定义使用的渲染队列来控制物体的渲染顺序	Tags { "Queue" = "Transparent" }
RenderType	对着色器进行分类，例如这是一个不透明的着色器，或是一个透明的着色器等。这可以被用于着色器替换（Shader Replacement）功能	Tags { "RenderType" = "Opaque" }
DisableBatching	一些 SubShader 在使用 Unity 的批处理功能时会出现问题，例如使用了模型空间下的坐标进行顶点动画（详见 11.3 节）。这时可以通过该标签来直接指明是否对该 SubShader 使用批处理	Tags { "DisableBatching" = "True" }
ForceNoShadowCasting	控制使用该 SubShader 的物体是否会投射阴影（详见 8.4 节）	Tags { "ForceNoShadowCasting" = "True" }
IgnoreProjector	如果该标签值为“True”，那么使用该 SubShader 的物体将不会受 Projector 的影响。通常用于半透明物体	Tags { "IgnoreProjector" = "True" }
CanUseSpriteAtlas	当该 SubShader 是用于精灵（sprites）时，将该标签设为“False”	Tags { "CanUseSpriteAtlas" = "False" }
PreviewType	指明材质面板将如何预览该材质。默认情况下，材质将显示为一个球形，我们可以通过把该标签的值设为“Plane”“SkyBox”来改变预览类型	Tags { "PreviewType" = "Plane" }

Queue列表

队列的指定在Shader中使用ShaderTag {"Queue"="xxxx"}也可以填写xxx+1或者直接指定2440这样的绝对值。其中0-2500 为不透明(Opaque)。2501-5000 透明(Alpha)。

Queue	描述	值
Background	背景层。最先被渲染的队列	1000
Geometry	几何体。一般来说绘制不透明的物件	2000
AlphaTest	透明检测队列。本质上还是不透明队列。	2450
GeometryLast	最终几何体。这是不透明渲染的临界点	2500
Transparent	透明混合队列。一般来说这队列中都是不写深度缓存的	3000
Overlay	覆盖队列。比如UI镜头光晕等等	4000

Summary

渲染状态 (RenderSetup)

- 状态设置

ShaderLab 提供了一系列渲染状态的设置指令，这些指令可以设置显卡的各种状态，例如是否开启混合/深度测试等。表 3.2 给出了 ShaderLab 中常见的渲染状态设置选项。

表 3.2

常见的渲染状态设置选项

状态名称	设置指令	解释
Cull	Cull Back Front Off	设置剔除模式：剔除背面/正面/关闭剔除
ZTest	ZTest Less Greater LEqual GEqual Equal NotEqual Always	设置深度测试时使用的函数
ZWrite	ZWrite On Off	开启/关闭深度写入
Blend	Blend SrcFactor DstFactor	开启并设置混合模式

Cull

```
Cull Back | Front | Off
```

ZTest

```
ZTest (Less | Greater | LEqual | GEqual | Equal | NotEqual | Always)
```

ZWrite

```
ZWrite On | Off
```

Blend

```
Blend sourceBlendMode destBlendMode
Blend sourceBlendMode destBlendMode, alphaSourceBlendMode alphaDestBlendMode
BlendOp colorOp
BlendOp colorOp, alphaOp
AlphaToMask On | Off
```

Offset

```
Offset Factor, Units
```

Allows you specify a depth offset with two parameters. *factor* and *units*. *Factor* scales the maximum Z slope, with respect to X or Y of the polygon, and *units* scale the minimum resolvable depth buffer value. This allows you to force one polygon to be drawn on top of another although they are actually in the same position. For example `Offset 0, -1` pulls the polygon closer to the camera ignoring the polygon's slope, whereas `Offset -1, -1` will pull the polygon even closer when looking at a grazing angle.

Title

Keyword

固定管线shader可以
用到的状态

ColorMask

```
ColorMask RGB | A | 0 | any combination of R, G, B, A
```

Fixed-function Lighting and Material

```
Lighting On | Off
Material { Material Block }
SeparateSpecular On | Off
Color Color-value
ColorMaterial AmbientAndDiffuse | Emission
```

Fixed-function Fog

```
Fog { Fog Block }
```

Fixed-function AlphaTest

```
AlphaTest (Less | Greater | LEqual | GEQual | Equal | NotEqual | Always) CutoffValue
```

Fixed-function Texture combiners

After the render state setup, use [SetTexture](#) commands to specify a number of Textures and their combining modes:

```
SetTexture textureProperty { combine options }
```

Summary

Pass

Keyword

```
Pass{
    [Name]
    [Tags]
    [RenderSetup]
    // other codes
}
```

通过 Name 可以定义该 pass 的名称，通过 UsePass 命令可以直接使用其他 Unity Shader 中的 pass 比如

`UsePass "MyShader/MYPASSNAME"`

Pass 也可以定义 Tags 但是只能使用下面几种

表 3.4

Pass 的标签类型

标签类型	说明	例子
LightMode	定义该 Pass 在 Unity 的渲染流水线中的角色	<code>Tags { "LightMode" = "ForwardBase" }</code>
RequireOptions	用于指定当满足某些条件时才渲染该 Pass，它的值是一个由空格分隔的字符串。目前，Unity 支持的选项有：SoftVegetation。在后面的版本中，可能会增加更多的选项	<code>Tags { "RequireOptions" @ "SoftVegetation" }</code>

特殊的 Pass

- 。 UsePass
- 。 GrabPass

Fallback 语义

它定义了这个着色器最次的一种形式，也就是如果上面所有的子着色器都用不了的话，就会用 Fallback 语义定义的着色器，你也可以关闭 Fallback，这样一来就相当于，如果所有的 SubShader 都不管用的话，那就随他去吧

Summary

Title

Keyword

Unity Shader 形式

包含表面着色器，固定管线着色器，顶点着色器，片元着色器

```
Shader "Custom/Simple VertexFragment Shader"{
    SubShader{
        Pass{
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag
            float4 vert(float4 v : POSITION):SV_POSITION{
                return mul(UNITY_MATRIX_MVP,v);
            }
            fixed4 frag():SV_TARGET{
                return fixed4(1.0,0,0,1.0);
            }
            ENDCG
        }
    }
}
```

Shader编程语言

既然Shader是一段代码，那必然要用一种语言来书写它，目前主流的有三种语言：

1. 基于OpenGL的OpenGL Shading Language, 简称GLSL。
2. 基于DirectX的High Level Shading Language,简称HLSL。
3. 还有NVIDIA公司的C for Graphic, 简称Cg语言。

Summary

Title

Keyword

内置的定义 `UnityCG.cginc`

- `appdata_base`: position, normal and one texture coordinate.
- `appdata_tan`: position, tangent, normal and one texture coordinate.
- `appdata_full`: position, tangent, normal, four texture coordinates and color.

To access different vertex data, you need to declare the vertex structure yourself, or add input parameters to the vertex shader. Vertex data is identified by Cg/HLSL [semantics](#), and must be from the following list:

- `POSITION` is the vertex position, typically a `float3` or `float4`.
- `NORMAL` is the vertex normal, typically a `float3`.
- `TEXCOORD0` is the first UV coordinate, typically `float2`, `float3` or `float4`.
- `TEXCOORD1`, `TEXCOORD2` and `TEXCOORD3` are the 2nd, 3rd and 4th UV coordinates, respectively.
- `TANGENT` is the tangent vector (used for normal mapping), typically a `float4`.
- `COLOR` is the per-vertex color, typically a `float4`.

Summary