姓名：Celine Loh Yan Yi 骆彦伊

学号：520030990040

上机作业#5：Custom Shader

<u>基于物理的 Shader (Physically based shader)</u>

1. 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。
2. Unity 标准 Shader 则使用了基于 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。

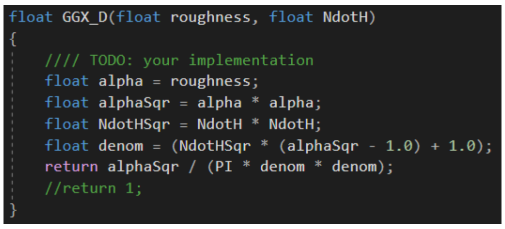
$$f(l,v) = \begin{cases} k_d f_d + k_s \dfrac{D(h)F(v,h)G(l,v,h)}{4(n\cdot l)(n\cdot v)}, & \theta < 90^\circ \\ 0, & else \end{cases}$$

a. Distribution term

对于 D 项，我们使用比较常用的 GGX 算法：

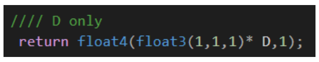
$$D_{GGX} = \frac{\alpha^2}{\pi((n\cdot h)^2(\alpha^2-1)+1)^2}$$

其中α是粗糙程度。



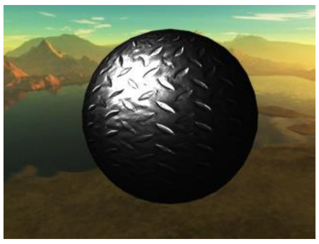
```
float GGX_D(float roughness, float NdotH)
{
    //// TODO: your implementation
    float alpha = roughness;
    float alphaSqr = alpha * alpha;
    float NdotHSqr = NdotH * NdotH;
    float denom = (NdotHSqr * (alphaSqr - 1.0) + 1.0);
    return alphaSqr / (PI * denom * denom);
    //return 1;
}
```

我们需要在 Fragment Shader 中使用



```
//// D only
 return float4(float3(1,1,1)* D,1);
```

即可得到以下效果（当 Smoothness = 0.5）



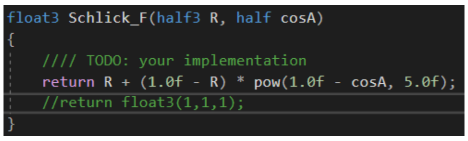
b. Fresnel term

对于F项，我们使用普通的 Schlick 模型进行估计，原本的公式中使用了反射系数 Reflection 如下

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

但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。

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

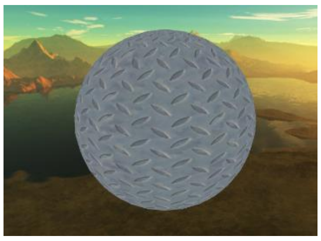


```
float3 Schlick_F(half3 R, half cosA)
{
    //// TODO: your implementation
    return R + (1.0f - R) * pow(1.0f - cosA, 5.0f);
    //return float3(1,1,1);
}
```

我们需要在 Fragment Shader 中使用

```
//// F only (looks like pure color)
  return float4(float3(1,1,1) * F,1);
```

即可得到以下效果（当 Metallicness = 1）



c.　Geometry term

对于G项，使用 Cook-Torrance 的模型。

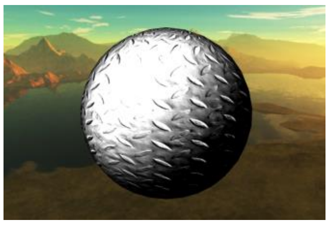
$$G_{Cook-Torrance} = \min\left(1, \frac{2(n \cdot h)(n \cdot v)}{v \cdot h}, \frac{2(n \cdot h)(n \cdot l)}{v \cdot h}\right)$$

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

直接对 G项进行渲染，可以看到模型几何阴影遮挡。

```
//// G only
 return float4(float3(1,1,1)* G,1);
```

效果：



3. 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));
```

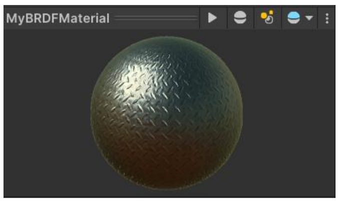
便可以得到直接光照的高光部分的反射光。这里为了得到更明显的高光效果，可以加上系数 π

```
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。

```
 return color;
```

通过逐渐调整参数，最终可以得到如下场景：



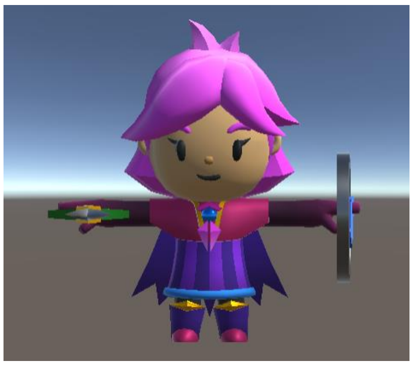
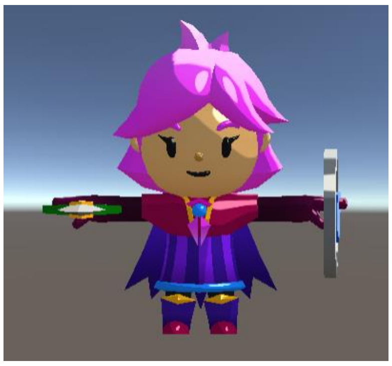
4. 材质球矩阵生成

首先需要创建一个球体，并创建一个该 Shader 对应的材质，设置好 Texture 和 Normal Map ，并绑定脚本 Material Change。将该球体保存成 Prefab ， 然后，在场景中创建一个空物体，为其绑定对应的 Material Matrix Controller ，并在脚本的 Prefab 一栏中挂上之前创建的 Prefab。运行场景后即可得到以下效果：

非真实感渲（Toon Shading）

左图：只有环境光照 (ambient) 和漫反射 (diffuse)时的效果

右图：添加 Toon Shader 后的效果

Steps:

1. 简化颜色
   实现原理是把 diffuse 漫反射颜色简化成对比较明显的几个色阶，首先尝试一下降到 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。
   ```
   float ndl = max(0, dot(normalDir, lightDir));
   ```

2. 设置属性：
   - RampThreshold 色阶阈值
   - RampSmooth 色阶间平滑度（可避免导致明显的分界线）

   使用 smoothstep 平滑函数，根据 RampSmooth 对色阶之间进行过渡。
   ```
   ramp = interval * smoothstep(level - _RampSmooth * interval * 0.5, level
       + _RampSmooth * interval * 0.5, diff) + level - interval;
   ```

3. 设置属性 HColor 高光颜色和 SColor 阴影颜色。
   ```
   _SColor = lerp(_HColor, _SColor, _SColor.a);
   float3 rampColor = lerp(_SColor.rgb, _HColor.rgb, ramp);
   ```
   ```
   fixed3 diffuse = s.Albedo * lightColor * rampColor;
   ```

4. 增加镜面高光和边缘光
   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

设置边缘光的属性：

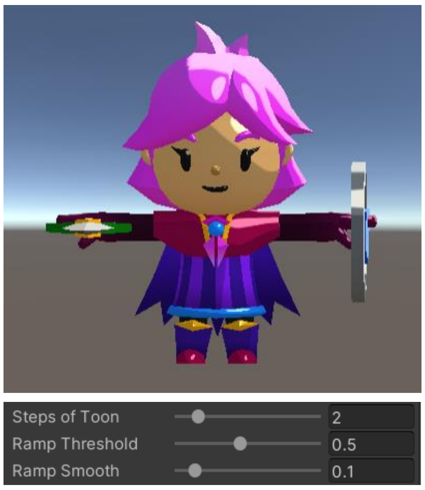
- RimColor 边缘光颜色
- RimThreshold 边缘光阈值
- RimSmooth 边缘光色阶的平滑度

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。

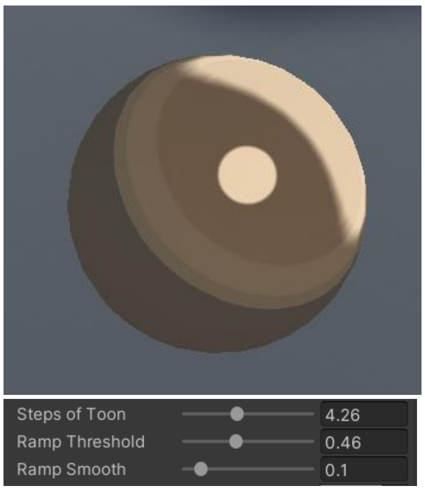
5. 阴影纹理
   单纯用颜色来做阴影会缺少层次感，可以使用纹理，对不同部分添加不同的阴影颜色。我们需要自定义 SurfaceOutput，添加 Shadow 保存纹理采样的颜色。

6. 多阶色阶
   同样使用 RampThreshold 控制光影的比例：
   ```
   float diff = smoothstep(_RampThreshold - ndl, _RampThreshold + ndl, ndl);
   ```
   增加属性 ToonSteps 表示色阶层数：
   ```
   float ramp = floor(diff * _ToonSteps) / _ToonSteps;
   ```
   色阶之间需要根据 RampSmooth 平滑过渡

7. 最后效果如下：



Steps of Toon	2
Ramp Threshold	0.5
Ramp Smooth	0.1



Steps of Toon	4.26
Ramp Threshold	0.46
Ramp Smooth	0.1

<u>Post-Processing (Bloom Effect)</u>

Bloom is an effect which messes up an image by making a pixels' color bleed into adjacent pixels. It's like blurring an image, but based on brightness. This way, we could communicate overbright colors via blurring.

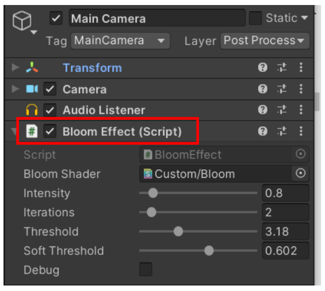
Main ideas:

1. Render to a temporary texture.
2. Blur via downsampling and upsampling.
3. Perform progressive sampling.
4. Apply a box filter.
5. Add bloom to an image.

Steps:

1. Create a new scene with default lighting and make sure that the camera is HDR enabled.

2. Create a new BloomEffect component (script). Add this component as the only effect to the camera object.



Main Camera — Static
Tag: MainCamera — Layer: Post Process

Transform
Camera
Audio Listener
Bloom Effect (Script)

Script	BloomEffect
Bloom Shader	Custom/Bloom
Intensity	0.8
Iterations	2
Threshold	3.18
Soft Threshold	0.602
Debug	☐

3. Add 'ImageEffectAllowedInSceneView' so it's easier to see the effect from a varying point of view.

```
[ExecuteInEditMode, ImageEffectAllowedInSceneView]
Unity Script (1 asset reference) | 0 references
public class BloomEffect : MonoBehaviour {
```

4. Blurring

   The bloom effect is created by taking the original image, blurring it somehow, then combining the result with the original image. Hence, we must first be able to blur an image before creating a blooming effect.

   a. Rendering to Another Texture

   - Applying an effect is done via rendering from one render texture to another. If we could perform all the work in a single pass, then we could simple blit from the source to the destination, using an appropriate shader. But blurring is a lot of work, so we can introduce an intermediate step to blit from the source to a temporary texture, then from that texture to the final destination.

```
RenderTexture.GetTemporary
```

   b. Downsampling

   - Blurring an image is done by averaging pixels. For each pixel, we have to decide on a bunch of nearby pixels to combine. Which pixels are included defines the filter kernel used for the effect. A little blurring can be done by averaging only a few pixels, which means a small kernel. A lot of blurring would require a large kernel, combining many pixels.
   - The simplest and quickest way to average pixels is to take advantage of the bilinear filtering built into the GPU. However, direct downsampling to a low resolution leads to poor result. A better approach is to downsample multiple times, halving the resolution each step until the desired level is reached.

```
int i = 1;
for (; i < iterations; i++) {
    width /= 2;
    height /= 2;
    if (height < 2) {
        break;
    }
    currentDestination = textures[i] = RenderTexture.GetTemporary(width, height, 0, format);
    Graphics.Blit(currentSource, currentDestination, bloom, BoxDownPass);
    currentSource = currentDestination;
}
```

   c. Upsampling

   - Instead of releasing and then claiming the same textures twice per render, we can keep track of them in an array to get a better blurring result compared to as before.

```
for (i -= 2; i >= 0; i--) {
    currentDestination = textures[i];
    textures[i] = null;
    Graphics.Blit(currentSource, currentDestination, bloom, BoxUpPass);
    RenderTexture.ReleaseTemporary(currentSource);
    currentSource = currentDestination;
}
```

   d. Custom Shading

   - To further improve our blurring, we have to switch to a more advanced filter kernel than simple bilinear filtering. Hence, we can create a custom shader.

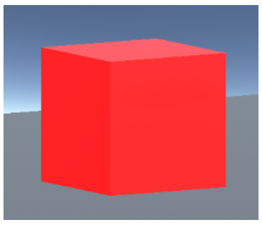
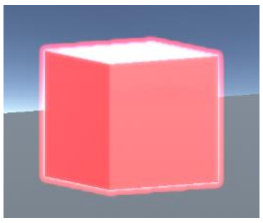
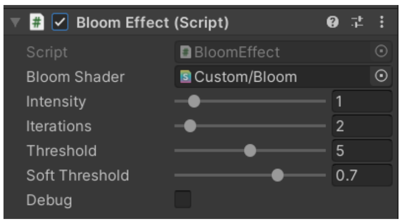
```
Shader "Custom/Bloom" {
    Properties {
        _MainTex ("Texture", 2D) = "white" {}
    }
```

   e. Box Sampling

   - Instead of relying on a bilinear filter only, we'll use a simple box filter kernel instead. It takes four samples instead of one, diagonally positioned so we get the averages of four adjacent 2×2 pixels blocks. Sum these samples and divide by four, so we end up with the average of a 4×4 pixel block, doubling our kernel size.

5. Creating Bloom
   Blurring the original image is the first step of creating a bloom effect. The second step is to combine the blurred image with the original, brightening it. However, we won't just use the final blurred result, as that would produce a rather uniform smudging. Instead, lower amounts of blurring should contribute more to the result that higher amounts of blurring. We can do this by accumulating the intermediate results, adding to the old data as we upsample.

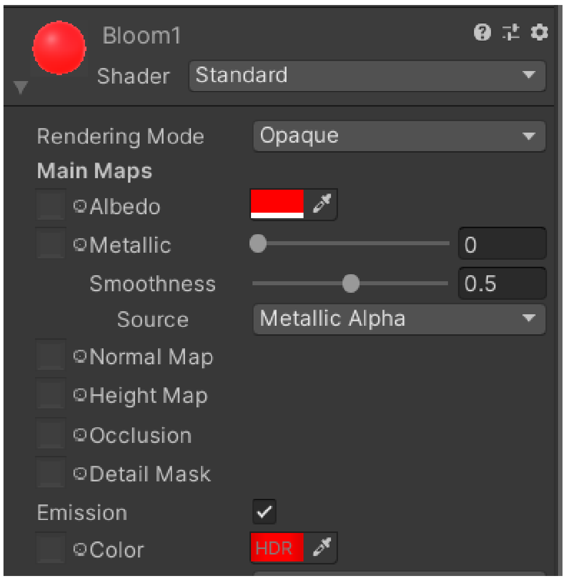
6. Result

Bloom Effect (Script)

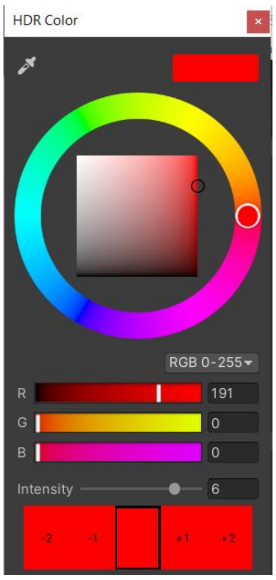
Script	BloomEffect
Bloom Shader	Custom/Bloom
Intensity	1
Iterations	2
Threshold	5
Soft Threshold	0.7
Debug	

7. Notes: To make bloom appear, we can increase the light contribution of some of the materials.



Bloom1

Shader	Standard
Rendering Mode	Opaque
Main Maps	
Albedo	🟥
Metallic	0
Smoothness	0.5
Source	Metallic Alpha
Normal Map	
Height Map	
Occlusion	
Detail Mask	
Emission	✓
Color	HDR



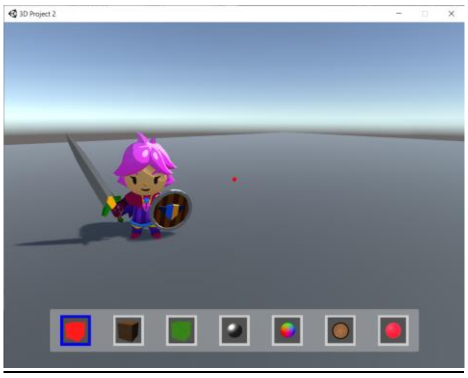
HDR Color

RGB 0-255

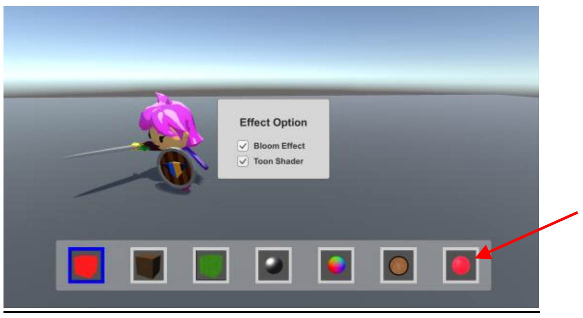
R	191
G	0
B	0
Intensity	6

Game Operation Guide

1. Game View:

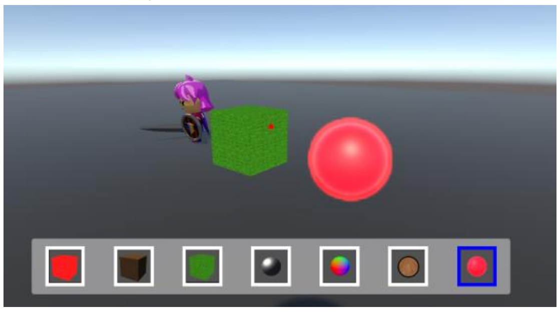


2. Keyboard:
   Use WASD key to move the player (FPS)
   Scroll up or down to change the block type
   Press ‘P’ key to enable/disable the ‘Effect Option’ menu



*Toggle ‘Bloom Effect’ or ‘Toon Shader’ to enable/disable the effects respectively.
*Only the last sphere has the bloom effect

3. Mouse:
   Move the mouse to rotate the camera view
   Left click to place a new block



References

1. Toon Shading

- https://zhuanlan.zhihu.com/p/514575315
- https://blog.csdn.net/qq_24153371/article/details/81837905?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522168104277016800225579103%2522%252C%2522scm%2522%253A%252220140713.130102334.pc%255Fall.%2522%257D&request_id=168104277016800225579103&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~first_rank_ecpm_v1~rank_v31_ecpm-1-81837905-null-null.142^v82^insert_down38,201^v4^add_ask,239^v2^insert_chatgpt&utm_term=toon%20shader%20unity%20%E4%BB%A3%E7%A0%81%E5%AE%9E%E7%8E%B0&spm=1018.2226.3001.4187

2. Bloom Effect

- https://catlikecoding.com/unity/tutorials/advanced-rendering/bloom/