Unity Shader

图形学Shader

实际上就是一小段程序，它负责将输入的Mesh（网格）以指定的方式和输入的贴图或者颜色等组合作用，然后输出。绘图单元可以依据这个输出来将图像绘制到屏幕上。输入的贴图或者颜色等，加上对应的Shader，以及对Shader的特定的参数设置，将这些内容（Shader及输入参数）打包存储在一起，得到的就是一个Material（材质）。之后，我们可以将材质赋予合适的renderer（渲染器）来进行渲染（输出）。

只是一段规定好输入（颜色，贴图等）和输出（渲染器能够读懂的点和颜色的对应关系）的程序。Shader开发者要做的就是根据输入，进行计算变换，产生输出。

计算机图形学的中渲染管线分类

- 固定功能渲染管线(fixed functionrendering pipelines)
- 可编程渲染管线(programable rendering pipelines)

分类

- 固定功能着色器（Fixed Function Shader）
  - 为了兼容老式显卡，Unity已经在内部为我们做了大量的工作，我们只要稍微记住一些关键字、一些规范就可以实现出很多不错的效果。
  - 特征是里面出现Material块，没有嵌套CG语言，代码段中没有CGPROGARAM和ENDCG关键字
- 表面着色器（Surface Shader）
  - Unity自己发扬光大的一项使Shader的书写门槛降低和更易用的技术
  - 嵌套了CG语言，代码段中有surf函数的，就是表面着色器
- 顶点着色器&片段着色器（Vertex Shader & Fragment Shader）
  - 顶点着色器：产生纹理坐标，颜色，点大小，雾坐标，然后把它们传递给裁剪阶段。
  片段着色器：进行纹理查找，决定什么时候执行纹理查找，是否进行纹理查找，及把什么作为纹理坐标
  - 嵌套了CG语言，代码段中有#pragma vertex name 和 #pragma fragment frag声明

基本框架

- Properties（属性）
- 一个或多个SubShader（子着色器）
  - 一个或多个Pass通道
- FallBack（备胎）

Shader的名字

决定shader在material里出现的路径

属性

_Name(“Display Name”, type) = defaultValue[{options}]

Shader在Unity编辑器暴露给美术的参数，通过Properties来实现，除了通过编辑器编辑Properties，脚本也可以通过Material的接口（比如SetFloat、SetTexture）来实现

- _Name　属性的名字，简单说就是变量名，在之后整个Shader代码中将使用这个名字来获取该属性的内容
- Display Name　这个字符串将显示在Unity的材质编辑器中作为Shader的使用者可读的内容
- type　属性的类型，可能的type所表示的内容有以下几种
  - Color　一种颜色，由RGBA（红绿蓝和透明度）四个量来定义
  - 2D　一张2的阶数大小（256，512之类）的贴图。这张贴图将在采样后被转为对应基于模型UV的每个像素的颜色，最终被显示出来
  - Rect　一个非2阶数大小的贴图
  - Cube　即Cube map texture（立方体纹理），简单说就是6张有联系的2D贴图的组合，主要用来做反射效果（比如天空盒和动态反射），也会被转换为对应点的采样
  - Range(min, max)　一个介于最小值和最大值之间的浮点数，一般用来当作调整Shader某些特性的参数（比如透明度渲染的截止值可以是从0至1的值等）
  - Float　任意一个浮点数
  - Vector　一个四维数
- defaultValue 定义了这个属性的默认值，通过输入一个符合格式的默认值来指定对应属性的初始值（某些效果可能需要某些特定的参数值来达到需要的效果，虽然这些值可以在之后在进行调整，但是如果默认就指定为想要的值的话就省去了一个个调整的时间，方便很多
  - Color　以0～1定义的rgba颜色，比如(1,1,1,1)
  - 2D/Rect/Cube　对于贴图来说，默认值可以为一个代表默认tint颜色的字符串，可以是空字符串或者"white","black","gray","bump"中的一个
  - Float，Range – 某个指定的浮点数
  - Vector　一个4维数，写为（x,y,z,w）
- 另外还有一个{option}，它只对2D，Rect或者Cube贴图有关，在写输入时我们最少要在贴图之后写一对什么都不含的空白的{}，当我们需要打开特定选项时可以把其写在这对花括号内。如果需要同时打开多个选项，可以使用空白分隔。可能的选择有ObjectLinear，EyeLinear，SphereMap，CubeReflect，CubeNormal中的一个，这些都是OpenGL中TexGen的模式

SubShader

一个Shader有多个SubShader。一个SubShader可理解为一个Shader的一个渲染方案。即SubShader是为了针对不同的渲染情况而编写的。每个Shader至少1个SubShader、理论可以无限多个，但往往两三个就足够。一个时刻只会选取一个SubShader进行渲染，具体SubShader的选取规则包括：

- 从上到下选取
- SubShader的标签、Pass的标签
  - 是否符合当前的“Unity渲染路径”
  - 是否符合当前的ReplacementTag
- SubShader是否和当前的GPU兼容
- 按此规则第一个被选取的SubShader将会用于渲染，未被选取的SubShader在这次渲染将被忽略

Tags

Tags { “RenderType”=“Opaque” }

Tag指定了这个SubShader的渲染顺序（时机），以及其他的一些设置

- 表面着色器可以被若干的标签（tags）所修饰，而硬件将通过判定这些标签来决定什么时候调用该着色器
- Unity定义了一些这样的渲染过程，RenderType标签，Unity可以运行时替换符合特定RenderType的所有Shader
  - “Opaque”：绝大部分不透明的物体都使用这个
  - “Transparent”：绝大部分透明的物体、包括粒子特效都使用这个
  - "Background"：天空盒都使用这个
  - "Overlay"：GUI、镜头光晕都使用这个
  - 用户也可以定义任意自己的RenderType这个标签所取的值
- 如果你使用Unity做过一些透明和不透明物体的混合的话，很可能已经遇到过不透明物体无法呈现在透明物体之后的情况。这种情况很可能是由于Shader的渲染顺序不正确导致的。Queue指定了物体的渲染顺序，预定义的Queue有：
  - Background – 最早被调用的渲染，用来渲染天空盒或者背景
  - Geometry – 这是默认值，用来渲染非透明物体（普通情况下，场景中的绝大多数物体应该是非透明的）
  - AlphaTest – 用来渲染经过Alpha Test的像素，单独为AlphaTest设定一个Queue是出于对效率的考虑
  - Transparent – 以从后往前的顺序渲染透明物体
  - Overlay – 用来渲染叠加的效果，是渲染的最后阶段（比如镜头光晕等特效）
  - 这些预定义的值本质上是一组定义整数，Background = 1000，Geometry = 2000，AlphaTest = 2450，Transparent = 3000，最后Overlay = 4000。在我们实际设置Queue值时，不仅能使用上面的几个预定义值，我们也可以指定自己的Queue值，写成类似这样：”Queue”=“Transparent+100”，表示一个在Transparent之后100的Queue上进行调用。通过调整Queue值，我们可以确保某些物体一定在另一些物体之前或者之后渲染，这个技巧有时候很有用处。
- 比较有用的标签还有”IgnoreProjector”=“True”（不被Projectors影响，不接受Projector组件的投影），”ForceNoShadowCasting”=“True”（从不产生阴影）以及”Queue”=“xxx”（指定渲染顺序队列）
- 一帧的渲染队列的生成，依次决定于每个渲染物体的
  - Shader的RenderType tag
  - Renderer.SortingLayerID
  - Renderer.SortingOrder
  - Material.renderQueue（默认值为Shader里的"Queue"）
  - Transform.z(ViewSpace)（默认为按z值从前到后，但当Queue是“Transparent”的时候，按z值从后到前）
  - 这个渲染队列决定了之后，渲染器再依次遍历这个渲染队列，“同一种”材质的渲染物体合到一个Batch里

Shader主体

Pass

一个SubShader（渲染方案）是由一个个Pass块来执行的。每个Pass都会消耗对应的一个DrawCall。在满足渲染效果的情况下尽可能地减少Pass的数量

Tag

最重要Tag是 “LightMode”，指定Pass和Unity的哪一种渲染路径（“RenderingPath”）搭配使用。除最重要的ForwardBase、ForwardAdd外Tag取值可包括：

- Always，永远都渲染，但不处理光照
- ShadowCaster，用于渲染产生阴影的物体
- ShadowCollector，用于收集物体阴影到屏幕坐标Buff里

LOD

是Level of Detail的缩写，Unity的内建Diffuse着色器的设定值是200。这个数值决定了我们能用什么样的Shader。在Unity的Quality Settings中我们可以设定允许的最大LOD，当设定的LOD小于SubShader所指定的LOD时，这个SubShader将不可用。Unity内建Shader定义了一组LOD的数值，我们在实现自己的Shader的时候可以将其作为参考来设定自己的LOD数值，这样在之后调整根据设备图形性能来调整画质时可以进行比较精确的控制。

编译指令

CGPROGRAM 开始标记，表明从这里开始是一段CG程序（写Unity的Shader时用的是Cg/HLSL语言）。最后一行的ENDCG与CGPROGRAM是对应的，表明CG程序到此结束

#pragma surface surfaceFunction lightModel [optional params]

- surface – 声明的是一个表面着色器
- surfaceFunction – 着色器代码的方法的名字
- lightModel – 使用的光照模型

#pragma vertex name
#pragma fragment name

替换name，来指定Vertex Shader函数、Fragment Shader函数

#pragma target name

替换name（为2.0、3.0等）。设置编译目标shader model的版本

#pragma only_renderers name name …
#pragma exclude_renderers name name…

#pragma only_renderers goes gles3,
#pragma exclude_renderers d3d9 d3d11 OpenGL,
只为指定渲染平台（render platform）编译

数据容器接口

加载以后的texture（贴图）说白了不过是一块内存存储的，使用了RGB（也许还有A）通道，且每个通道8bits的数据。而具体地想知道像素与坐标的对应关系，以及获取这些数据，我们总不能一次一次去自己计算内存地址或者偏移，因此可以通过sampler2D来对贴图进行操作。简单地理解，sampler2D就是GLSL中的2D贴图的类型，相应的，还有sampler1D，sampler3D，samplerCube等等格式

shader其实是由两个相对独立的块组成的，外层的属性声明，回滚等等是Unity可以直接使用和编译的ShaderLab；而在CGPROGRAM...ENDCG这样一个代码块中，是一段CG程序。对于这段CG程序，要想访问在Properties中所定义的变量的话，必须使用和之前变量相同的名字进行声明。于是sampler2D _MainTex;做的事情就是再次声明并链接了_MainTex，使得接下来的CG程序能够使用这个变量

struct结构体

Input是需要我们去定义的结构，这给我们提供了一个机会，可以把所需要参与计算的数据都放到这个Input结构中，传入surf函数使用；SurfaceOutput是已经定义好了里面类型输出结构，但是一开始的时候内容暂时是空白的，我们需要向里面填写输出，这样就可以完成着色了

float和vec都可以在之后加入一个2到4的数字，来表示被打包在一起的2到4个同类型数，在访问这些值时，可以只使用名称来获得整组值，也可以使用下标的方式（比如.xyzw，.rgba或它们的部分比如.x等等）来获得某个值

UV mapping的作用是将一个2D贴图上的点按照一定规则映射到3D模型上，是3D渲染中最常见的一种顶点处理手段。在CG程序中，有这样的约定，在一个贴图变量之前加上uv两个字母，就代表提取它的uv值（代表贴图上点的二维坐标）。之后就可以在surf程序中直接通过访问uv_MainTex来取得这张贴图当前需要计算的点的坐标值

surf函数

#pragma段中已经指出了着色器代码方法的名字叫做surf，就是这段代码是着色器的工作核心。CG规定了声明表面着色器方法（就是我们这里的surf）的参数类型和名字，因此我们没有权利决定surf的输入输出参数的类型，只能按照规定写。这个规定就是第一个参数是一个Input结构，第二个参数是一个inout的SurfaceOutput结构

数据类型

有3种基本数值类型：float、half、fixed,这3种基本数值类型可以再组成vector和matrix，比如half3是由3个half组成、float4x4是由16个half组成

- float：32位高精度浮点数
- half：16位中精度浮点数。范围是[-6万， +6万]，能精确到十进制的小数点后3.3位
- fixed：11位低精度浮点数。范围是[-2，2]，精度是1/256

数据类型影响性能
精度够用就好

颜色和单位向量，使用fixed
其他情况，尽量使用half（即范围在[-6万， +6万]内、精确到小数点后3.3位）；否则才使用float

可写的SurfaceOutput，SurfaceOutput是预定义的输出结构，我们的surf函数的目标就是根据输入把这个输出结构填上。SurfaceOutput结构体的定义如下

- half3
  - Albedo　像素的颜色
  - Normal　像素的法向值
  - Emission　像素的发散颜色
- half
  - Specular　像素的镜面高光
  - Gloss　像素的发光强度
  - Alpha　像素的透明度

half和我们常见float与double类似，都表示浮点数，只不过精度不一样，half指的是半精度浮点数，精度最低，运算性能相对比高精度浮点数高一些，因此被大量使用

常用的CG函数

- UnpackNormal
  - fixed4输入
  - 返回fixed3
  - 返回对应的法线贴图
- saturate
  - 限制值函数
  - 限制0–1
- dot
  - 点乘
- tex2D
  - 通过uv查找贴图纹理的每个定点颜色值

FallBack

备胎

当本Shader的所有SubShader都不支持当前显卡，就会使用FallBack语句指定的另一个Shader。FallBack最好指定Unity自己预制的Shader实现，因其一般能够在当前所有显卡运行