UnityShader快速上手指南（一） - 玄雨

简介

引言

其实网上有很多shader教程，但是大概看了下，也不知是网上各位大神已经脱离了代码层面的高度还是啥原因。貌似没有找到从代码方面作为入门讲解的，导致了shader对于苦逼程序员入门有一定要求，鄙人不才，来写个比较低级的从代码入门的shader教程吧。

写在前面的话

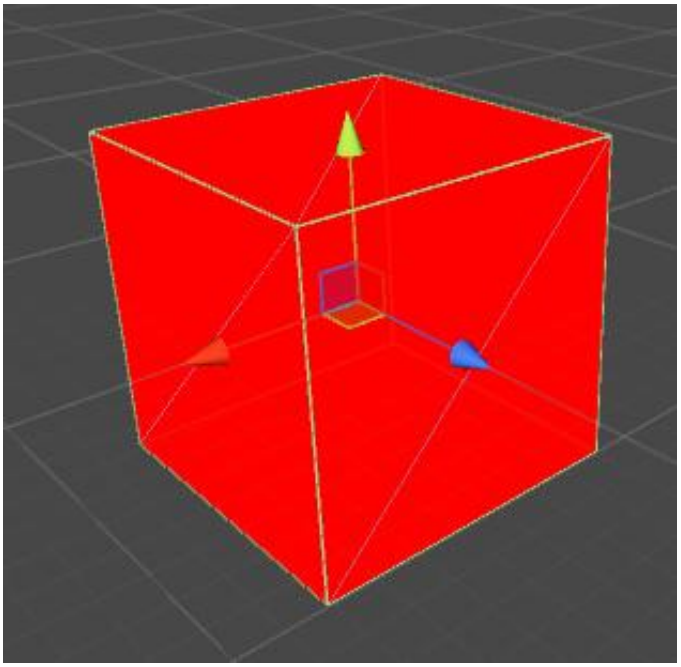
了解过unityshader的人都知道，unityshader分三种，固定管线、表面着色器、顶点和片段着色器，具体区别书面上以及网上大神已经解释的很清楚了，我就不多做赘述了，我这一系列教程只从顶点和片段着色器教程开始，跟其他教程可能也有些区别（一来就上比较坑爹的部分），如果您没听过顶点和片段着色器的，请止步于此。

写Shader的工具

如果已经习惯了自己的IDE请直接跳过，只是推荐一下
1.下载安装sublime text
2.下载该插件https://github.com/cjsjy123/Unity-Shader
3.将文件夹改名为Unity-Shader（就是去掉后面的-master）然后复制到sublime的packages路径下
4.在sublime下设置unity的shader include路径
    preferences ->Packages settting -> Unity-Shader - >setting - Default
    路径是：D:/Program Files/Unity/Editor/Data/CGIncludes（自行类比）
5.愉快的用sublime编写shader吧（支持代码跳转及基本自动完成功能）

20行的Shader

效果：



```
Shader "LT/Lesson1"
{
    SubShader
    {
        Pass
        {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag

            float4  vert ( float4 position : POSITION ) : SV_POSITION
            {
                return mul( UNITY_MATRIX_MVP, position );
            }

            fixed4 frag ( void ) : COLOR
            {
                return float4 (1,0,0,1);
            }

            ENDCG
        }
    }
}
```

这里我们大概解释一下我们要用的shader的代码结构：
Shader的语法是

```
    Shader "名字" // 这个名字自己随便取，类似于类名,不需要与文件名相同
```

```
    {
        Properties {} // 可以从unity中设置的外接属性，后面具体讲怎么设置
        SubShader {
            // 执行的渲染PASS，可以有多个Pass，都会被执行
            Pass{
            // 具体代码
            }
        }
        .
        . // 可以有很多个，gpu执行的时候从上往下，找到第一个可以执行的SubShader执行
        .
        SubShader {
            // 执行的渲染
        }
        // 备胎，在没有任何Subshader可以执行时，用该shader代替
        Fallback "Diffuse"
    }
```

以上就是我们大概要用的代码结构，其实还有其他的，但是为了快速上手，后续要扩展再引入
其中除了SubShader不能删，其他的不用的都可以删掉
所以精简出我们的第一个shader的结构:

```
Shader "名字"
{
    SubShader
    {
        Pass
        {
            // 代码
        }
    }
}
```

下面讲代码部分:
我们要用的CG语言，前后必须用CGPROGRAM和ENDCG包起来，
然后中间的部分就是我们真正的cg代码了

```
#pragma vertex vert
// 这一行是定义处理顶点信息的函数名
// 模型中每一个顶点会调用一次该函数，GPU执行运算，注意性能
// 格式是: #pragma vertex 函数名，函数名字可以自己随意取

#pragma fragment frag
// 这一行是定义处理片面信息的函数名
```

```
// 具体调用时机不是很确定,应该是跟面数相关
// 格式是: #pragma fragment 函数名，函数名字可以自己随意取

float4  vert ( float4 position : POSITION ) : SV_POSITION
// 对应的上面顶点处理的函数，传入对象是一个float4
// 其实是一个顶点的详细信息，只是我们这里通过: POSITION单独选取了位置信息而已
// 后面的: SV_POSITION表示return的float4作为SV_POSITION(不可变顶点)
// 其实传出为POSITION也是可以的,DX10之后才有区别，SV_POSITION性能更高
{
    return mul( UNITY_MATRIX_MVP, position );
    // 这里做了一个与UNITY的矩阵世界做了一个乘法操作
    // 用于将模型坐标换算成世界坐标,如果需要通过shader更改顶点位置，需要在这里进行操作
}

fixed4 frag ( void ) : COLOR
// 对应上面的面片处理的函数，这里为了简单，不处理任何数据
// : COLOR 表示返回的fixed4类型作为COLOR处理
{
    return float4 (1,0,0,1);
    // 直接返回一个固定颜色（float4可以强转成fixed4，这里为了教程演示强转使用）
}
```

我们想在Unity中改颜色

先来明确编写思路：
从上面的解释可以看出，我们的颜色处理是在frag 函数中返回的
所以我们要想改变物体颜色，只需要更改这个返回值即可
那么要从unity中更改颜色,我们就需要一个外接属性
恩，上面也讲了，从Properties 中设置！
直接上代码吧

```
Shader "LT/Lesson1"
{
    Properties {
        _Color ("Main Color", Color ) = (1,0,0,1)
    }
    SubShader
    {
        Pass
        {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag
```

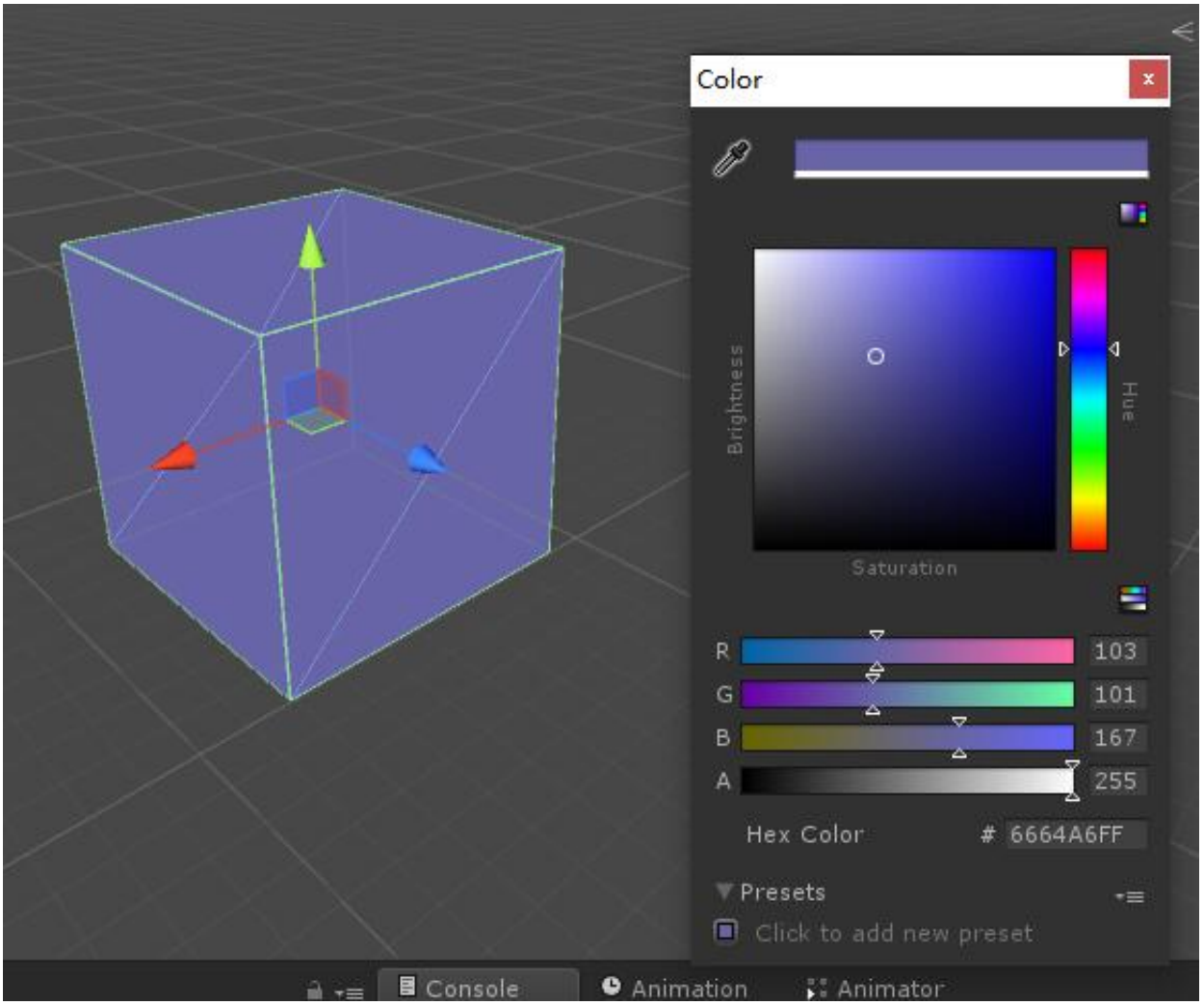
```
            uniform float4 _Color;

            float4  vert ( float4 position : POSITION ) : SV_POSITION
            {
                return mul( UNITY_MATRIX_MVP, position );
            }

            fixed4 frag ( void ) : COLOR
            {
                return _Color;
            }

            ENDCG
        }
    }
}
```

好~
来解释一下：
_Color ("Main Color", Color) = (1,0,0,1)
表示我在Unity的Shader面板中定义了一个_Color的外接属性
格式为 [变量名字] ("Unity中显示的名字", [类型]) = (值)
具体后面还有其他的，我也背不下来，用到再讲，也可以自行查阅
然后定义了之后，并不能直接被CG代码识别，
需要用到 uniform float4 _Color;申明一下该变量名（名字必须完全一致）
然后后续操作就不解释了吧，直接该颜色，返回就行
我们来看看效果



我想更改顶点渲染位置

哈哈，这里就是shader开始牛逼的地方了：
我们可以很高效的对模型顶点数据进行特殊处理！
这次我们要更改vert函数（这个应该不难想吧,顶点相关的操作一般都在这里做了）
直接来代码：

```
Shader "LT/Lesson1"
{
    Properties {
        _Color ("Main Color", Color ) = (1,0,0,1)
    }
    SubShader
    {
        Pass
        {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag
```

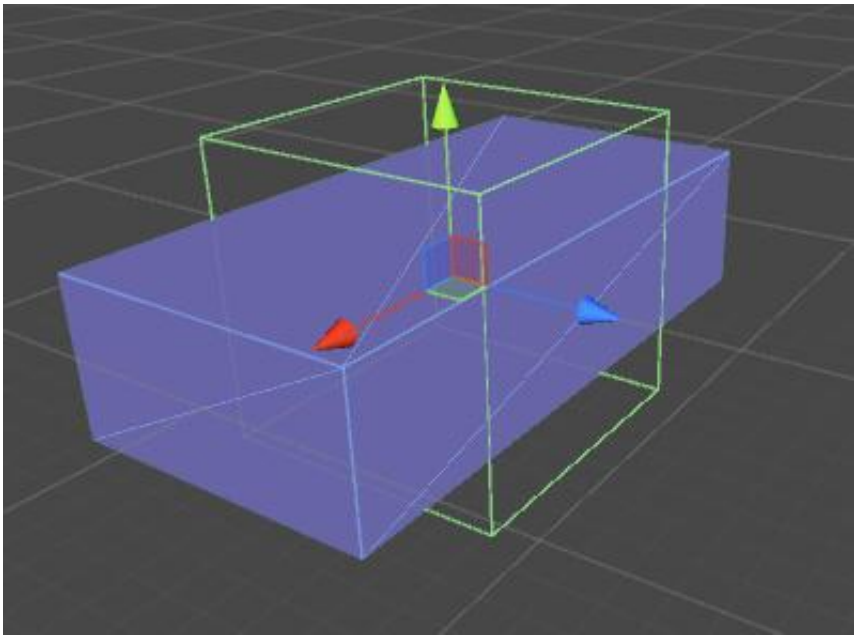
```
            uniform float4 _Color;

            float4  vert ( float4 position : POSITION ) : SV_POSITION
            {
                return mul( UNITY_MATRIX_MVP, position * float4 (2,0.5,1,1));
            }

            fixed4 frag ( void ) : COLOR
            {
                return _Color;
            }

            ENDCG
        }
    }
}
```

来，先看效果再解释



可以很清楚地看到，X轴被放大到了2倍，y变为了0.5倍，z没有变化，
然后对比一下position * float4 (2,0.5,1,1)这个操作，应该很容易猜出这个操作的含义了吧
float4 (x,y,z,u)这个float4的xyz分别对应三个轴上的缩放量,u呢则是我们的单位长度
具体的缩放关系体现为 坐标位置 = 原始位置 * (x / u)
当然，你也可以在这个函数中对顶线信息进行其他操作，这里只是最简单一个例子

总结

到这里，今天的blog就到此结束了，希望大家能看懂，不明白的欢迎加QQ：821580467一起探讨