作业4-Simple Shader作业报告

# Shader实现过程

在unity中创建自定义的Material之后，就可以对Material应用自己写的Shader。

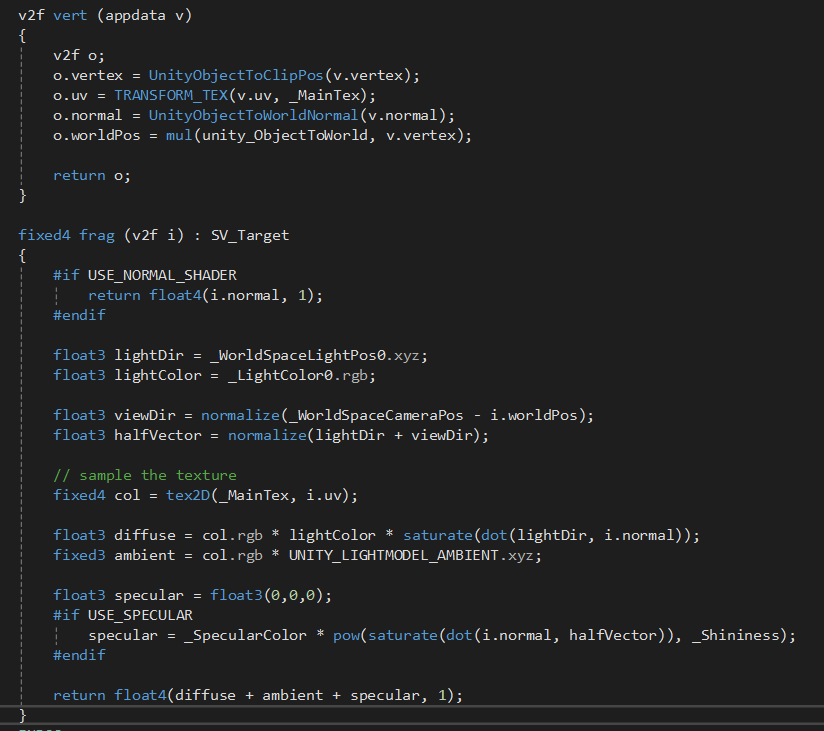
此部分演示的Shader为实现了基于模型法线显示和基于Blinn-Phong显示的基本材质着色器。

代码部分，大体继承于Unity原生的Unlit Shader的框架。



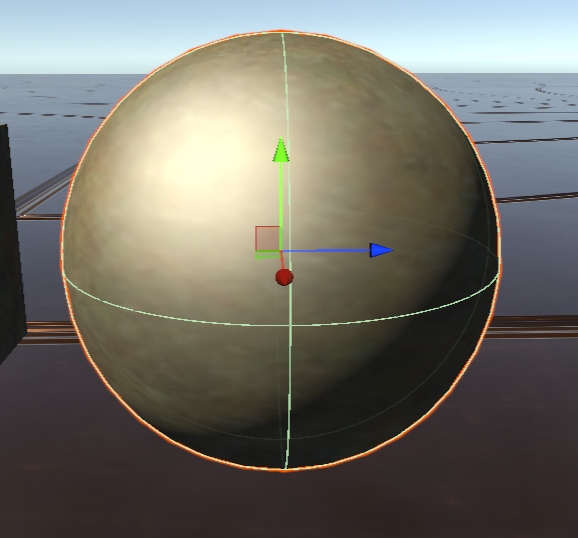
在Shader部分定义了着色器的名称，在Properties部分定义了本着色器中需要从GUI中获取的属性与名称，并在Sub Shader中定义了需要执行的部分。其中Tags和LOD为这个Sub Shader的配置属性，而PASS则代表了渲染的一个管道。由于这个着色器较为简单，因此只使用了一个管道。在管道中，通过CGPROGRAM和CGEND来框出着色器的CG程序。

CG程序中定义了需要使用的变量，以及渲染所使用的顶点着色器和片段着色器。



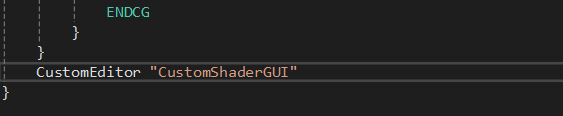
在这个Shader中，顶点着色器负责的是获取点的坐标、法线、纹理的UV坐标和点的世界坐标，并传给片段着色器。片段着色器中则根据这些数据以及UnityCG和UnityStandardBRDF库中的函数来获取光照相关的参数来帮助计算。对于根据法向量显示的部分，只要返回点的法向量即可。对于光照的材质显示，则是在计算了环境光、漫反射光和镜面反射光后，将结果相加返回。其中的环境光和漫反射光由Unity光照系统的参数决定，而镜面反射光则是根据了Blinn-Phong的光照模型来进行计算，返回一个由Shininess系数所影响的镜面反射效果。

最后，将应用了此着色器的材质应用到物体上便能正常显示。

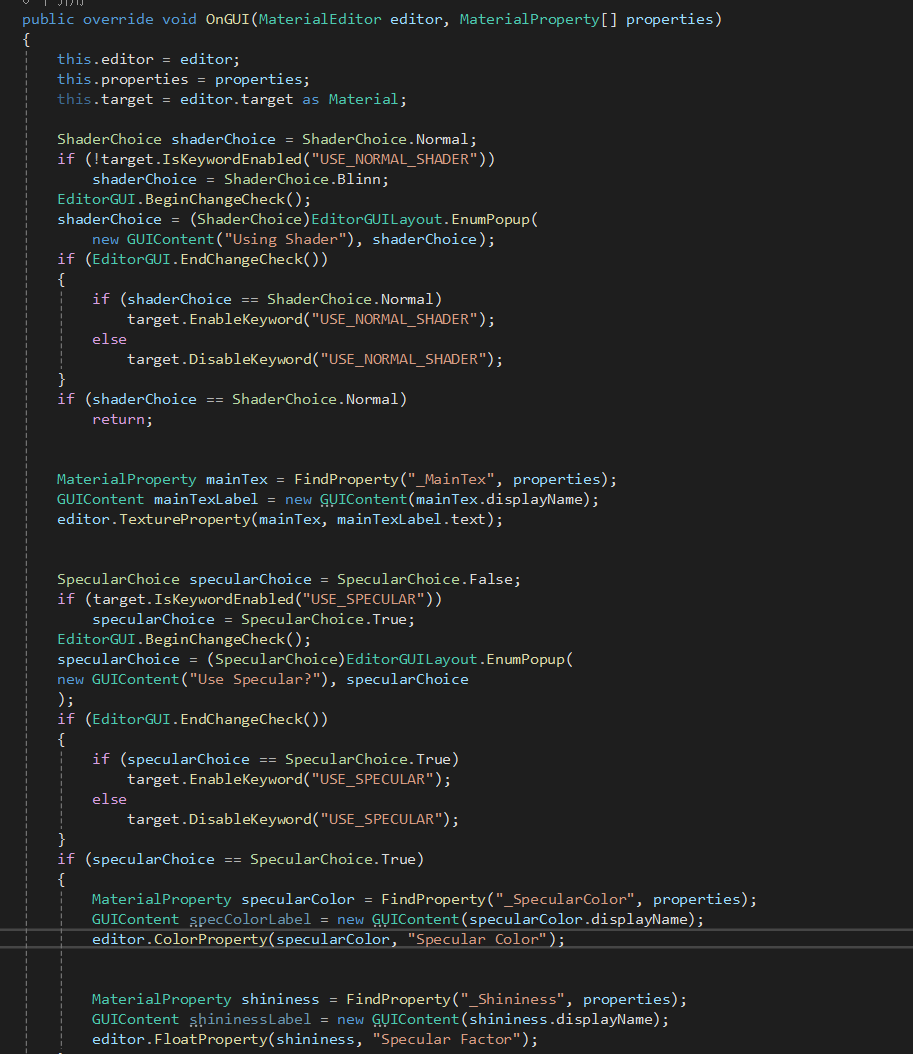


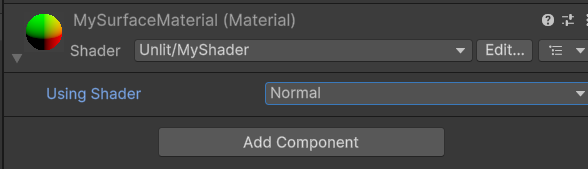
# Shader GUI实现过程

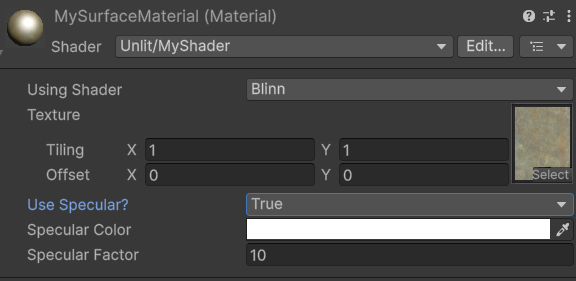
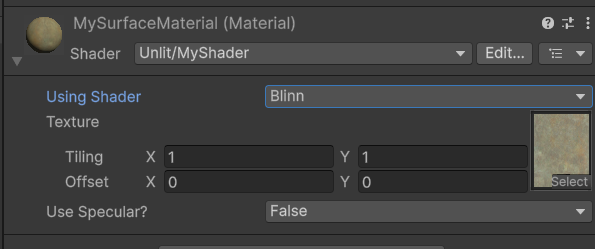
在这个过程中，首先是创建了一个继承于ShaderGUI的customShaderGUI类，并将这个customShaderGUI作为CustomEditor的参数放在着色器的末尾。



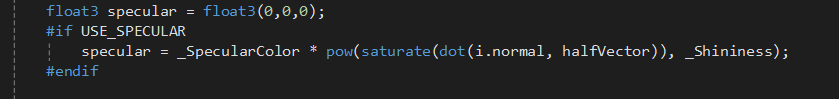
随后，在customGUI部分中的onGUI函数来定义GUI面板中需要的选项和与着色器交互的逻辑。首先初始化editor，properties和target参数。随后对于不同的GUI显示接口，有不同的实现方法。对于大部分属性，显示逻辑为先从Properties中获取到对应名称的属性，然后创建一个GUIContent来绑定，最后把这个GUIContent显示在GUI面板上面。而对于选择框的部分，需要在前后开启检测更改的逻辑代码，即BeginChangeCheck()和EndChangeCheck()。并且这些选择项需要先检测Properties中对应属性的或者对应关键字的状态来显示正确结果，并在更新之后也要根据选项来更改对应的属性和关键字。另外，由于代码执行也是按照正常行数的顺序，因此也可以通过情况来开关某些属性在GUI面板上的显示情况。



当完成之后，就可以在Unity的编辑器中看到自己编辑的GUI了。



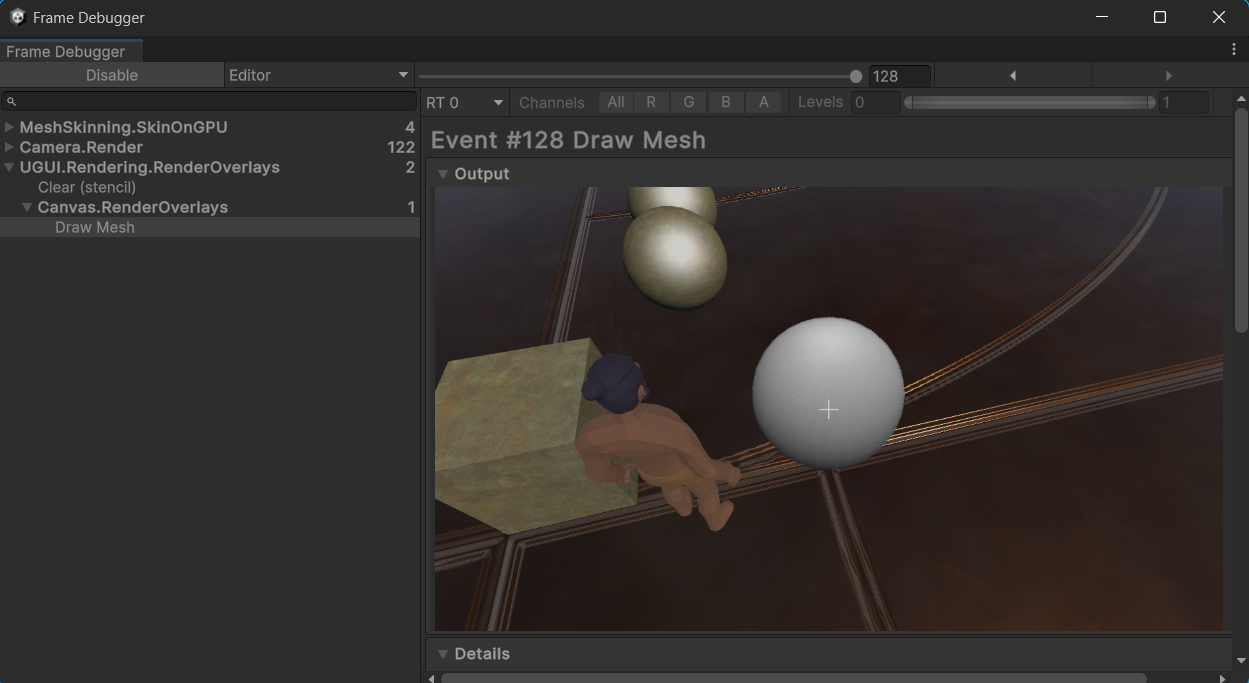
另外，在GUI中修改的关键字在着色器中需要通过#pragma定义着色器特征后就可以使用#if和#endif来开启代码的情况了。



# Debug工具使用过程

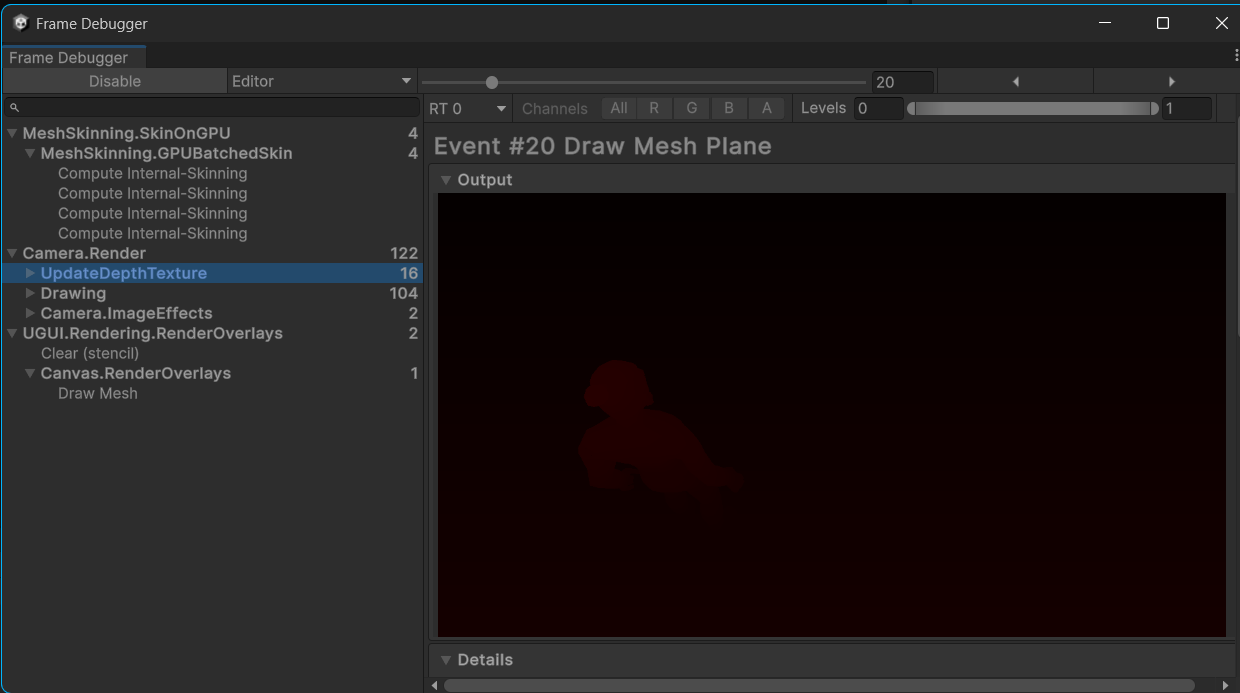
Frame Debugger：

在Windows->Analysis->Frame Debugger打开后就有一个窗口。

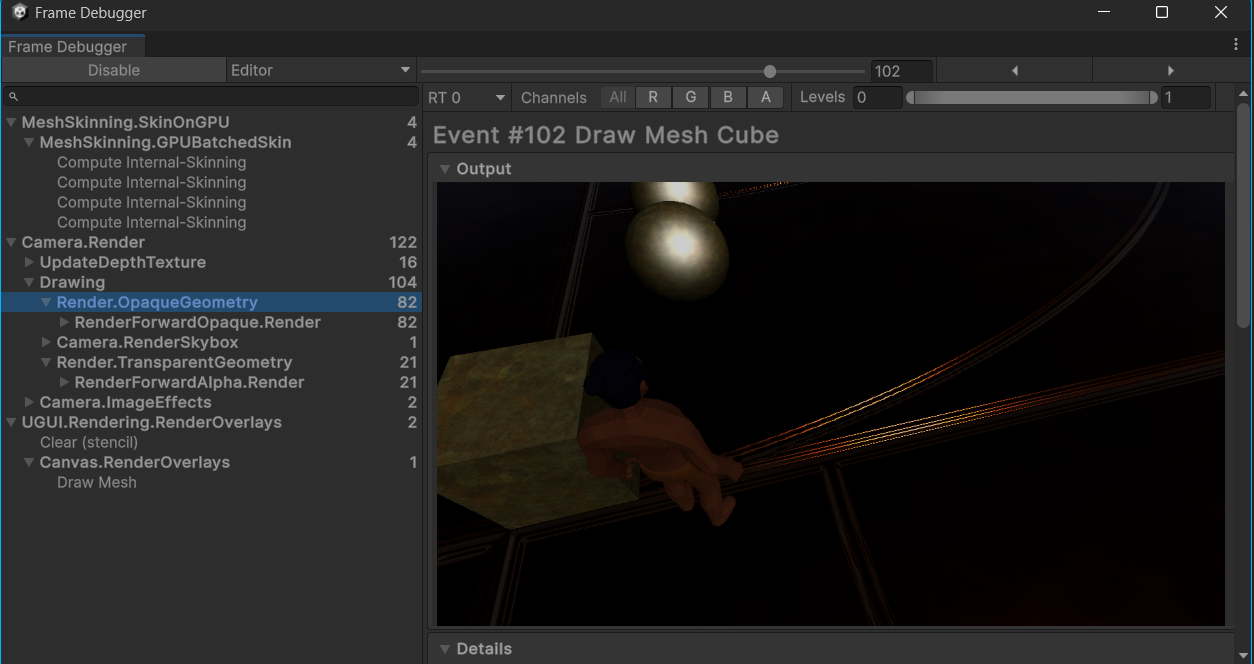


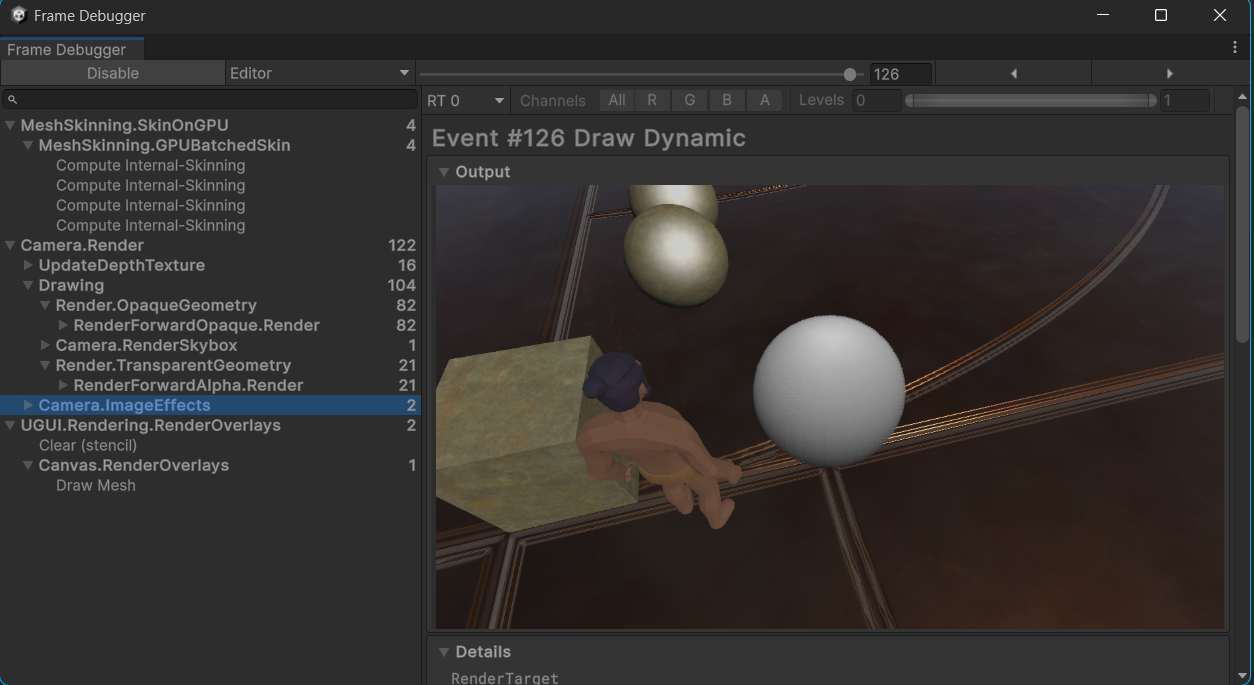
此窗口中，左边显示了在这一帧进行渲染的所有过程，右边则是这一帧进行渲染后的结果。它可将正在运行的游戏的状态冻结到特定帧来自由回放，并查看用于渲染该帧的各个绘制调用。除了列出绘制调用，调试器还可逐个单步执行这些调用，以便详细查看场景是如何从场景的图形元素构建的。

例如若想查看这一帧的这个相机的渲染过程，点开就可以发现为：



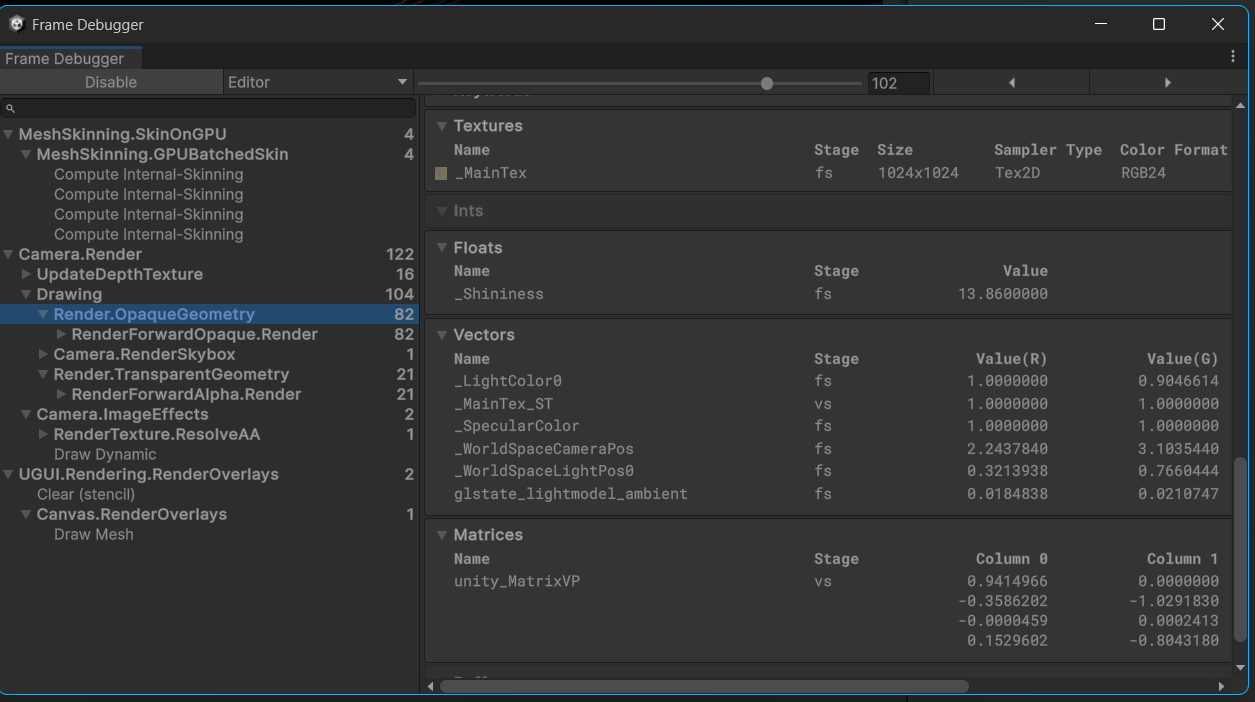
生成深度贴图

依次绘制不透明的，天空盒，然后是透明物体。



添加相机效果

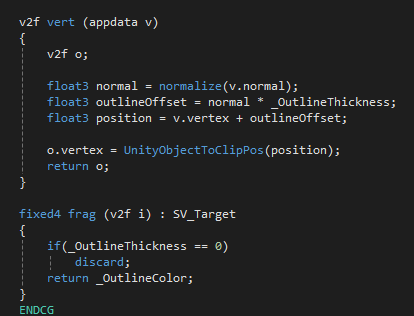
另外，也可以在右边的面板中看到渲染过程中的参数，以便进一步调试。



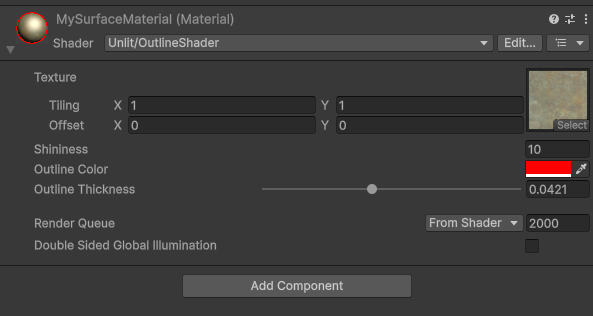
# 复杂Shader实现和使用

1. 描边Shader

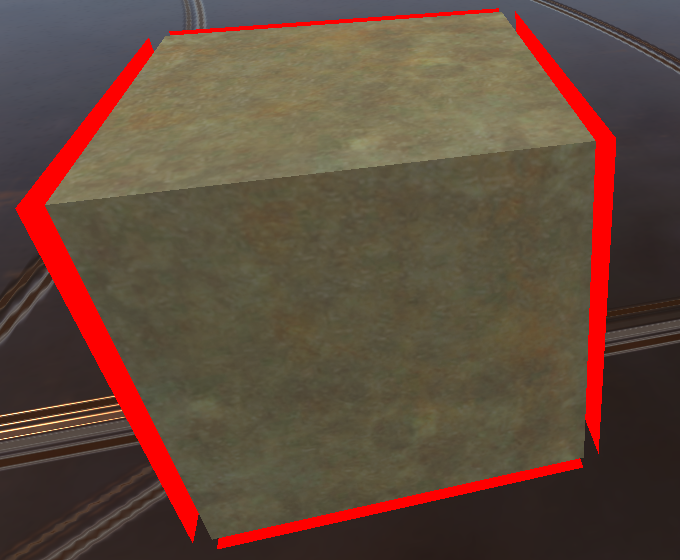
实现方法为在基础的渲染管道之外额外加一个PASS用来渲染外轮廓。该PASS在顶点着色器的部分将每个点沿着法向量向外拓展一定距离，并在片段着色器中直接根据外轮廓颜色进行渲染。为了使这个外轮廓能正确显示在模型外部，需要在这个PASS头部额外添加一个Cull front，用来表示渲染顺序。



这样，就可以正确渲染出圆滑物体的外轮廓，并且也可以在GUI中修改轮廓大小和颜色等属性。



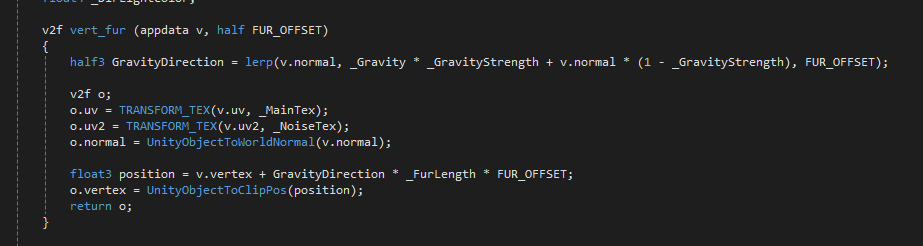
但是，这个方法非常简朴，实现效果也很有限。例如，它需要额外渲染一边原本的模型。最大的问题是，它无法应用于不连续的网格体所组成的物体，或是平面所组成的物体。即，它无法正确显示平面的轮廓。因为平面的边缘部分没有三角片，而且平面的所有法向量都是与平面的法向量同向，因此，用此办法去渲染平面的轮廓时，只能得到另一个距离平面一定距离的平面，而这根本无法在视觉上构成外轮廓。



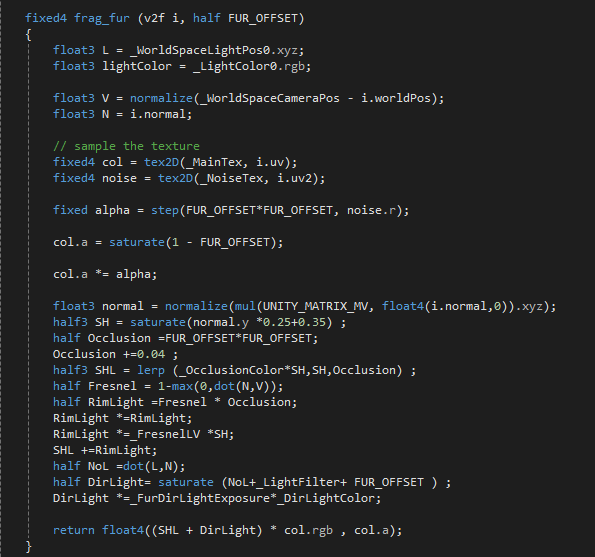
为此，需要采取其他办法，但我目前尚未解决。有一个办法是通过后处理来实现，但后处理相关的知识和着色器实现方法等与目前阶段我所掌握的着色器有较大出入，因此暂时无法理解。另一个办法，是根据深度贴图中的像素梯度来检测模型边缘，但当我进行渲染以及通过Frame Debugger中可以发现自定义的着色器的物体并没有绘制深度贴图，而我也不知道这背后的原因，因此也无法获得深度贴图，所以外轮廓的进一步实现便未能完成。

1. 毛质Shader

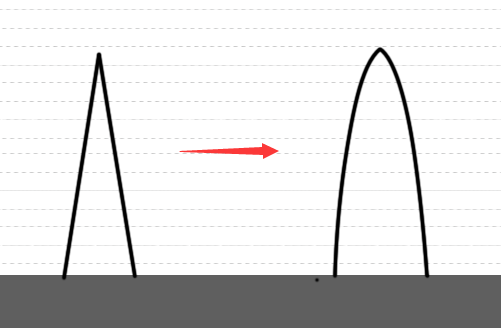
毛的渲染也是由多个管线组成，并且PASS数越多渲染效果越好。它的原理在于，将原本图像根据法向量延伸出来作为毛在表面，另外引入一个噪音贴图来使不同地方的毛产生区别。



在顶点着色器中，每一层根据当前层的FUR\_OFFSET来将顶点延伸出不同的距离，作为这一层的毛的顶点。另外，再计算重力等外力因素来修正位置。

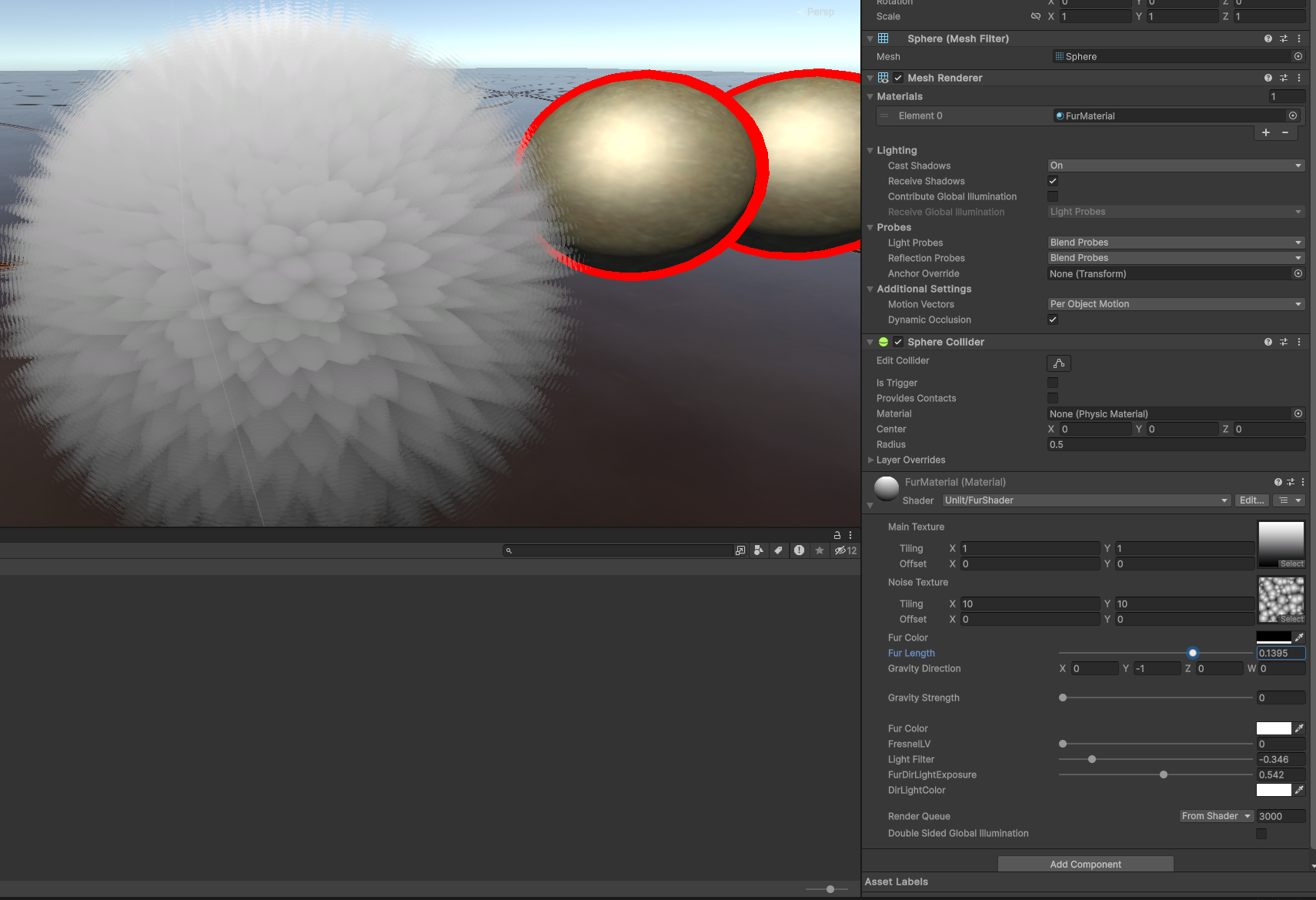


在片段着色器中，同样需要FUR\_OFFSET来决定这一层的毛的渲染情况。首先根据OFFSET\*OFFSET来得到不同的alpha值，这样就可以使毛的形状成为一个较为圆滑的凸出物体。

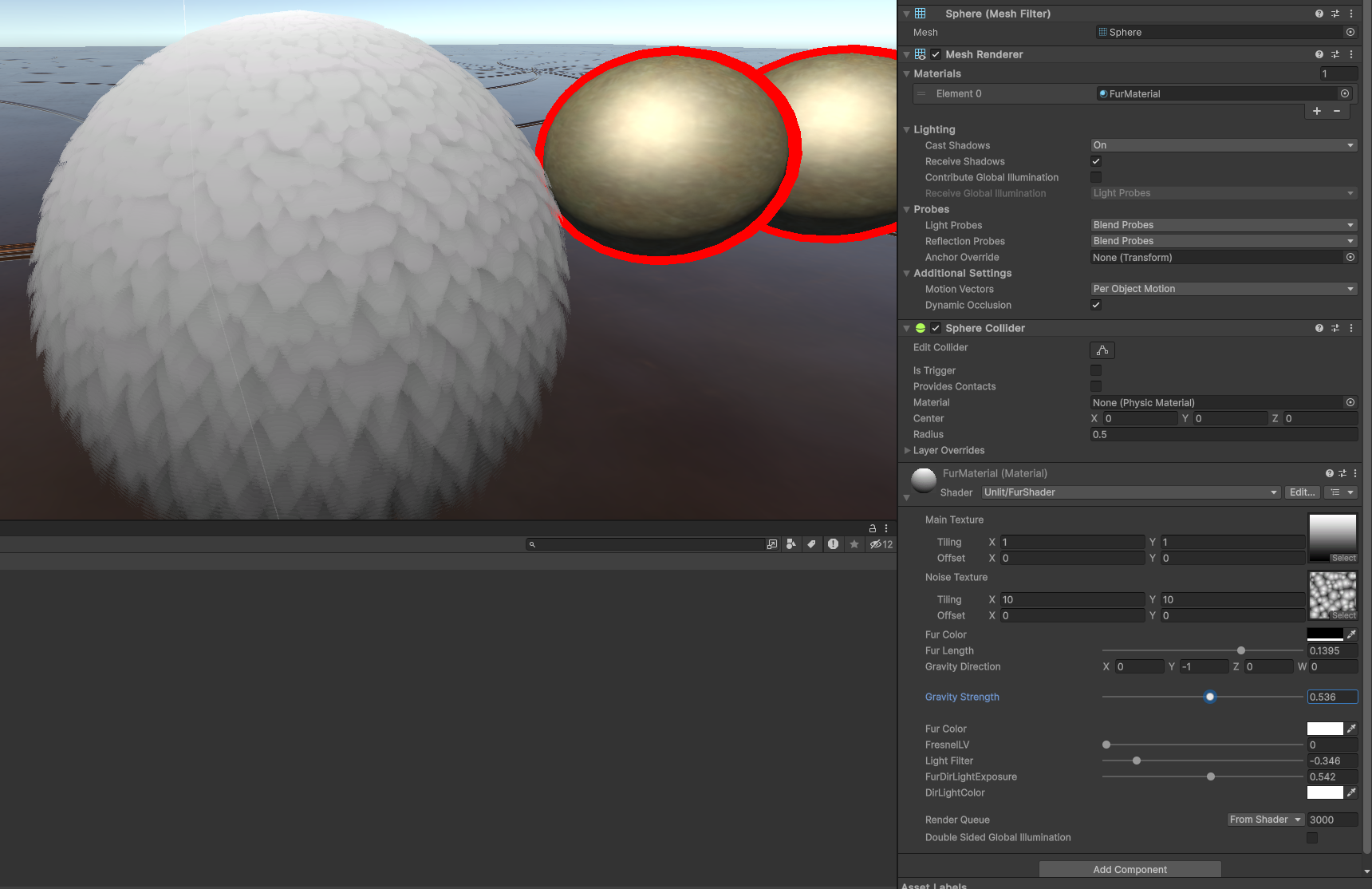


另外，再将其乘以（1-FUR\_OFFSET），来使越是尖端的部分越透明，来贴合毛发的性质。接下来对毛发进行光照计算，依次计算了环境光遮蔽下的环境光、漫反射光和太阳光，并将光照结合，就得到了拥有光照的毛发着色器。

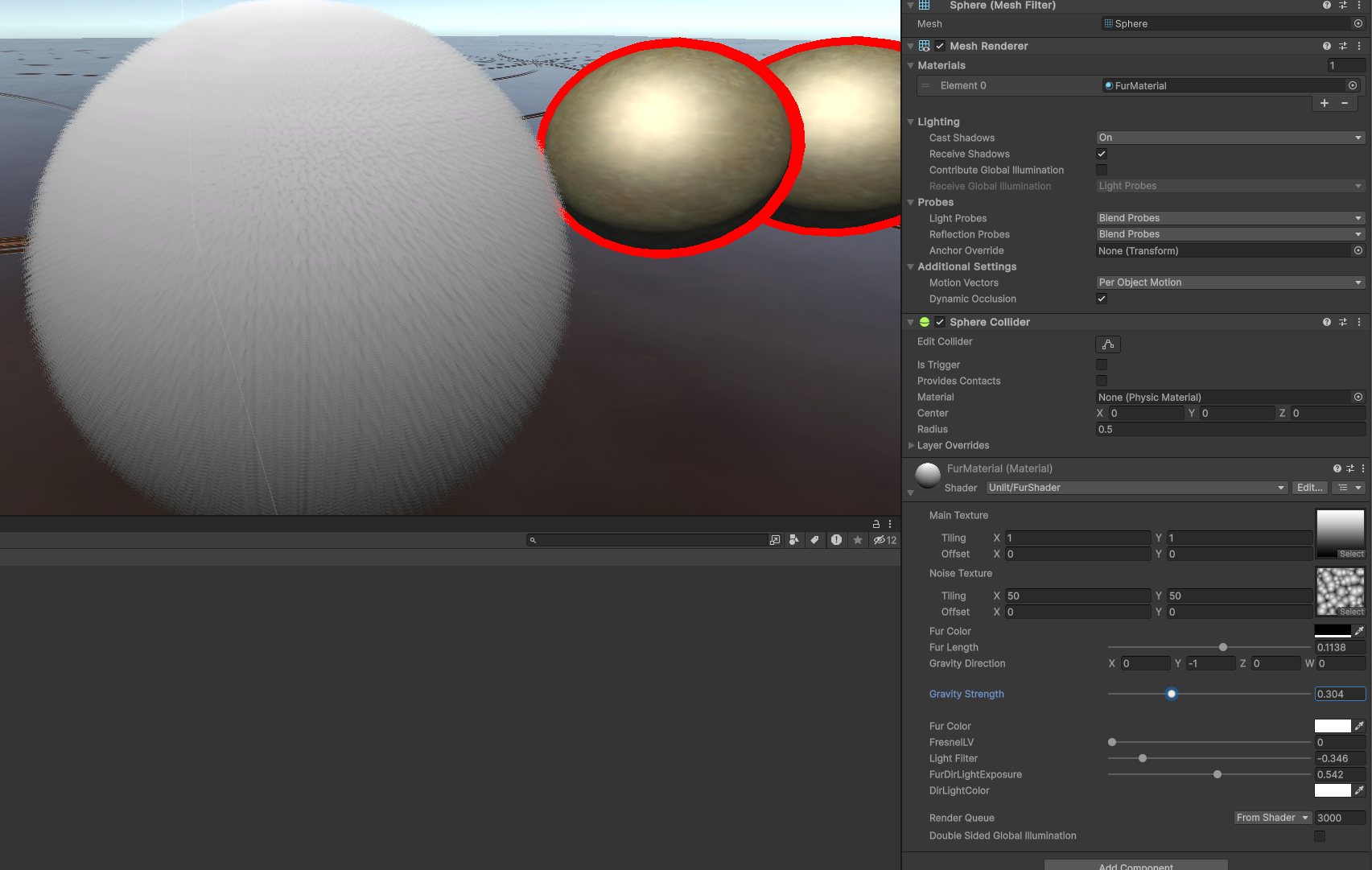
实现效果如下：



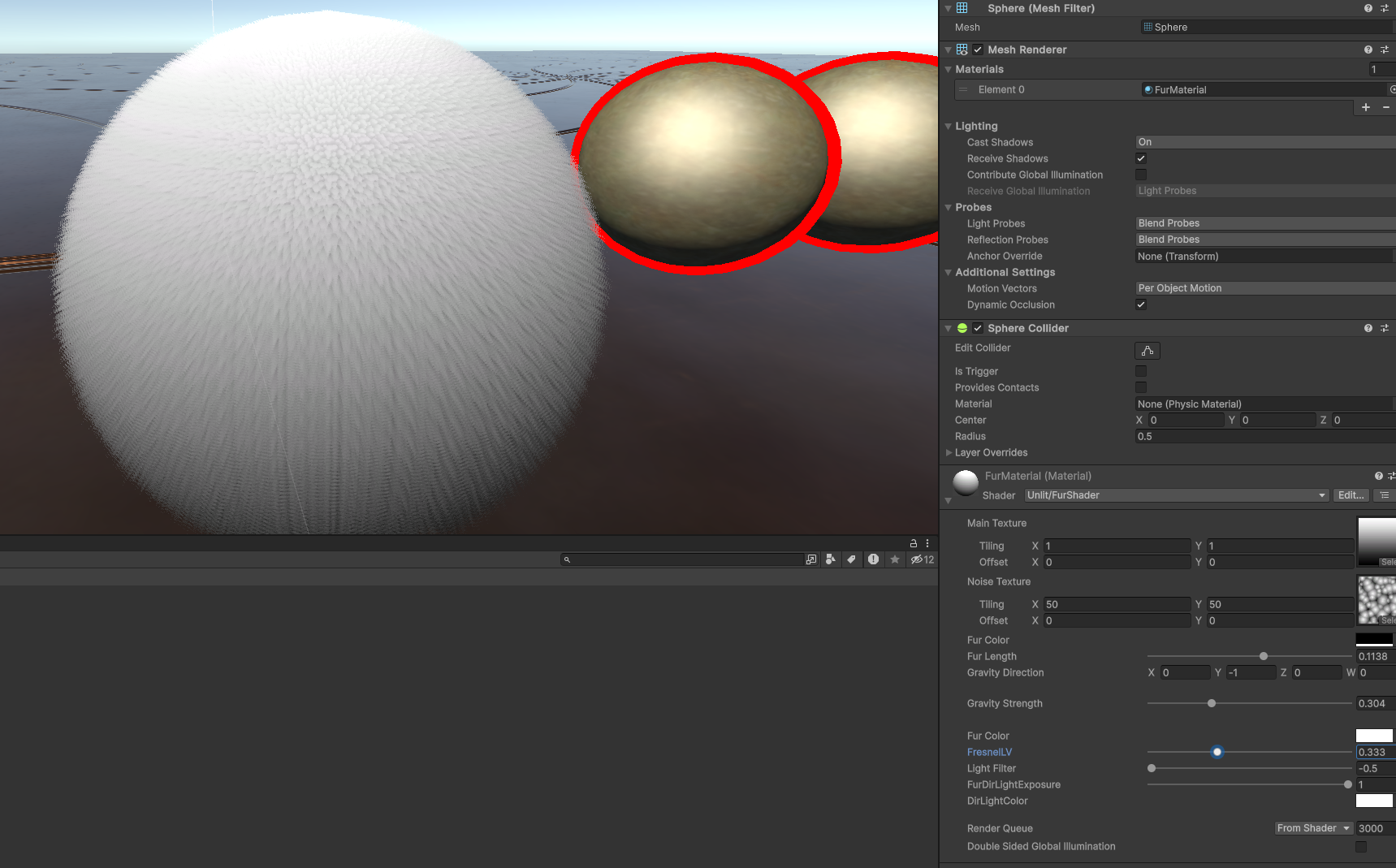
较疏的毛



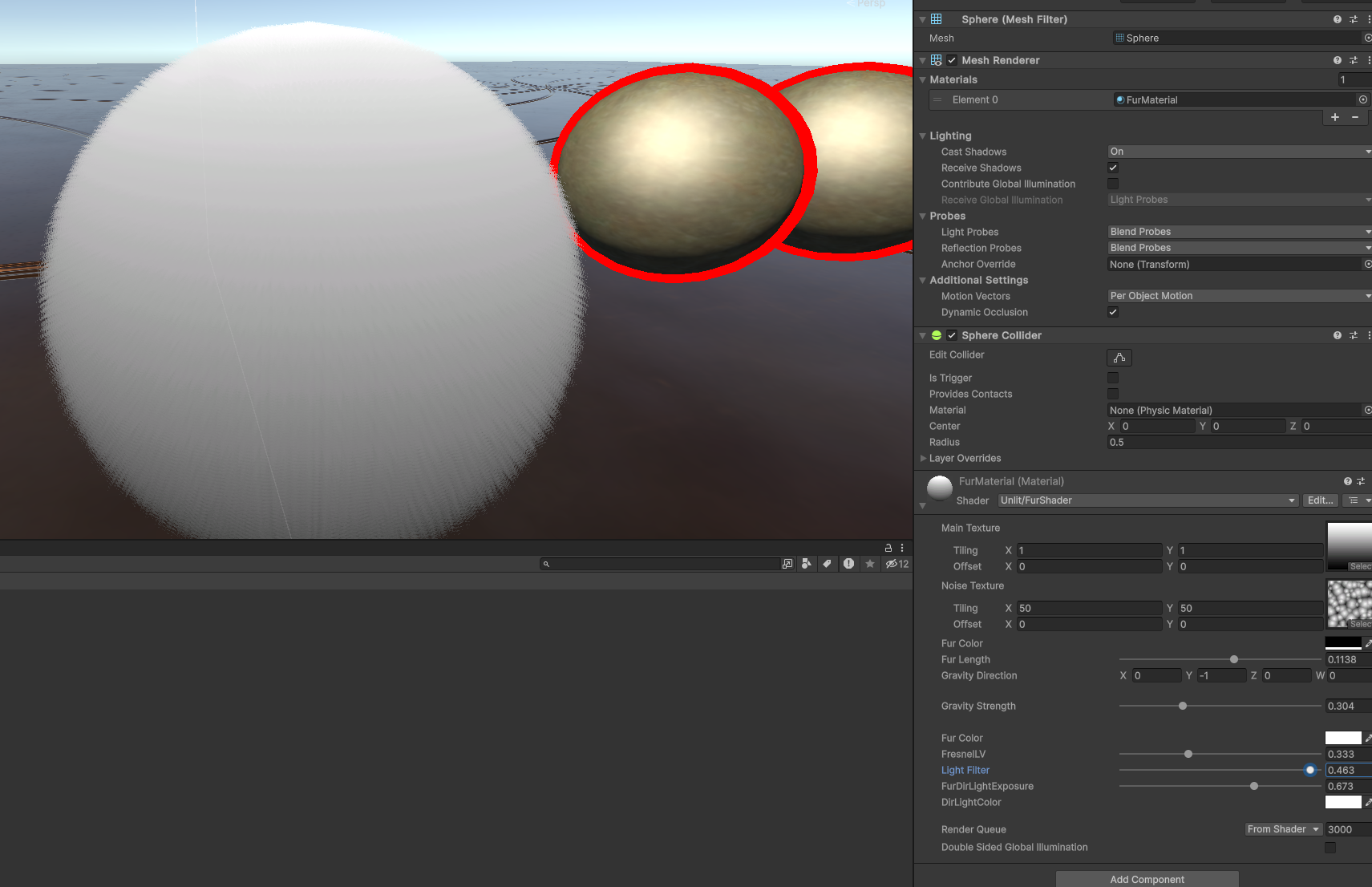
较疏的毛+一定重力



较细的毛+较小的重力



调整光照系数



调整光照系数

# 游戏操作指南

Wasd控制移动，空格跳跃，滚轮和数字键选择方块，右键放置。

（当前的运动系统有点问题，人物跳跃功能实现尚不完善）

另外，我在导出的时候遇到了如下问题，并且经过搜索也没有查找到解决方法。因此在导出的时候，我将customShaderGUI的脚本整个注释掉了。

