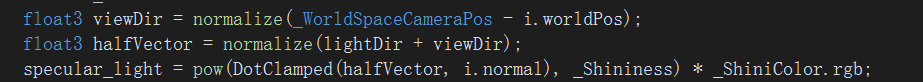
# Homework4报告

## 基础shader的实现

1. 实现过程：

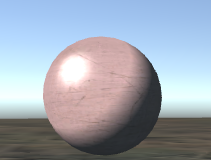
基础着色器参照作业文档实现，其中Blinn-Phong中计算镜面高光的代码为



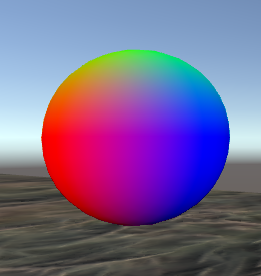
\_Shininess为反射系数，\_ShiniColor为反射光的颜色。

1. 实现结果：

Blinn-Phong shader：



Normal shader：



## ShaderGUI的实现

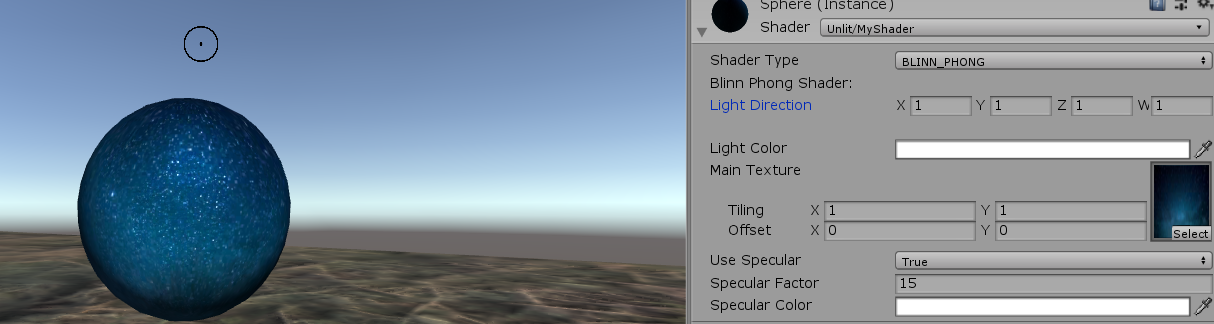
1. 实现过程：

实现过程参照文档。

在MyShader着色器中定义了两个shader\_feature：一个是USE\_NORMAL，由用户选择使用Blinn-Phong渲染效果和Normal渲染效果；一个是USE\_SPECULAR，当用户选择Blinn-Phong时，由用户选择是否打开高光。

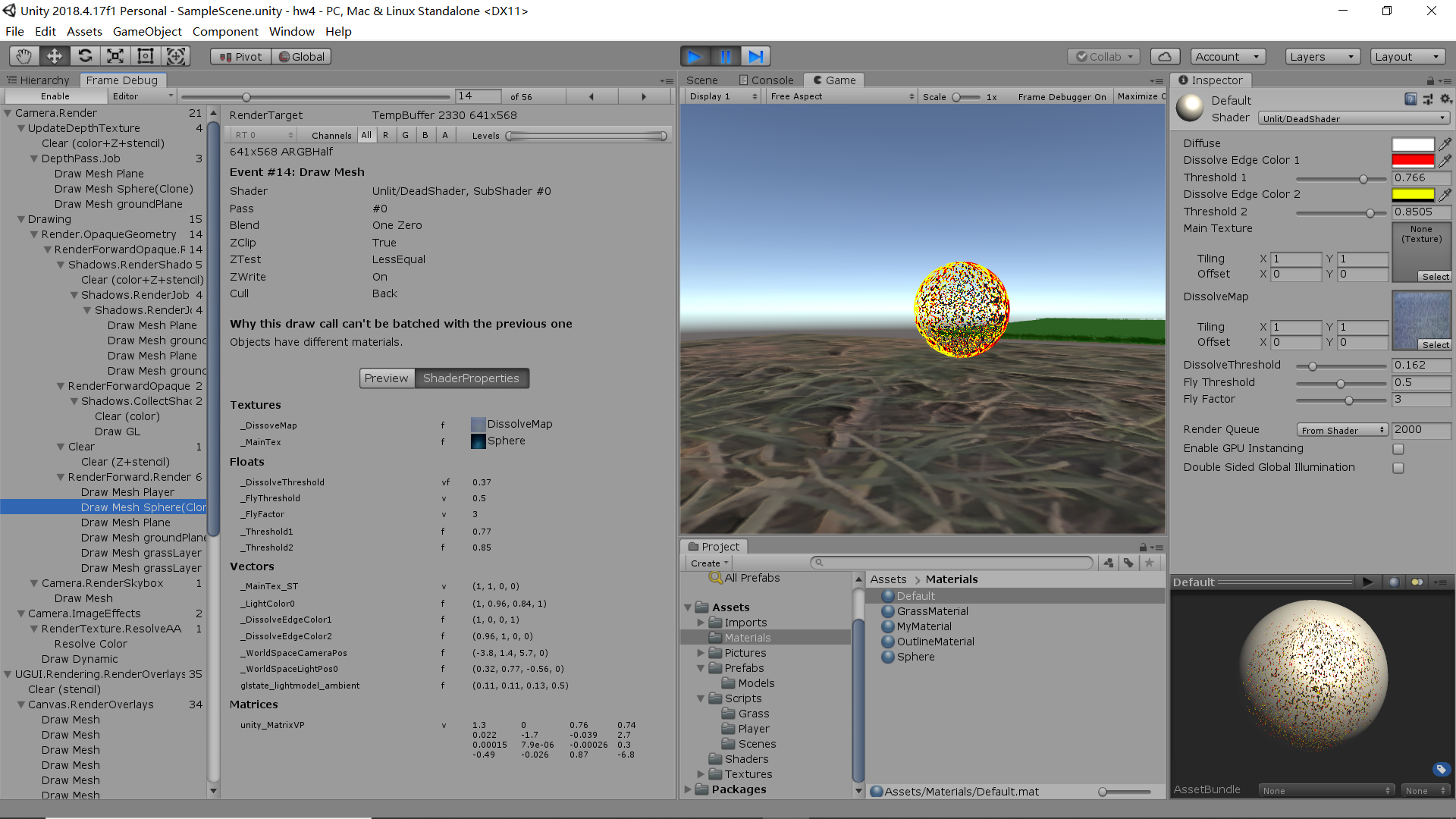
在用户选择Blinn-Phong渲染效果时，除了文档中提到的参数，另外添加了Light Direction（平行光光源方向）、Light Color（光源颜色）、Specular Color（高光颜色）三个参数。

1. 实现结果：

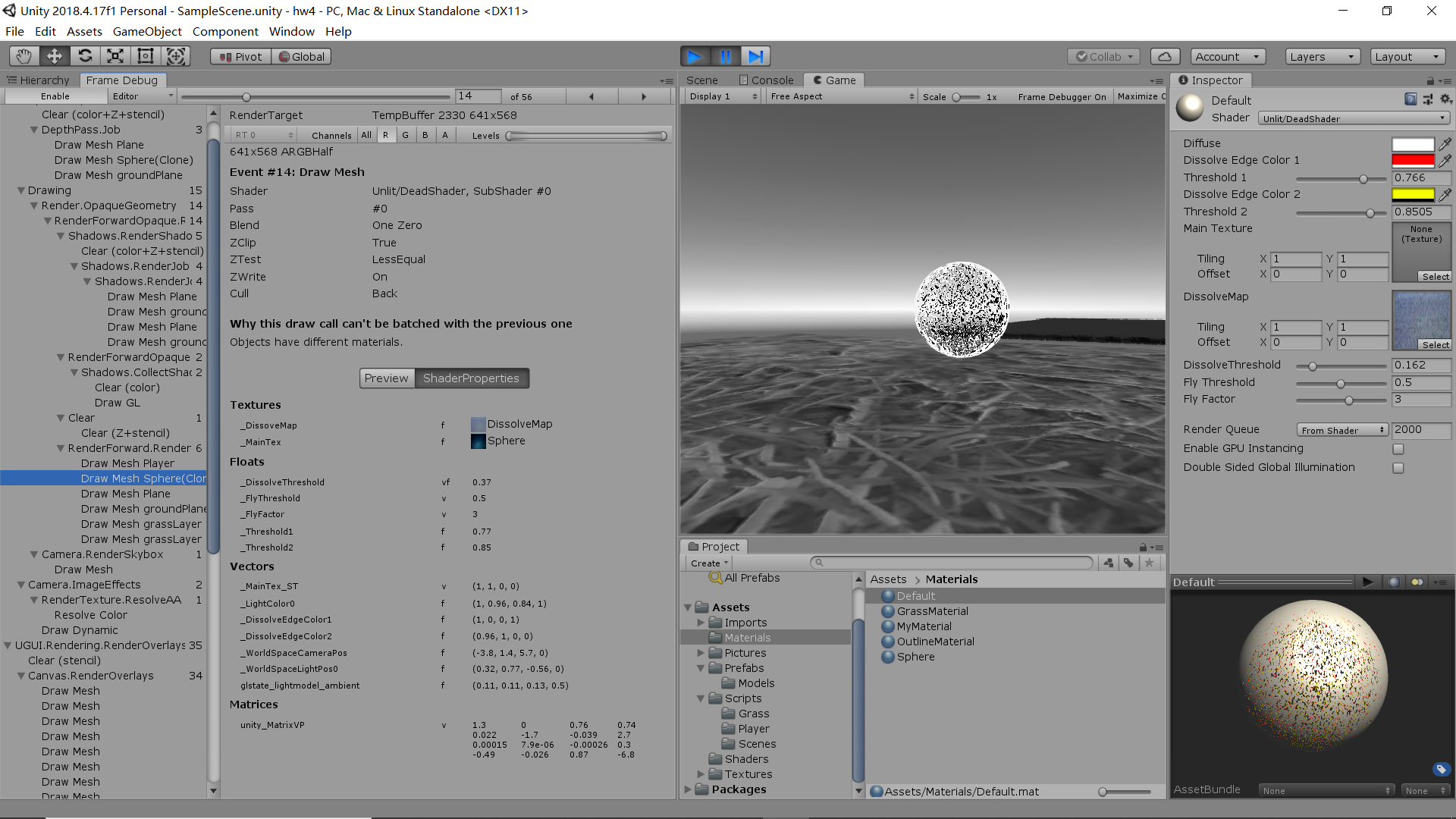




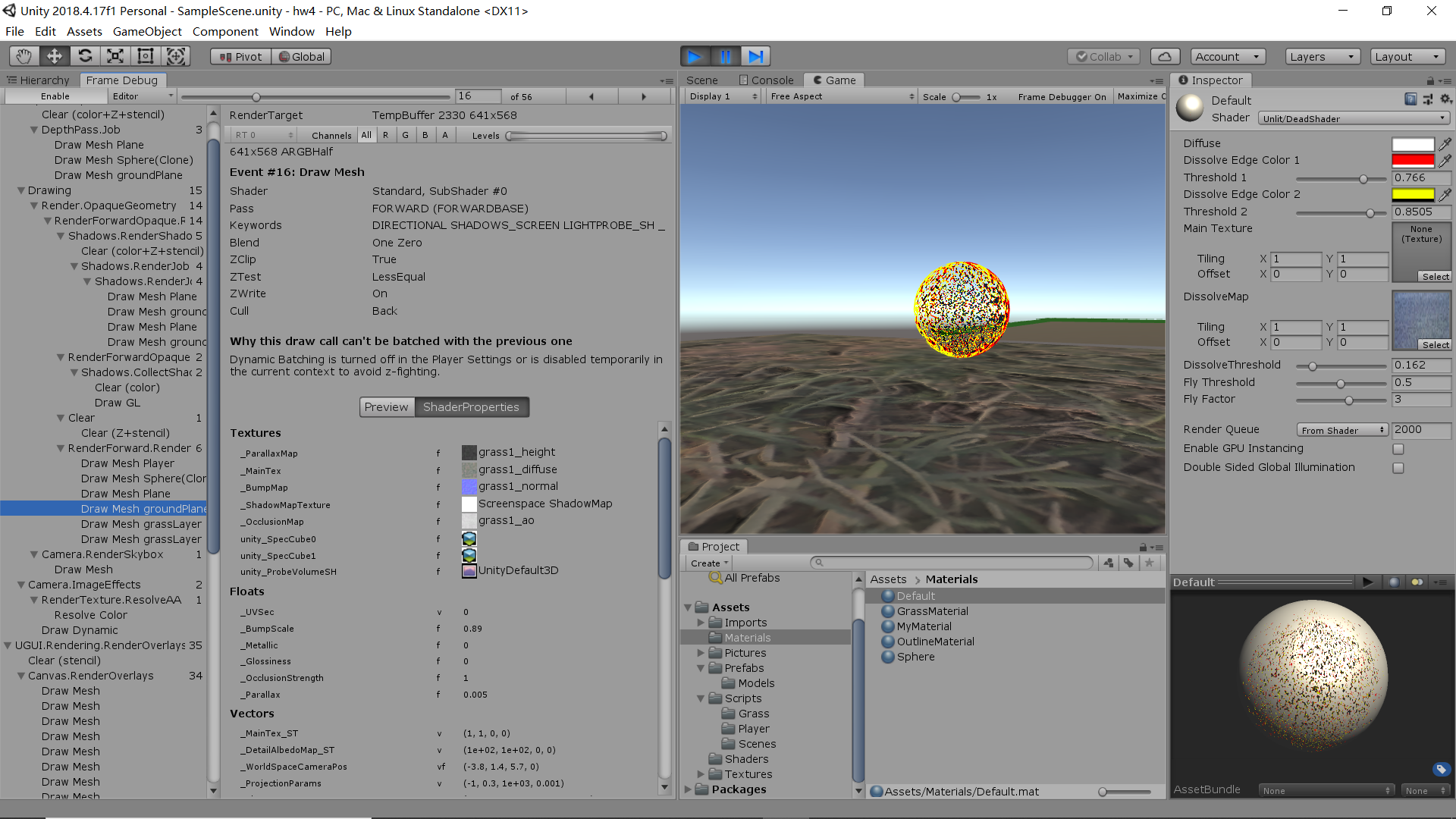
## Debug工具的使用过程

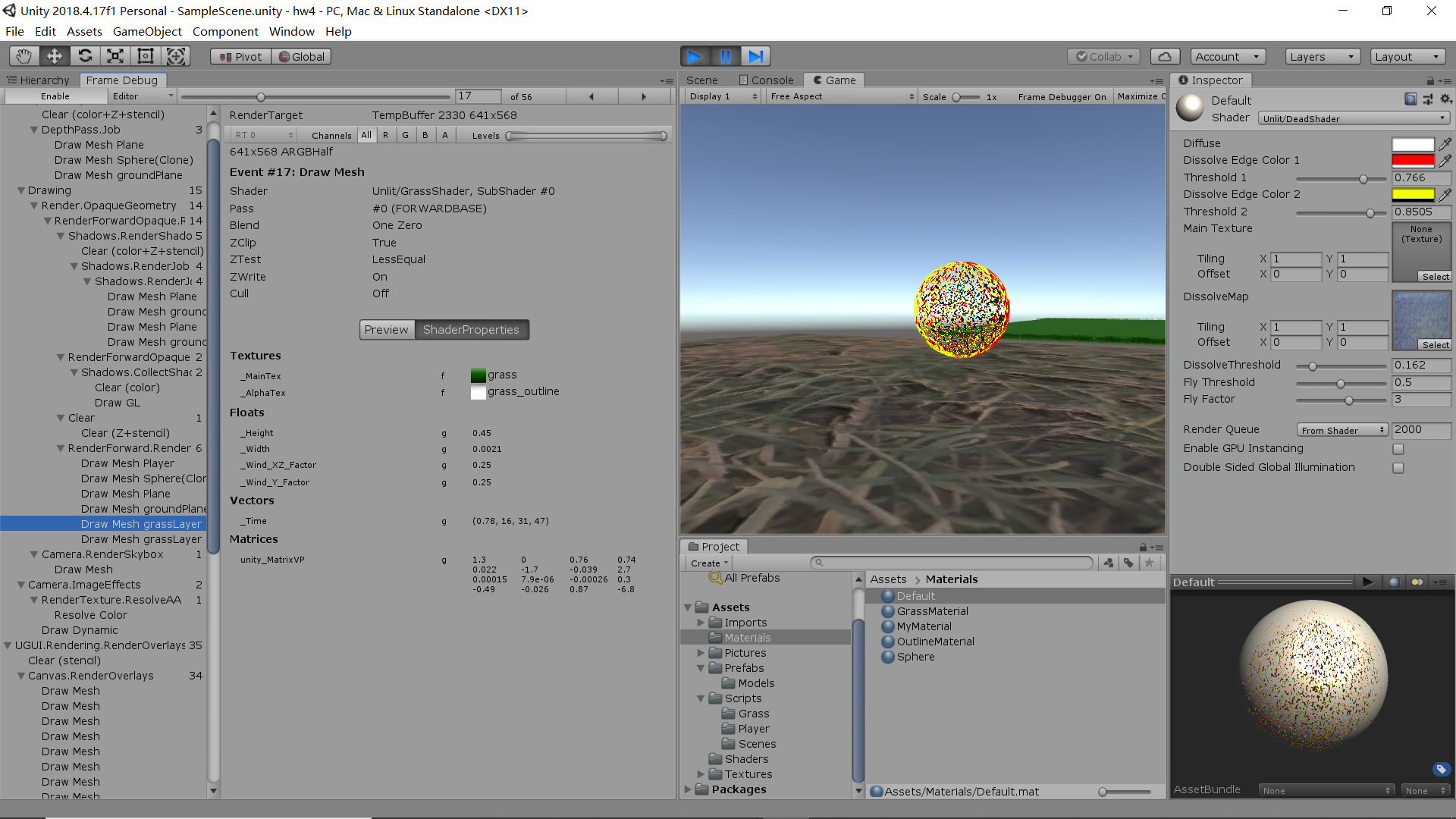


在frame debugger中可以看到这个球使用了dead shader，并且可以看到shader详细的参数

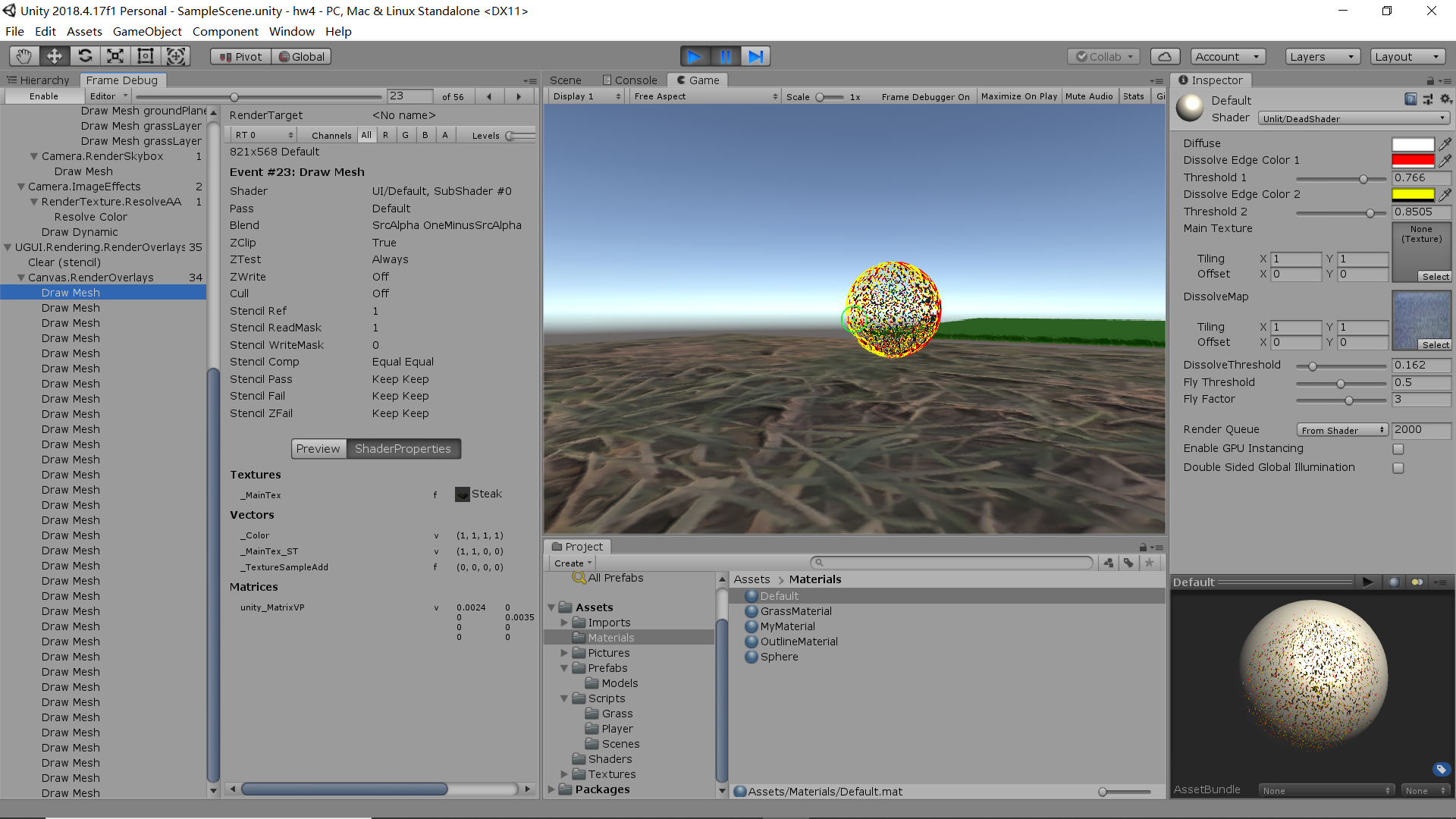


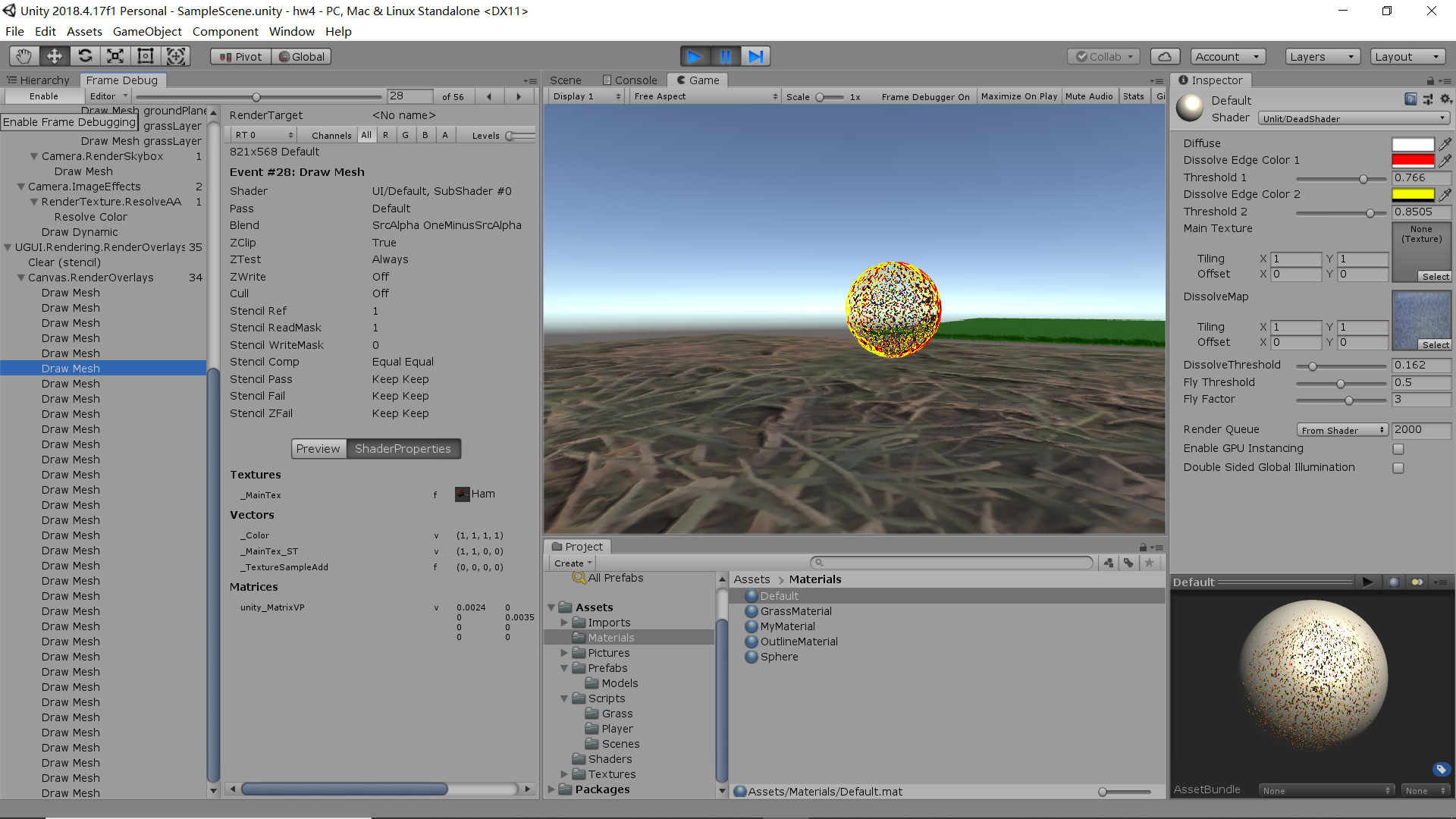
在Channels中可以选择颜色输出通道





可以看到，第一块草地的渲染在第17个DrawCall中进行。





UGUI是在最后渲染的，可以看到所有UI控件都使用UI/Default shader的第一个SubShader进行渲染。

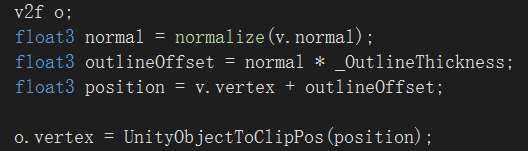
## 复杂shader的实现过程

除了基本shader外，我另外实现了三个shader：

1. OutLine shader：可以用于实现对物体的描边的着色器，实现过程参考

<https://www.ronja-tutorials.com/2018/07/21/hull-outline.html>

其基本实现原理为：通过两个Pass对物体进行渲染。其中一个Pass正常绘制物体；另一个Pass则在原物体的每个顶点上加上一个沿其法向量的偏移，然后进行绘制



同时通过Cull Front来剔除第二个Pass的前向面，因为第二个Pass绘制的物体比正常绘制的要大（法向量偏移造成绘制物体的膨胀），所以剔除前向面后可以同时看到原物体和绘制出来的边缘。其中用户可以通过调整Outline Color来调整边缘的颜色，调整Outline Thickness来调整边缘的大小。

实现效果：





1. Grass shader：模拟草地的着色器，实现过程参考

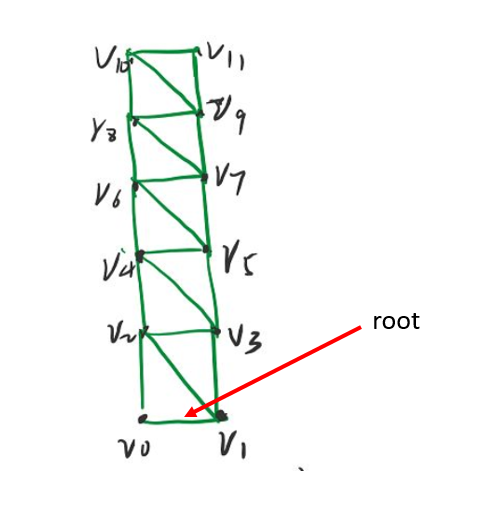
<https://zhuanlan.zhihu.com/p/29632347>

<https://github.com/chenjd/Realistic-Real-Time-Grass-Rendering-With-Unity>

其基本实现原理为：

a、一根草的绘制

通过一个root点来模拟草的根节点，在几何着色器中从传入的root节点开始，绘制如图的片元



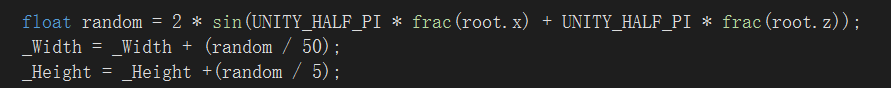
草的颜色通过读入如下纹理模拟：



草的轮廓通过读入如下纹理，将读入颜色的r（g或者b都可以）作为片段着色器输出的alpha值来模拟：



草的高度和宽度通过shader中的变量来控制，同时在绘制过程中通过



为不同根节点的草设置不同的高宽度。

这样就可以绘制出一根草了：

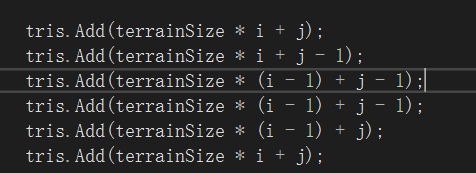


b、草地的绘制

上面已经可以成功地绘制一根草，那么批量生成不同的root节点，就可以绘制出一片草地了。同时为了保证草是依附在ground上的，因此首先要绘制出一个ground，然后在这个ground上生成不同的root节点，进而绘制出一片草地。

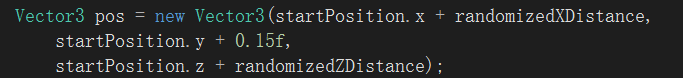
Ground和root节点的生成通过C#脚本GrassSpawner实现：

其中ground通过一个mesh网格来绘制，mesh由一个个三角形片元组成，通过terrainSize来控制地形的尺寸



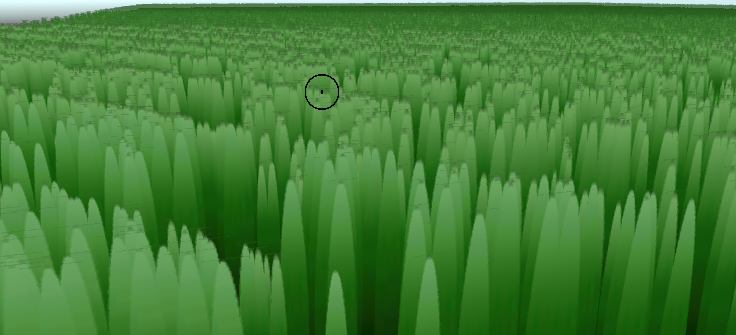
Root节点的绘制以patch为单位，通过变量来控制一个patch中草的根节点的数量，通过变量来控制每行/每列（绘制的地形为正方形）patch的数量，同时通过在生成的根节点的位置上加上随机变量来保证生成root节点位置的随机性（ground上每个顶点的y值相同，每个root节点的y值为ground的y值加上一个小偏移量，如果不加上偏移会出现草根穿过ground的现象）





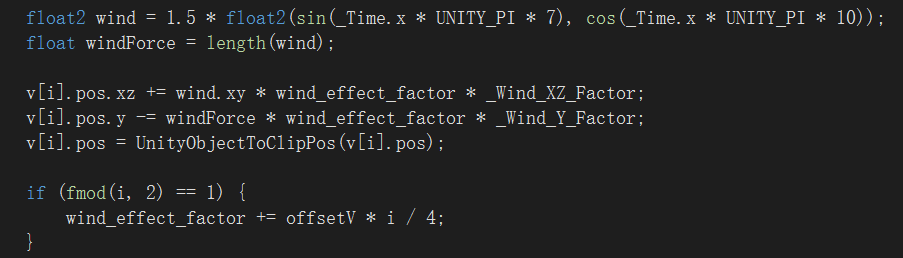
Pos即每个根节点的位置，startPosition为每个patch左下角的坐标。

绘制结果如下：



c、模拟风对草的作用

上面的草暂时还不会摆动，现在来为它添加被风吹动的效果。添加风的效果实际上就是在草的每个顶点上添加位移来模拟风的力，同时注意到，真实的风对同一根草的不同位置的影响程度是不同的，越往上，草受风的影响程度越大，即顶点坐标的偏移越大。通过如下代码来模拟风对草的影响



其中wind\_effect\_factor为风的影响系数，fmod（i，2）==1代表绘制顶点的高度增加（参考一根草的绘制），随着高度的增加，wind\_effect\_factor非线性增加，从而实现越往上，风对草的影响越大，达到草的弯曲效果。教程中风的随机模拟那一块我暂时看得不是很懂，因此先实现所有草统一风向的效果。同时通过三角函数加上shader内置的时间\_Time来实现风向的变化，效果见grass.mp4。

1. Dead shader：模拟物品消亡的着色器，实现过程参考

<https://blog.csdn.net/puppet_master/article/details/72455945>

其实现的基本原理为：

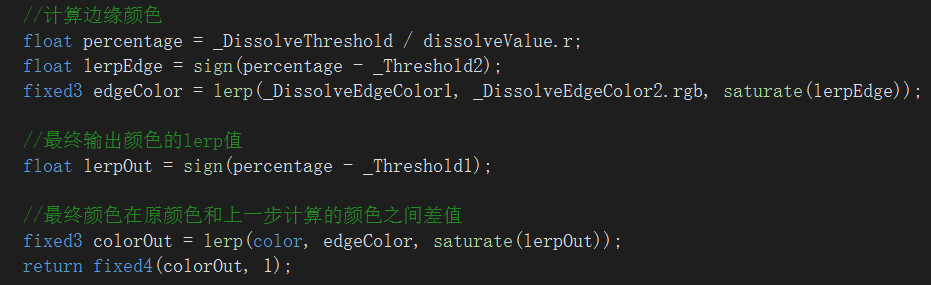
a、溶解效果的实现

设置一个溶解阈值，读入一个噪声图，如果读取到的噪声颜色的r值小于阈值，那么在fragment shader中discard掉对该片元的渲染。通过脚本控制阈值在0和1之间波动，因此可以模拟物体逐渐溶解和逐渐复原的动画。实现效果如下：

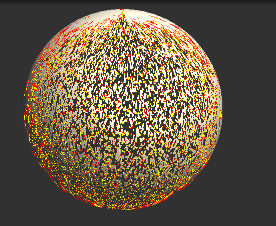


b、溶解过程中颜色插值

溶解过程中根据溶解程度的不同，在设置好的两种溶解边缘色和原色之间进行插值



Color为原色，\_DissolveEdgeColor1和\_DissolveEdgeColor2为两种不同的边缘色，\_Threshold1和\_Threshold2为两种颜色的溶解阈值，即：当溶解程度小于\_Threshold1时，为原色；当溶解程度大于\_Threshold1小于\_Threshold2时，为\_DissolveEdgeColor1；当溶解程度大于\_Threshold2时，为\_DissolveEdgeColor2。实现效果如下



c、碎片向上消散效果

溶解到一定程度后，碎片开始向上（世界坐标）消散



其中\_FlyThreshold为消散的阈值，即当消散程度大于\_FlyThreshold时，碎片开始向上运动；\_FlyFactor控制运动程度。效果见dead.mp4。

## 五、游戏操作指南

WASD和鼠标控制移动；

按B键弹出背包，可以选择其中的模型进行放置；再按B键或者鼠标右键返回游戏；

鼠标中央的准心在碰到模型时会变绿：

按下鼠标左键，可以对模型的着色器进行选择，在着色器选择界面可以按下鼠标右键返回游戏；

按下鼠标中键，销毁模型；

按下鼠标右键，恢复模型默认的着色器。