

# hw5\_CustomShader作业报告

肖蔚尔 520030910314

## 1. 基于物理的渲染模型

### 1. Cook-Torrance 和双向反射分布函数 ( BRDF )

双向反射分布函数 (BRDF) 描述了一个表面在每个入射光、出射光方向设定下反射了多少光。在渲染方程中，反射部分的公式可以如下表示：

$$L_r(v) = \int_{\Omega} L_i(l) f_r(l, v)(n \cdot l) dl$$

其中函数  $f_r$  即为 BRDF，它反映了从  $l$  入射的光线有多少被反射到  $v$  方向。而公式中对入射光  $l$  的积分则表示所有入射方向光线在一个反射方向  $v$  处反射光的总和。BRDF 被分成漫反射项 (Diffuse) 和镜面反射项 (Specular)。

$$f_r = k_d f_d + k_s f_s$$

Unity 标准 Shader 使用了基于 Cook-Torrance 模型的 BRDF，它将表面看作一系列的微面元。在分子中包含三部分，分别是 Facet slope distribution term  $D$ 、Fresnel term  $F$  以及 Geometrical attenuation term  $G$ 。其中  $D$  描述了微面元上的朝向分布，表现了局部的反射效果；而  $F$  描述了菲涅尔效应的程度； $G$  则描述了微面元之间的几何遮挡因素。

$$f(l, v) = \begin{cases} k_d f_d + k_s \frac{D(h) F(v, h) G(l, v, h)}{4(n \cdot l)(n \cdot v)}, & \theta < 90^\circ \\ 0, & \text{else} \end{cases}$$

Shader 中已经实现了函数 `GetUnityIndirect`，它利用 Unity 内置的 `UnityGlobalIllumination` 函数获取当前全局光照，并得到间接光照，存放在 `UnityIndirect gi` 中。通过间接光照，可以根据周围环境，渲染出真实的反射效果。

### 2. 代码实现&效果展示

需要按照 Cook-Torrance 模型分别实现其中的  $D$ 、 $F$  和  $G$  项。

对于  $D$  项，我们使用比较常用的 GGX 算法，它是由 Walter 等人提出的，其形式如下：

$$D_{GGX} = \frac{\alpha^2}{\pi((n \cdot h)^2(\alpha^2 - 1) + 1)^2}$$

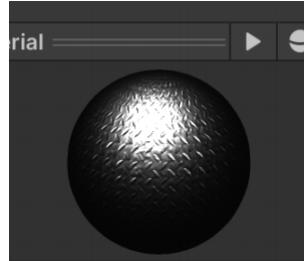
其中  $\alpha$  是粗糙程度，即代码中的 `roughness`。

```

float GGX_D(float roughness, float NdotH)
{
    float rough = roughness;
    float nh = NdotH * NdotH;
    float down = nh * (rough * rough - 1) + 1;
    float result = rough * rough / (3.14159 * down * down);
    return result;
}

```

当参数 Smoothness = 0.5 时，可以看到高光效果：



对于  $F$  项，为了更好地得到效果，我们采用和 Unity 内置 Shader 一样的做法，即首先利用 DiffuseAndSpecularFromMetallic 计算高光颜色 specColor，再用这个值作为参数代替  $R$  在 Schlick 模型中使用。

$$F_{Schlick} = C_{spec} + (1 - C_{spec})(1 - l \cdot h)^5$$

```

float3 schlick_F(half3 R, half cosA)
{
    return R + (1 - R) * pow((1 - cosA), 5);
}

```

对于  $G$  项，使用 Cook-Torrance 的模型。

$$G_{Cook-Torrance} = \min \left( 1, \frac{2(n \cdot h)(n \cdot v)}{v \cdot h}, \frac{2(n \cdot h)(n \cdot l)}{v \cdot h} \right)$$

```

float CookTorrence_G (float NdotL, float NdotV, float VdotH, float
NdotH){
    float p2 = 2 * NdotH * NdotV / VdotH;
    float p3 = 2 * NdotH * NdotL / VdotH;
    return min(min(1, p2), p3);
}

```

效果如下，可以看出几何遮挡：



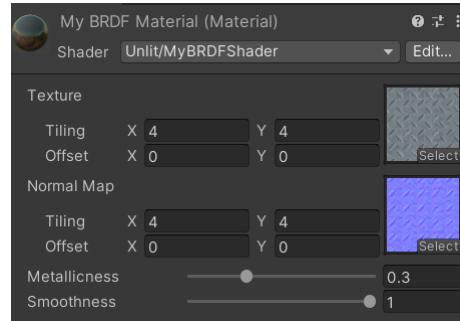
实现了上述的功能，我们便完成了一个 Cook-Torrance BRDF 模型。之后使用

```
float3 directSpecular = (D * F * G) / (4 * (NdotL * NdotV));
```

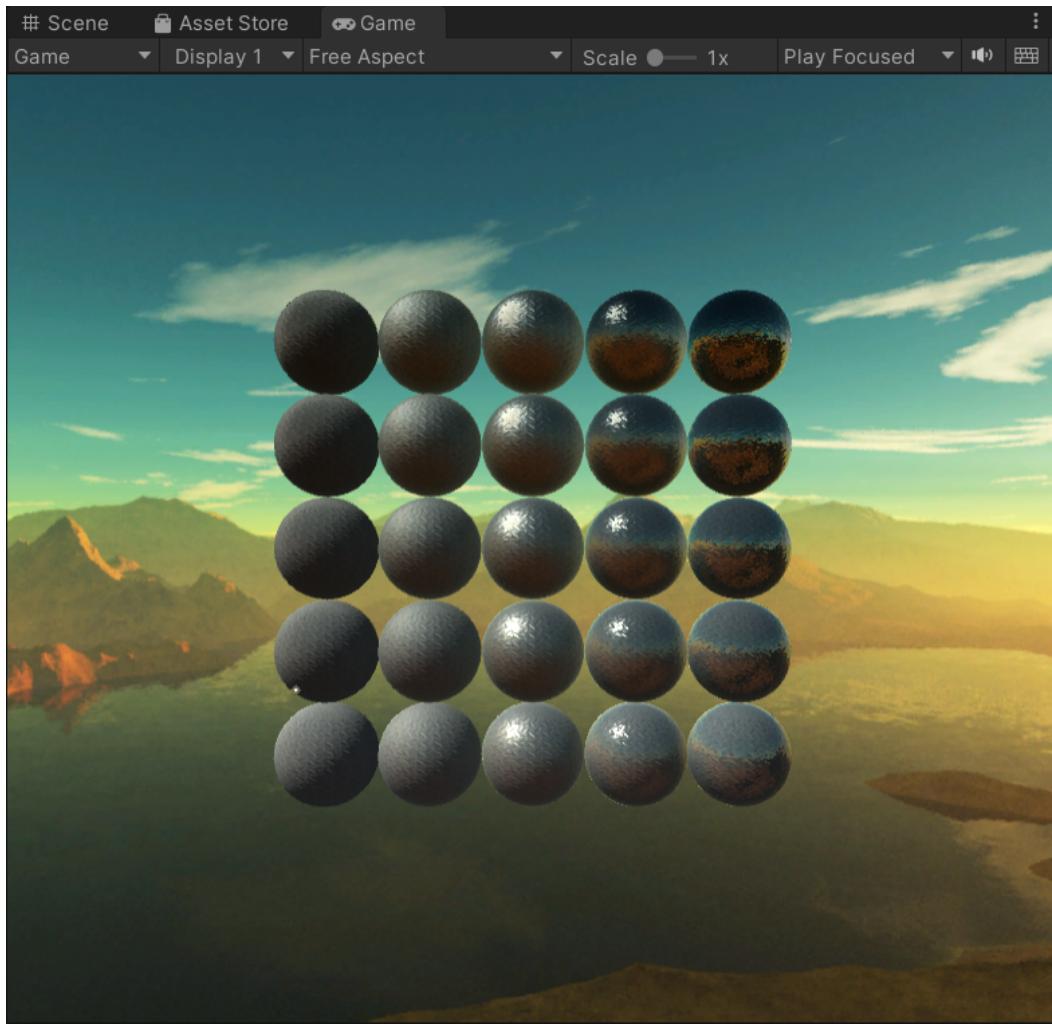
便可以得到直接光照的高光部分的反射光。这里为了得到更明显的高光效果，可以加上系数  $\pi$

```
float3 directSpecular = (D * F * G) * UNITY_PI / (4 * (NdotL * NdotV));
```

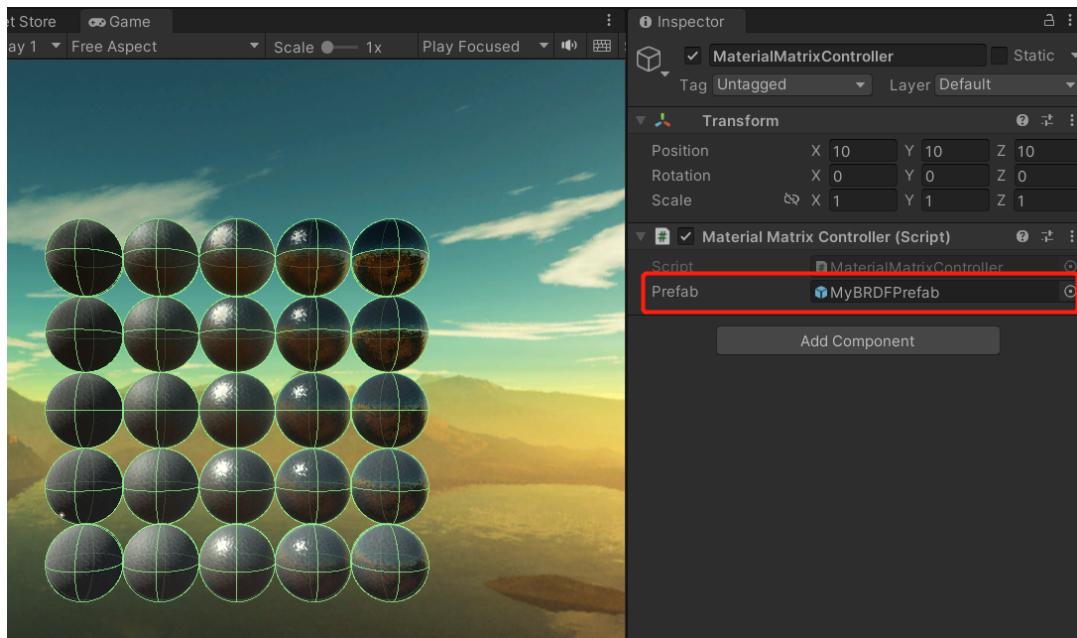
之后我们分别计算整体的直接光照和间接光照，并最终输出颜色。通过逐渐调整参数如下：



最终可以得到如下场景：



材质球矩阵图Prefab绑定：



## 2. 非真实感渲染 (卡通风格渲染)

- 实现原理：原理是把 diffuse 漫反射颜色简化成对比较明显的 2 阶色阶。Diffuse 模型中，法线方向向量与光线方向向量的点积控制着漫反射的强度。
- 设置属性：RampThreshold 色阶阈值，增加属性：RampSmooth 色阶间平滑度。使用 smoothstep 平滑函数，根据 RampSmooth 对色阶之间进行过渡。直接用颜色叠加作为阴影和高光。设置属性 HColor 高光颜色和 SColor 阴影颜色。

```

fixed3 ramp = smoothstep(_RampThreshold - _RampSmooth * 0.5,
_RampThreshold + _RampSmooth * 0.5, nd1);
ramp *= atten;
s.Albedo = lerp(s.Shadow, s.Albedo, ramp);
_sColor = lerp(_HColor, _sColor, _sColor.a);
float3 rampColor = lerp(_sColor.rgb, _HColor.rgb, ramp);
...
fixed3 diffuse = s.Albedo * lightColor * rampColor;

```

- 设置镜面光照的相关属性：SpecColor 高光颜色、SpecSmooth 高光色阶的平滑度、Shininess 镜面反射度。  
surf 着色器里，使用纹理的 alpha 通道作为光泽度 Gloss。设置边缘光的属性：RimColor 边缘光颜色、RimThreshold 边缘光阈值、RimSmooth 边缘光色阶的平滑度。法线方向与视线方向夹角越小，与光线方向夹角越大，则边缘光强度越强。

```

void surf(Input IN, inout SurfaceOutput o)
{
    ...
    o.Specular = _Shininess;
    o.Gloss = mainTex.a;
}

```

```

inline fixed4 LightingToon(SurfaceOutput s, half3 lightDir, half3
viewDir, half atten)
{
    ...
    float ndv = max(0, dot(normalDir, viewDir));

```

```

    ...
    float rim = (1.0 - ndv) * ndl;
    rim *= atten;
    rim = smoothstep(_RimThreshold - _RimSmooth * 0.5, _RimThreshold +
    _RimSmooth * 0.5, rim);
    ...
    fixed3 rimColor = _RimColor.rgb * lightColor * _RimColor.a * rim;
    color.rgb = diffuse + specular + rimColor;
    color.a = s.Alpha;
    return color;
}

```

4. 对于模型不同部分，单纯用颜色来做阴影可能缺少层次感，可以考虑使用纹理，对不同部分添加不同的阴影颜色。这里需要自定义 SurfaceOutput，添加 Shadow 保存纹理采样的颜色。

```

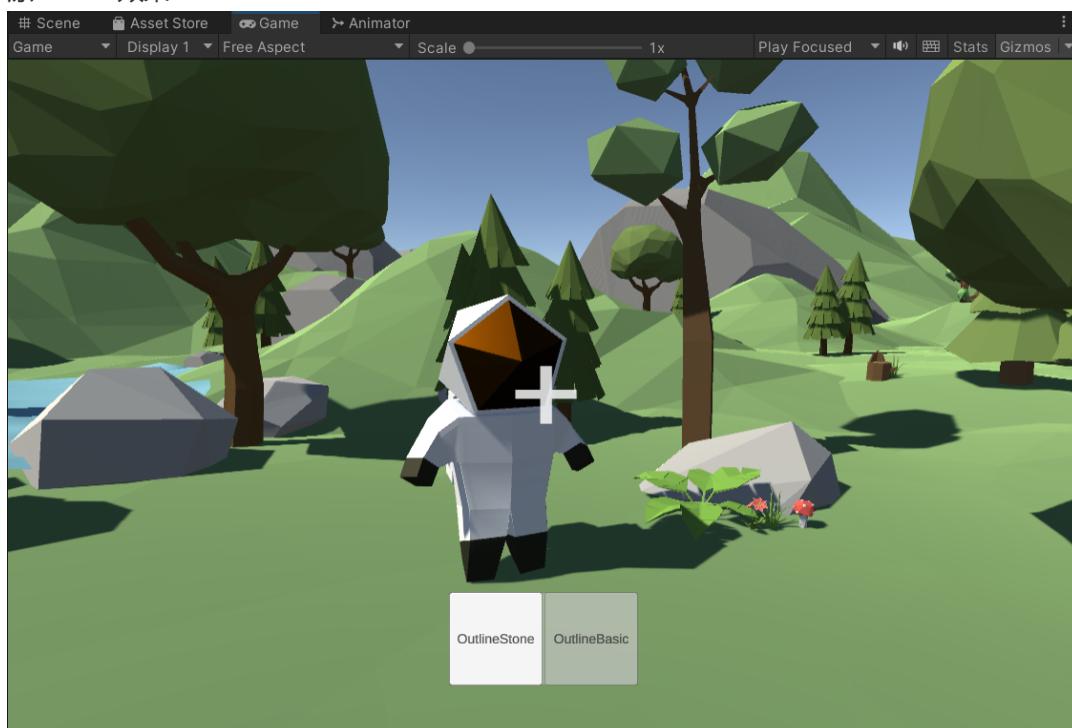
void surf(Input IN, inout SurfaceOutputCustom o)
{
    ...
    fixed4 shadowTex = tex2D(_ShadowTex, IN.uv_MainTex);
    o.Shadow = shadowTex.rgb;
    ...
}

inline fixed4 LightingToon(SurfaceOutputCustom s, half3 lightDir, half3
viewDir, half atten)
{
    ...
    s.Albedo = lerp(s.Shadow, s.Albedo, ramp);
    ...
}

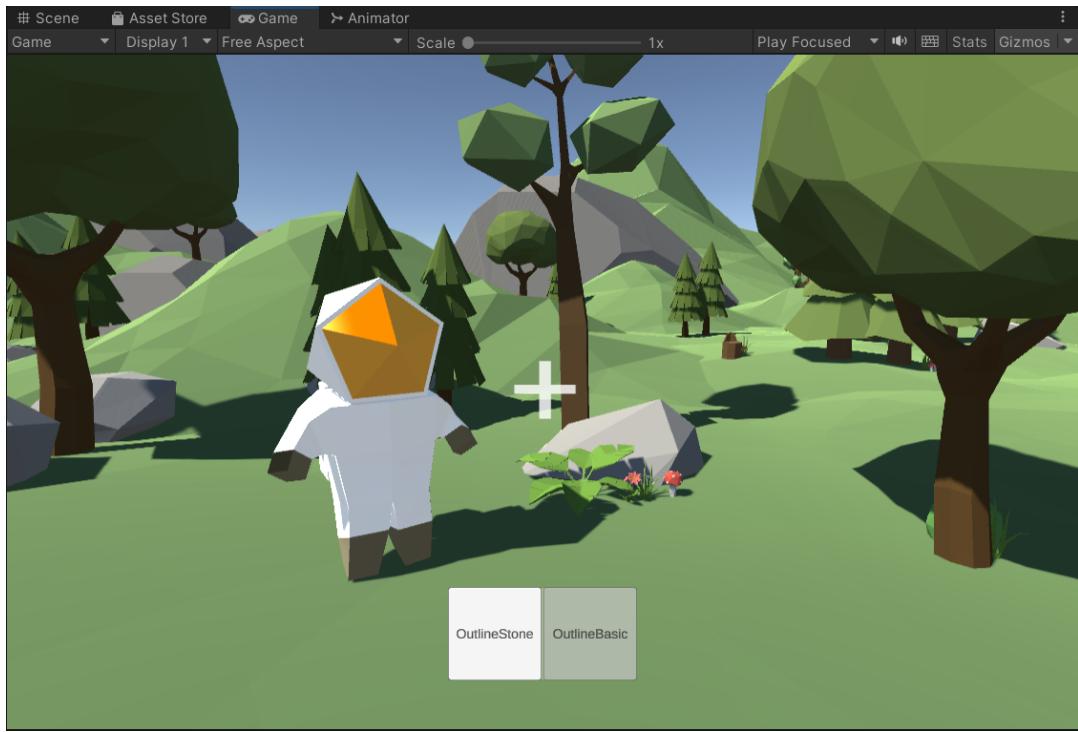
```

效果如下：

原shader效果：



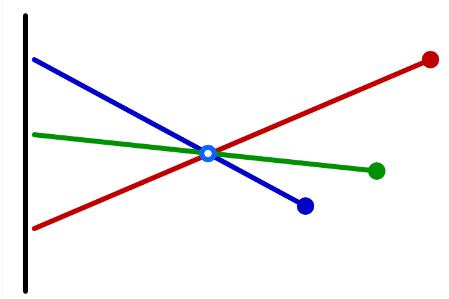
### 卡通shader效果：



### 3. 屏幕后处理效果 (景深)

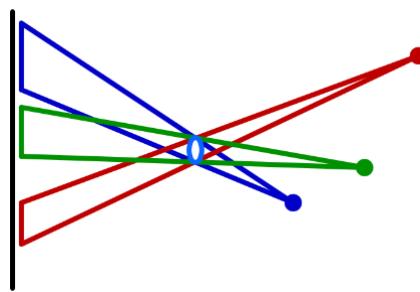
相机可以记录光线，因为光子会击中它们的胶片或图像传感器。在所有情况下，光线都会聚焦以产生清晰的图像，但并非所有内容都可以同时聚焦。只有一定距离的物体才能聚焦，而更近或更远的事物似乎都失焦了。这种视觉效果称为景深。失焦投影的外观细节被称为散景，散景可以用来很好地引导观众的注意力，它是一种艺术工具。下面将创建类似于 Unity 后期效果堆栈 v2 中的景深效果。

最简单的相机形式是完美的针孔相机。像所有相机一样，它有一个图像平面，光线在其上投射和记录。在图像平面的前面是一个小孔 - 称为光圈 - 刚好足够允许一束光线穿过它。相机前的物体会向多个方向发射或反射光线，从而产生大量光线。对于每个点，只有一条射线能够穿过孔并被记录下来。

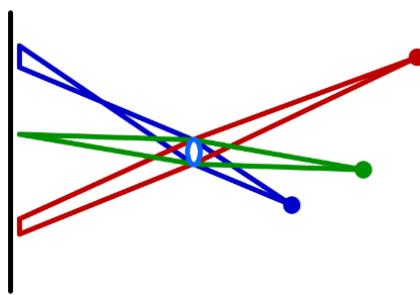


由于每个点仅捕获一束光线，因此图像始终清晰。不幸的是，单条光线不是很亮，因此生成的图像几乎不可见。您必须等待一段时间才能积累足够的光线才能获得清晰的图像，这意味着这款相机需要很长的曝光时间。当场景是静态的时，这不是问题，但所有移动的东西，哪怕是一点点，都会产生大量的运动模糊。因此，它不是一款实用的相机，不能用于录制清晰的视频。

为了能够减少曝光时间，光线必须积累得更快。唯一的方法是同时记录倍增的光线。这可以通过增加孔径半径来完成。假设孔是圆形的，这意味着每个点都将用光锥而不是线投影到图像平面上。所以我们接收到更多的光，但它不再在一个点上结束。相反，它被投影为光盘。覆盖的区域有多大取决于点、孔和图像平面之间的距离。结果是更亮但未聚焦的图像。

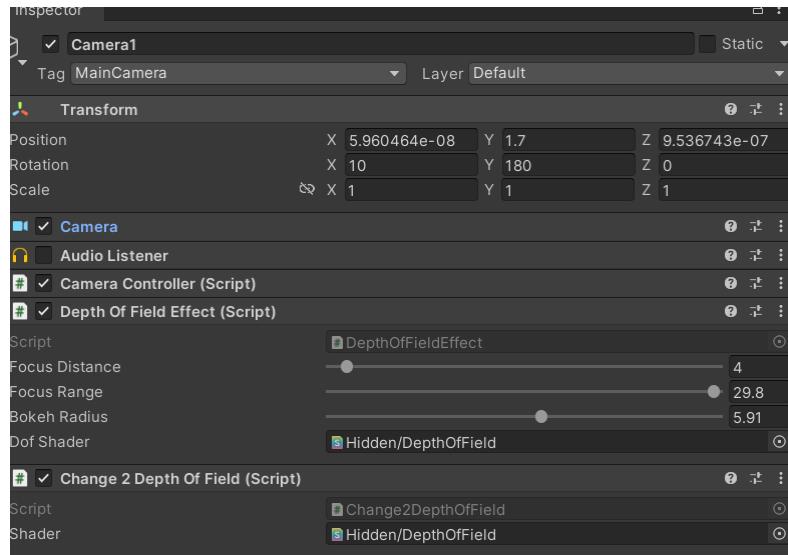


为了再次聚焦光线，我们必须以某种方式获取一个入射的光锥并将其带回一个点。这是通过将镜头放入相机的孔中来完成的。镜头弯曲光线的方式使散射光再次聚焦。这可以产生明亮而清晰的投影，但仅适用于与相机保持固定距离的点。来自较远点的光线聚焦不够，而来自离相机太近的点的光线聚焦太多。在这两种情况下，我们最终都会将点投影为圆盘，它们的大小取决于它们失焦的程度。这种未聚焦的投影被称为混乱圈，简称CoC。



- 步骤：

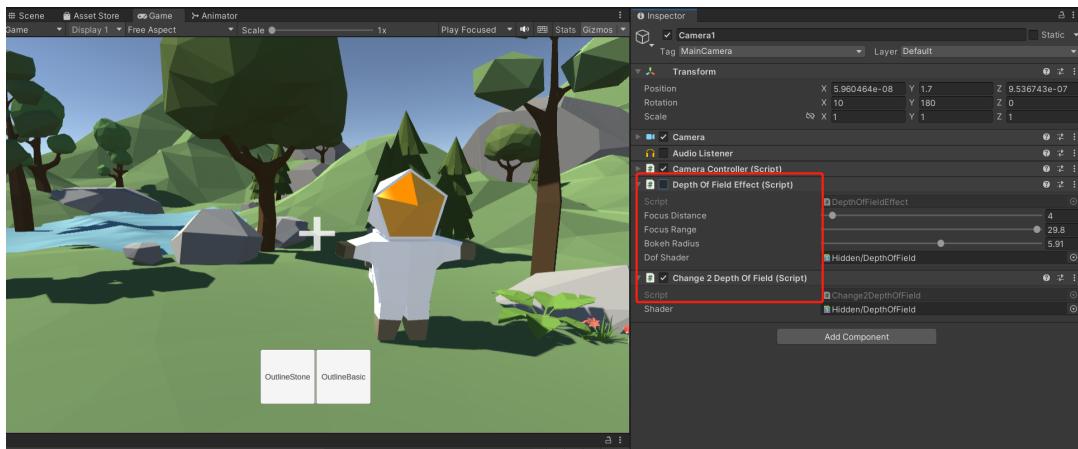
1. 将景深shader作为唯一的效果附加到相机上。由于我们将依赖于深度缓冲区，因此效果不会考虑透明几何体。



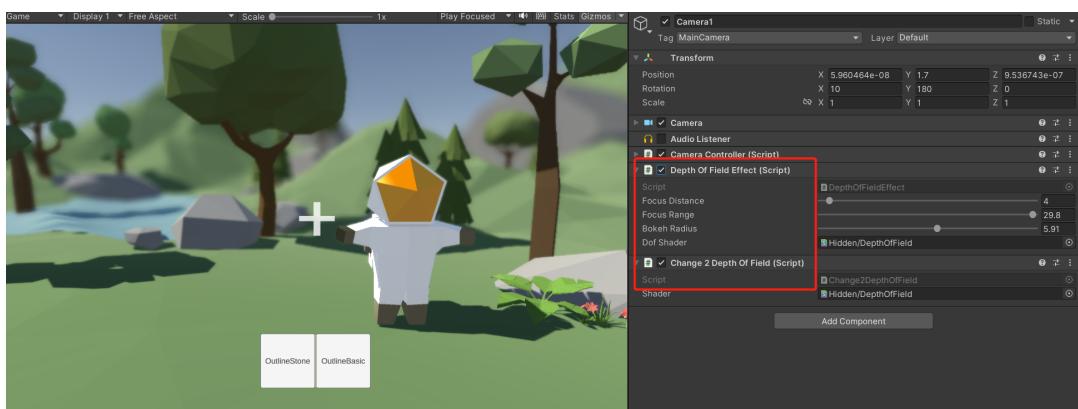
2. 模糊圆的半径是一个度量点投影失焦程度的指标。我们对模糊圆可视化。因为CoC依赖于与相机的距离，我们需要从深度缓冲读取数据。事实上，深度缓冲正是我们所需要的，因为相机的焦点区域是一个平行于相机的平面，假设镜头和图像平面是对齐的和完美的。从深度纹理取样，转换成线性深度，然后渲染。
3. 一幅图像是由光圈状在图像平面上的许多投影组成的。所以创造散景的一种方法便是使用每个texel的颜色去渲染模型，并基于其CoC去创造大小和不透明度。另一种方法不是将一个片段投射到许多个片段上，每个片段从所有可能影响它的texel中积累颜色。

- 最终效果：

未启用景深（由右侧Change2DepthOfField控制DepthOfFieldEffect是否启用）：

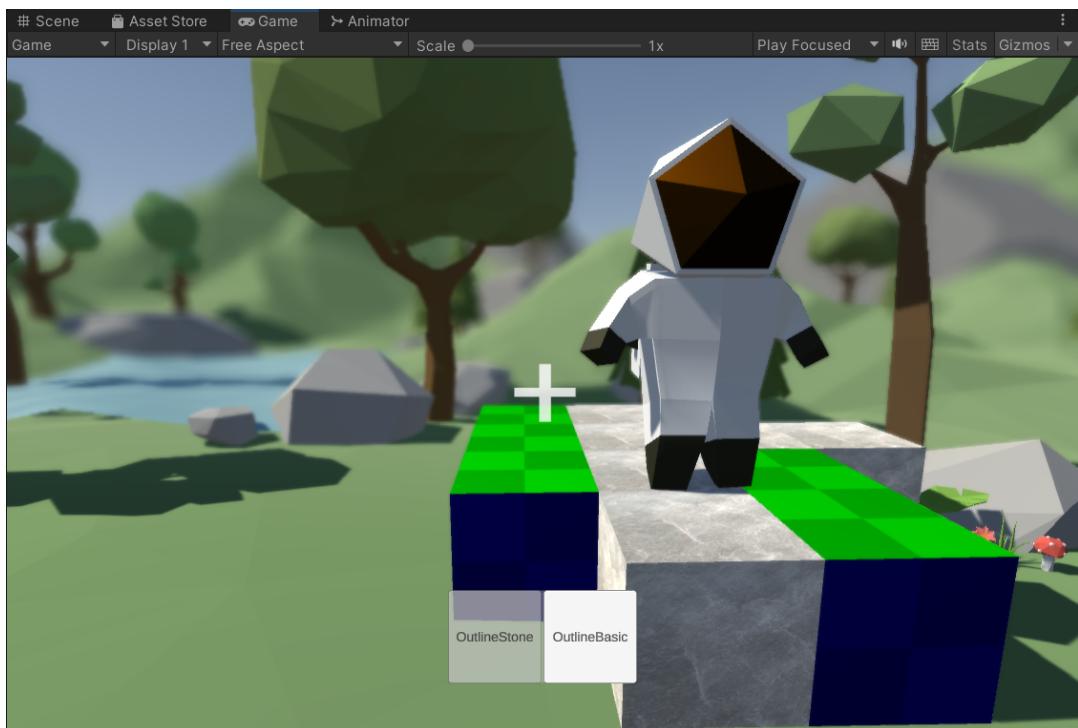


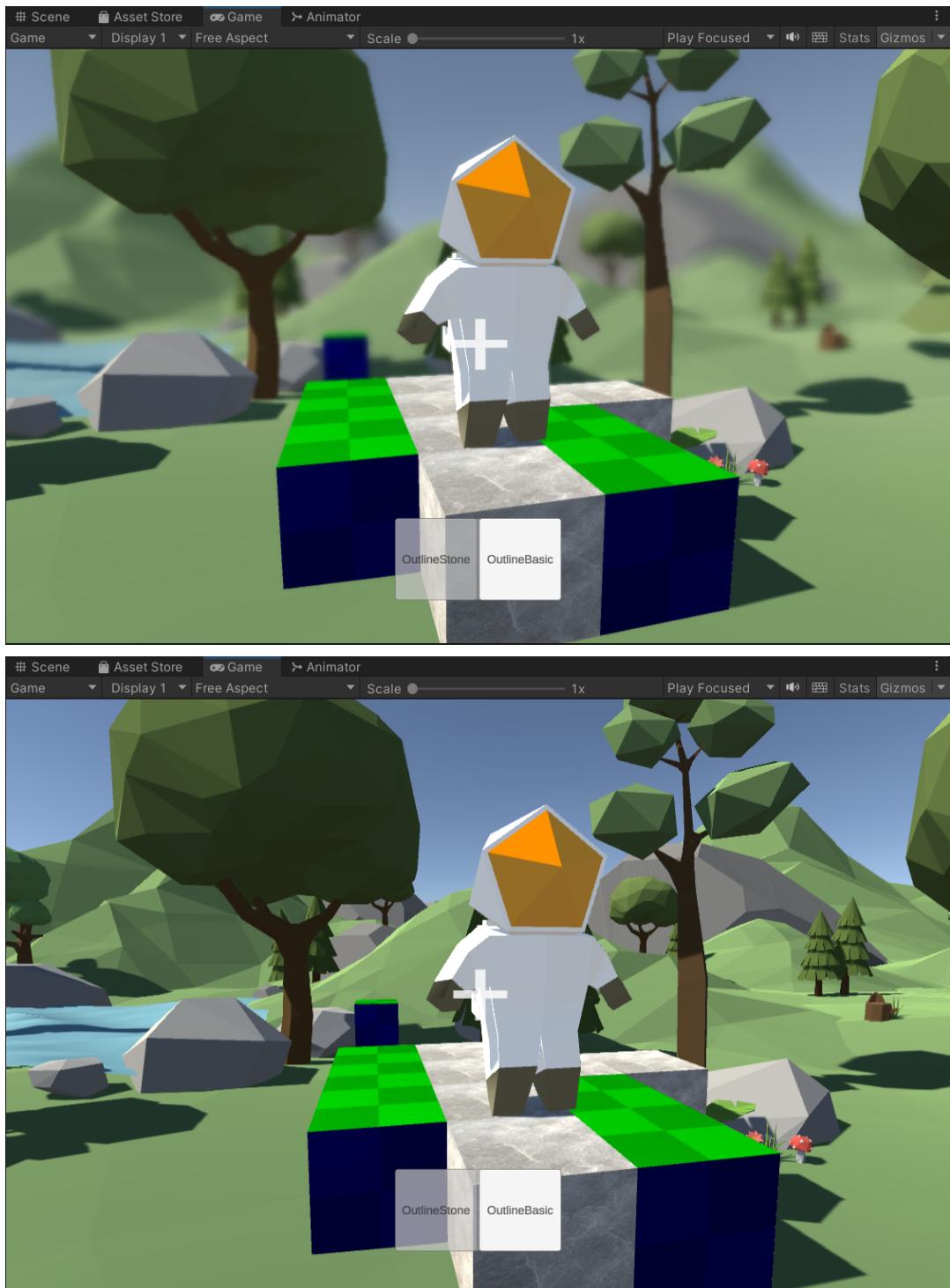
启用景深：



#### 4. 操作说明&最终效果

- 按下键盘C控制场景中宇航员的渲染方式在原shader和卡通风格之间切换  
(ChangeCartoonShader.cs)
- 按下键盘P控制场景是否启用景深效果 (Change2DepthOfField.cs)
- 其他同hw4





## 5. 参考文档

- 【2】[http://dalab.se.sjtu.edu.cn/gp2019/reference/NPAR07\\_IllustrativeRenderingInTeamFortress2.pdf](http://dalab.se.sjtu.edu.cn/gp2019/reference/NPAR07_IllustrativeRenderingInTeamFortress2.pdf)
- 【2】[\(54条消息\)Unity Toon Shader 卡通着色器 \(一\)：卡通着色 unity\\_toonshader\\_Sorumi的博客-CSDN博客](#)
- 【3】[景深 \(catlikecoding.com\)](#)