游戏设计与开发 自定义特效 Shader 参考文档

SJTU DALab 2020.04

本文档以介绍作业中 BRDF 相关项为主。在本次作业中，你需要通过实现 Cook-Torrance 模型中的几个函数来实现一个基于物理的Shader（Physically based shader）。

1. BRDF Introduction

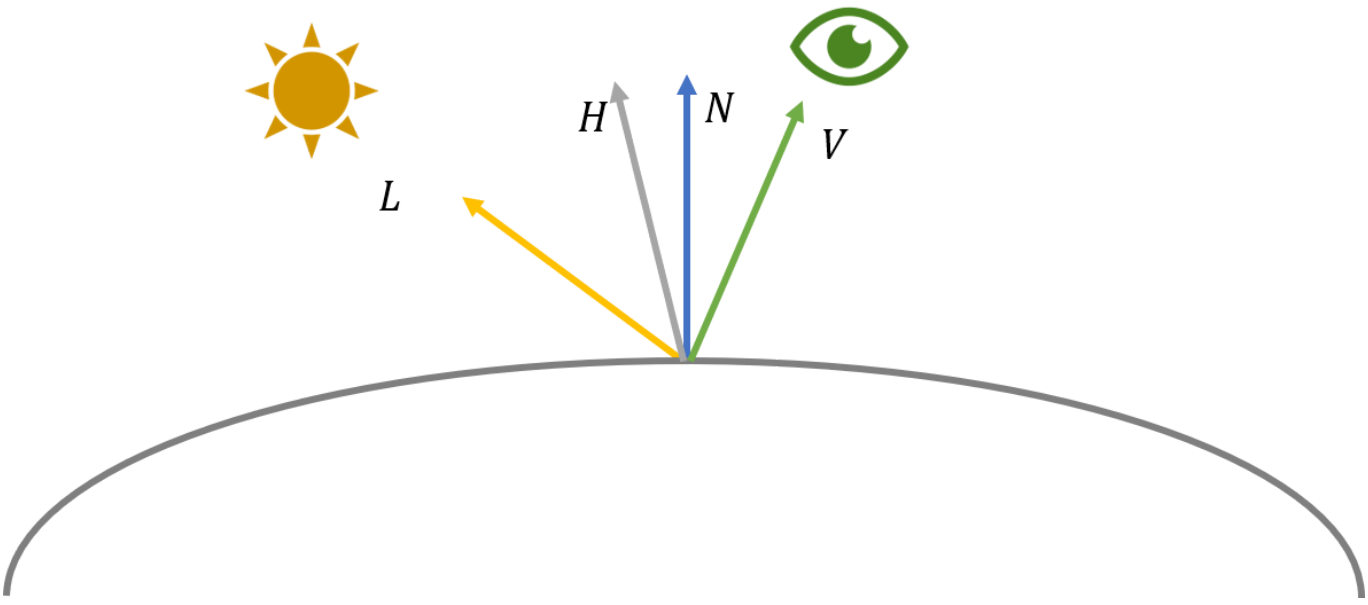
双向反射分布函数（ BRDF ）描述了一个表面在每个入射光、出射光方向设定下反射了多少光。

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 I 入射的光线有多少被反射到 v 方向。而公式中对入射光 I 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）。

$$f_r = k_d f_d + k_s f_s$$



在上一次作业中，我们其实已经实现了简单的 BRDF ，其中我们对于漫反射使用了 Lambert 的模型，而对高光使用了 Blinn-Phong 模型。在当时的计算中有

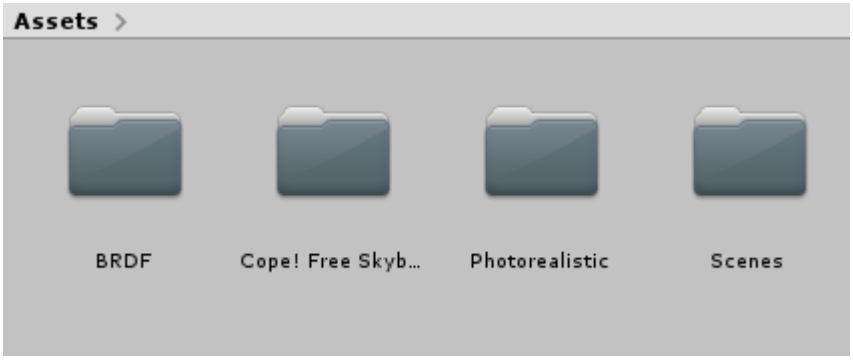
$$f_r(l, v) = \begin{cases} k_d f_{Lambert} + k_s f_{Blinn-Phong}, & \theta < 90^\circ \\ 0, & else \end{cases}$$

而 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。

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

##2. Package

本次作业提供 Shader 的框架代码和相关资源，请在 Canvas 上下载，并导入 Unity 。



本次作业中已经提供了天空盒资源（`Asset/Cope! Free Skybox`）、材质贴图、法线贴图资源（`Asset/Photorealistic`）中，你可以选择其中合适的资源进行本次作业。

本次作业中，已经写好了一个初步的 Shader 框架 `Asset/BRDF/MyBRDFShader.shader`，你需要在这个基础上实现自己的功能。我们仍然使用 Vertex Shader 和 Fragment Shader 来完成这次作业，其中 Vertex Shader 已经完成， Fragment Shader 中可能需要的变量也已经定义。

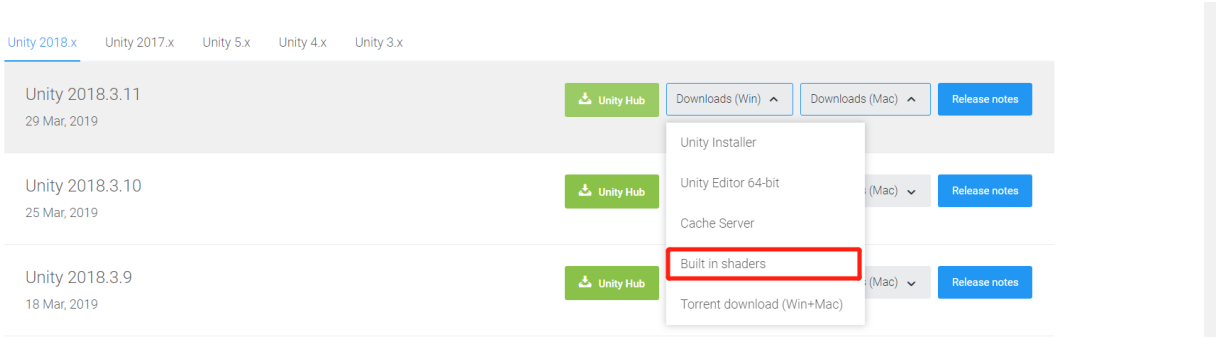
```
float _Glossiness;
float _Metallic;

fixed4 frag (FragmentData i) : SV_Target{
    ...
    float4 mainTex = ...; //// texture color
    float3 L = ...; //// light direction
    float3 V = ...; //// view direction
    float3 H = ...; //// half vector direction
    float3 N = ...; //// normal direction
    float NdotL = saturate( dot( N,L ));
    float NdotH = saturate( dot( N,H ));
    float NdotV = saturate( dot( N,V ));
    float VdotH = saturate( dot( V,H ));
    float LdotH = saturate( dot( L,H ));
    ...
    UnityIndirect gi = ...; //// using UnityGlobalIllumination
```

```
    float roughness = SmoothnessToRoughness(_Glossiness); //// roughness
    float3 specColor = ...;
    float3 albedo = ...;
}
```

为了得到更好的视觉效果， 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。

Unity 内置也实现了一些真实渲染的 Shader ，这次作业中会用到其中部分接口，如果你希望了解更多 Unity 的实现，可以通过 Unity download archive 下载内置的 Shader 代码进行阅读。



其中和本次作业相关的文件在 `builtin_shaders/CGIncludes/UnityPBSLighting.cginc` 和 `builtin_shaders/CGIncludes/UnityStandardBRDF.cginc` 中。这里对其结构作简单介绍，有兴趣的同学可以深入研究。

在 `UnityPBSLighting.cginc` 中定义了默认使用的 BRDF 函数，根据不同选项会使用不同的 BRDF 模型。

```
// Default BRDF to use:
#if !defined (UNITY_BRDF_PBS)
    #if SHADER_TARGET < 30 || defined(SHADER_TARGET_SURFACE_ANALYSIS)
        #define UNITY_BRDF_PBS BRDF3_Unity_PBS
    #elif defined(UNITY_PBS_USE_BRDF3)
        #define UNITY_BRDF_PBS BRDF3_Unity_PBS
    #elif defined(UNITY_PBS_USE_BRDF2)
        #define UNITY_BRDF_PBS BRDF2_Unity_PBS
    #elif defined(UNITY_PBS_USE_BRDF1)
        #define UNITY_BRDF_PBS BRDF1_Unity_PBS
    #else
        #error something broke in auto-choosing BRDF
    #endif
#endif
```

并在下面使用

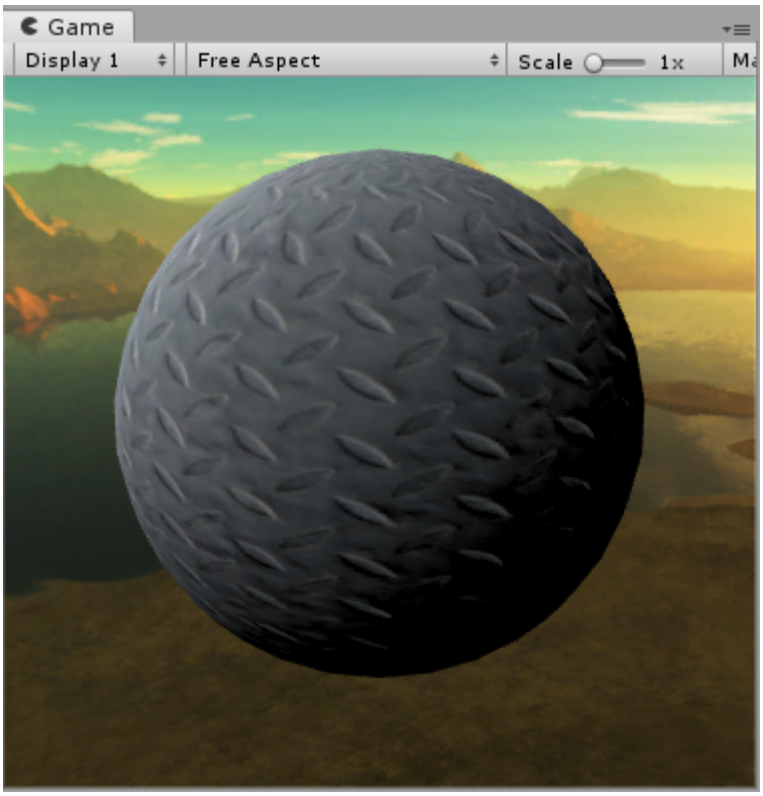
```
...
inline half4 LightingStandard (SurfaceOutputStandard s, float3 viewDir, UnityGI
gi)
```

```
{
    ...
    s.Albedo = DiffuseAndSpecularFromMetallic (s.Albedo, s.Metallic,
        /*out*/ specColor, /*out*/ oneMinusReflectivity);
    ...
    half4 c = UNITY_BRDF_PBS (s.Albedo, s.Specular, oneMinusReflectivity,
        s.Smoothness, s.Normal, viewDir, gi.light, gi.indirect);
    ...
}
...
```

而这个函数的实际定义在 `UnityStandardBRDF.cginc` 中，如 `half4 BRDF1_Unity_PBS(...)`。

##3. Diffuse

在本次实验中，不要求实现漫反射项，因此我们直接使用了 Unity 内置的漫反射实现公式 `DisneyDiffuse` 实现漫反射。其源码在 `UnityStandardBRDF.cginc` 中，有兴趣的同学可以通过链接自行了解这一方法。单独使用漫反射时效果如下：



##4. Specular

本次实验中需要实现的部分在直接高光反射中，你需要按照 Cook-Torrance 模型分别实现其中的 D、F 和 G 项。

4.1 Distribution term

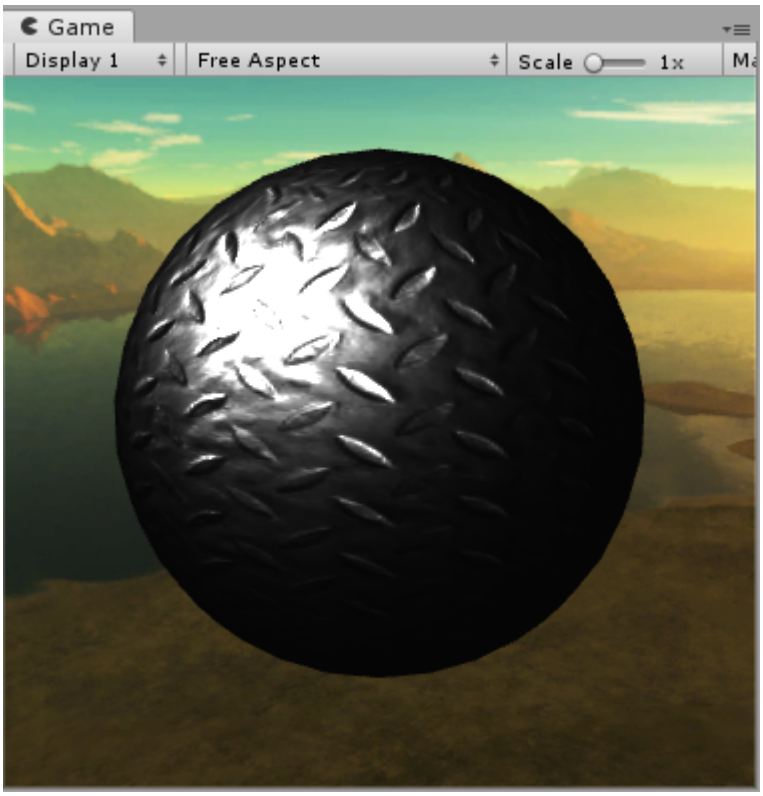
对于 D 项，我们使用比较常用的 GGX 算法，它是由 Walter 等人提出的，其形式如下：

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

其中 α 是粗糙程度，即代码中的 `roughness` 。你需要修改代码中的`GGX_D` 函数来实现。当实现这步骤后，你可以通过直接在 Fragment Shader 中使用

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

来看到只有 D 项效果。当参数 `Smoothness = 0.5` 时，可以看到高光效果



4.2 Fresnel term

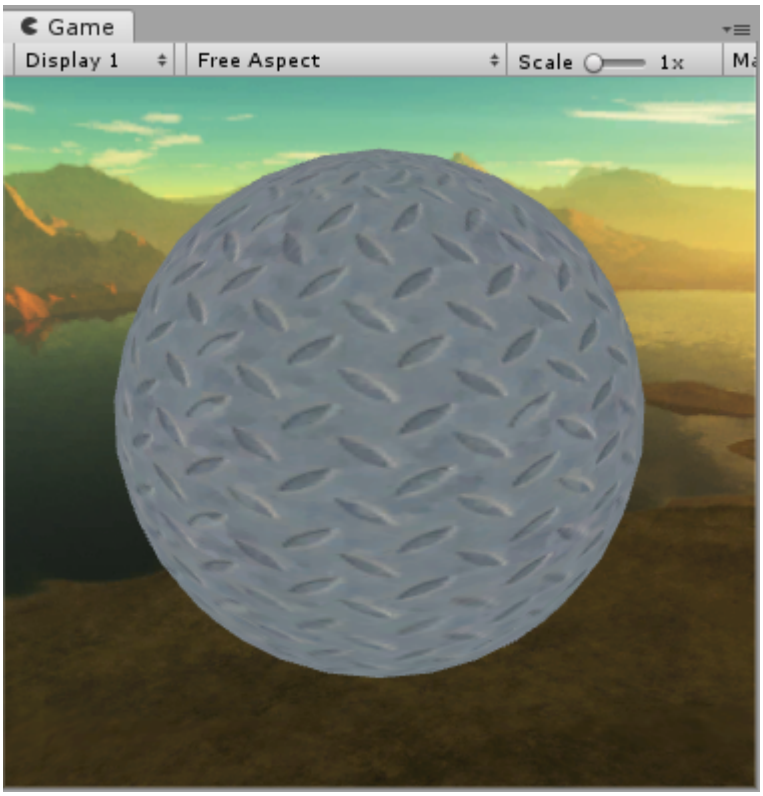
对于 F 项，我们使用普通的 Schlick 模型进行估计，原本的公式中使用了反射系数 `Reflection` 如下

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

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。

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

同样，你需要修改 `Schlick_F` 来实现这一部分。实现完成后，只有 F 项时，当 `Metallicness = 1` ，可以看到以下现象。

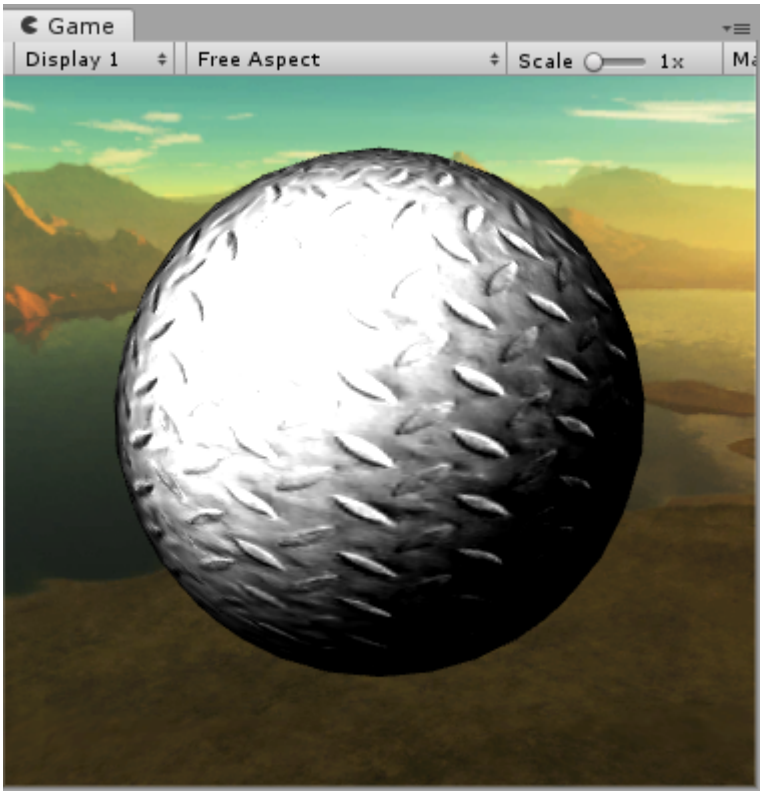


4.3 Geometry term

对于 G 项，使用 Cook-Torrance 的模型。

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

直接对 G 项进行渲染，可以看到模型几何阴影遮挡。



5. Result

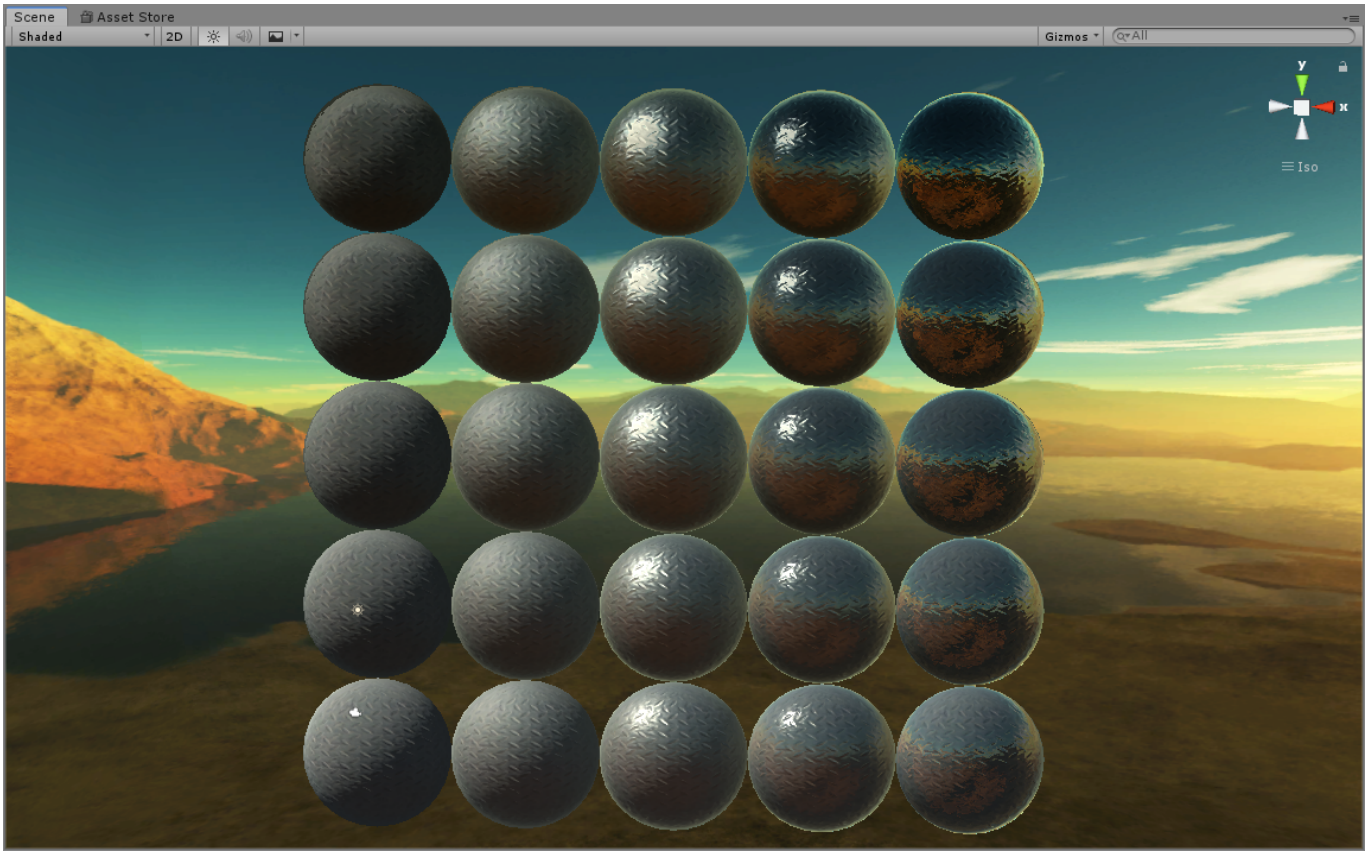
实现了上述的功能，我们便完成了一个 Cook-Torrance BRDF 模型。之后使用

```
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。通过逐渐调整参数，最终可以得到如下场景。

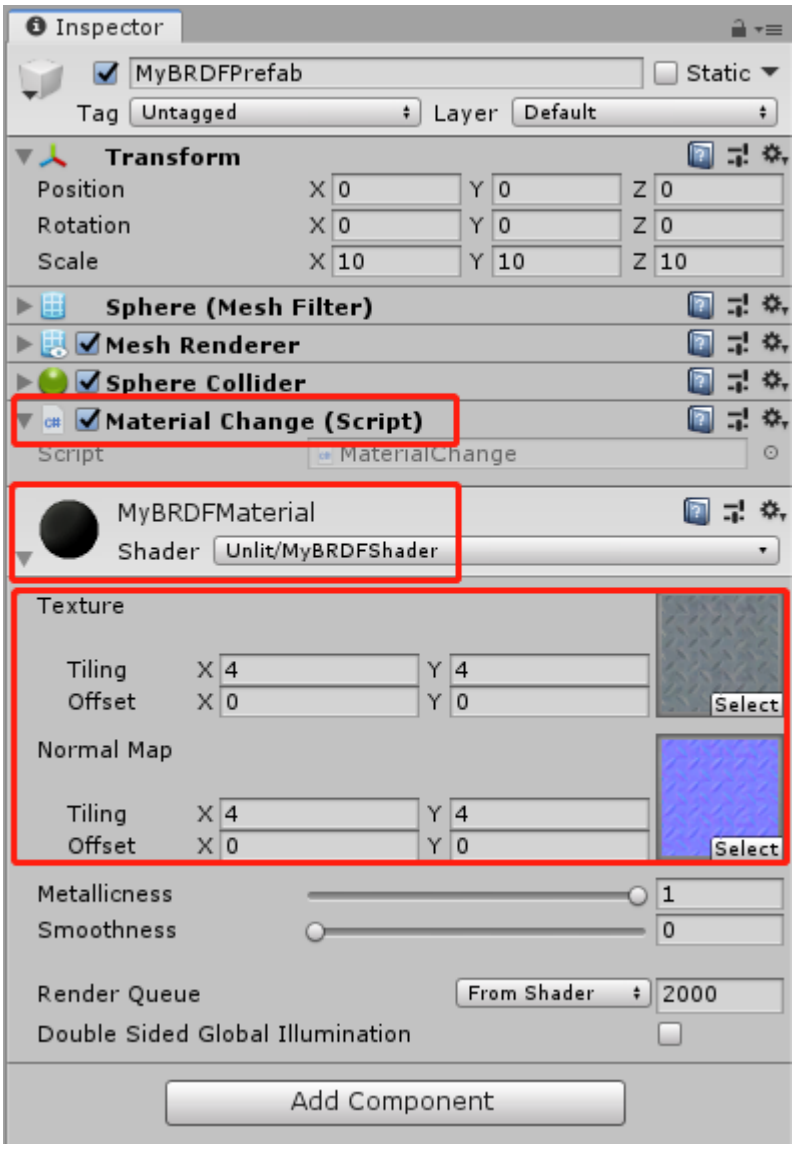


6. Appendix

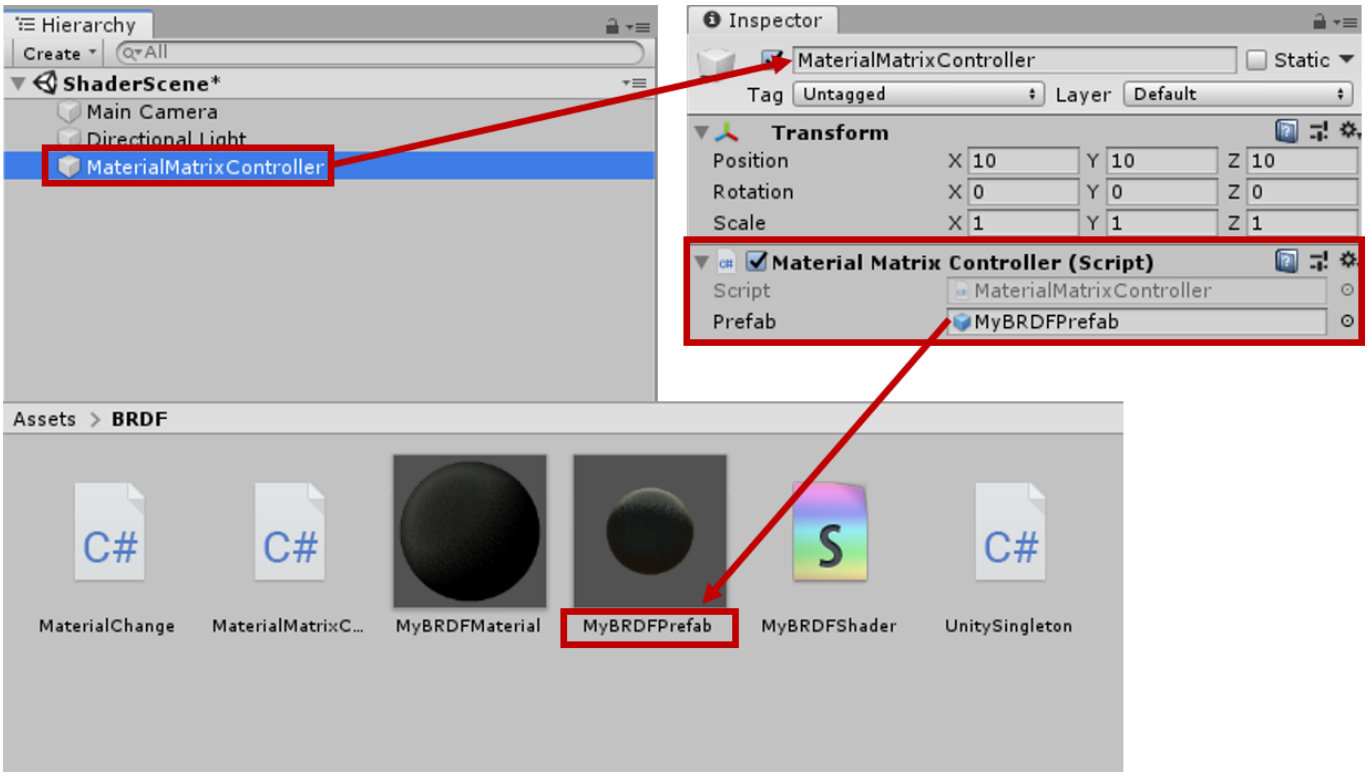
6.1 材质球矩阵生成

为了方便效果查看，作业中提供了代码可以直接生成如上图的材质球矩阵。

首先你需要创建一个球体，并创建一个你实现的 Shader 对应材质，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。



然后运行场景，就会看到如上图的材质球矩阵了。

Reference

- Websites
  - Unity build-in shaders
  - Brief BRDF Summary Kadircenk Blog
  - Siggraph 2015 shading course
- Papers
  - Cook Tarrance BRDF
  - GGX Distribuition term
  - Schlick Fresnel term
  - Cook Torrance Geometry term
  - PBS at Disney
- Books
  - Physically Based Rendering: From Theory To Implementation