hw4_SimpleShader作业报告

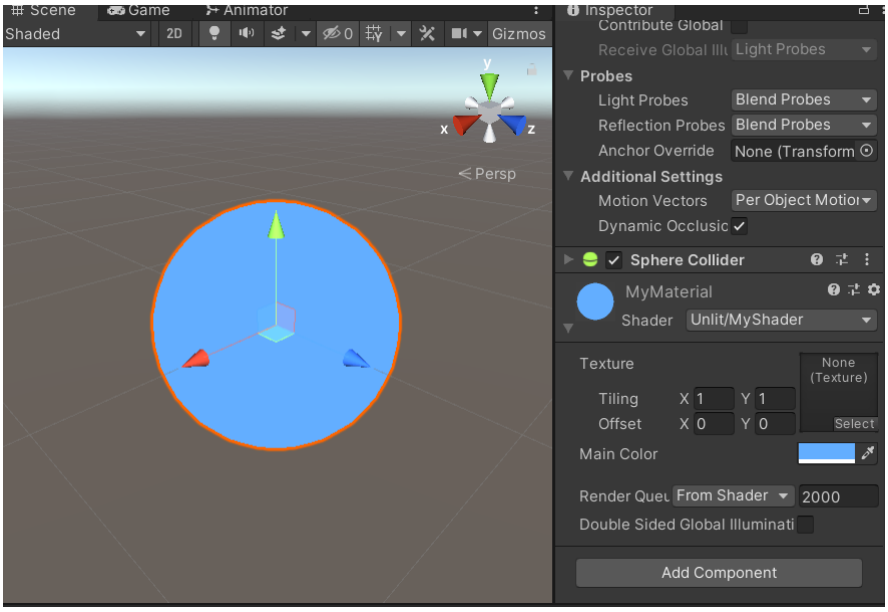
肖蔚尔 520030910314

1. Simple Shader实现过程&效果展示

1. 纯色Shader

使用vertex shader 和 fragment shader, 传递参数position ，在 Vertex Shader 中，程序应该通过输入的局部坐标系下的顶点位置，利用 MVP 矩阵（模型观察投影矩阵）计算得到屏幕空间中的位置。这个矩阵运算可以直接用 UnityObjectToClipPos 来代替。

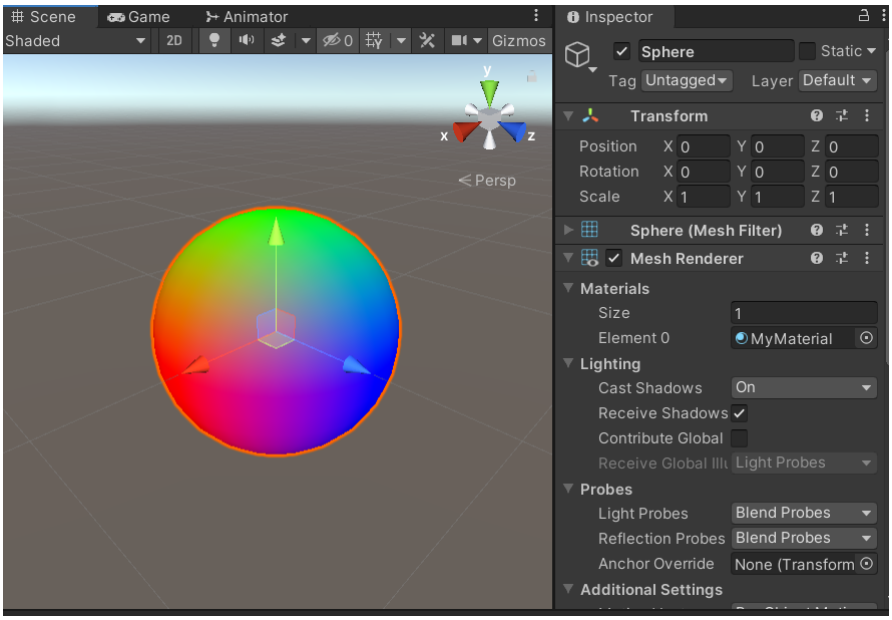
在fragment shader中，程序应该直接以颜色 _MainColor为结果返回该颜色。



2. 法线Shader

使用vertex shader 和 fragment shader, 传递参数normal（法线用），在vertex shader 中把物体空间的法线转化为世界坐标系下。

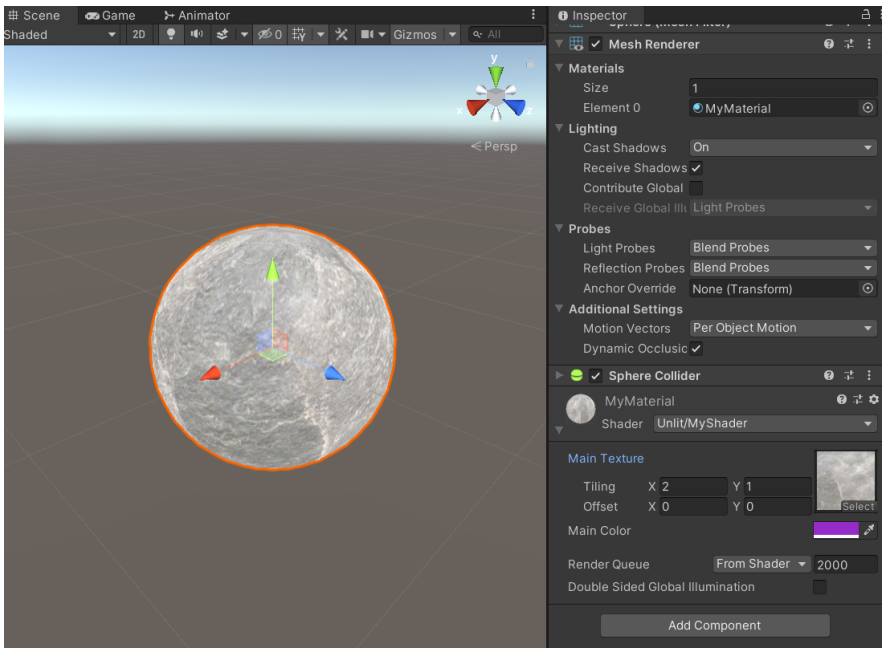
在fragment shader中，程序应该直接以颜色 normal 为结果返回该颜色。



3. 纹理Shader

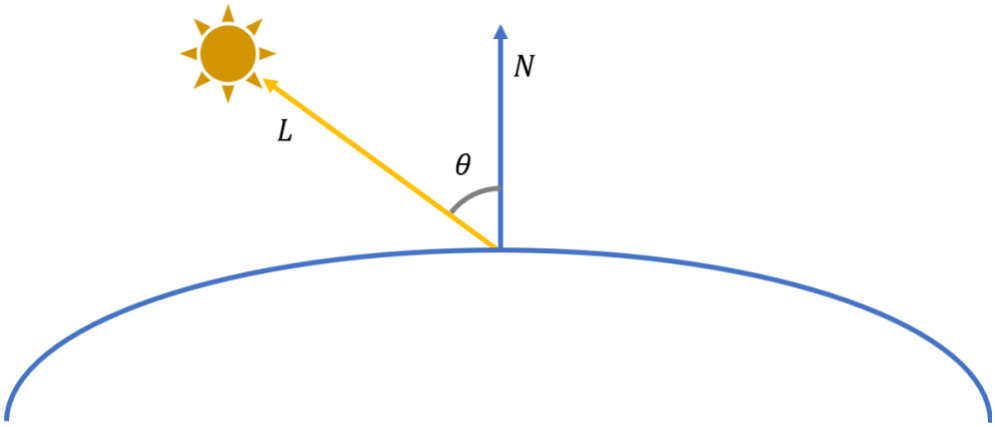
使用vertex shader 和 fragment shader, 传递参数uv（纹理用），在vertex shader 中将 VertexData.uv 赋值给 FragmentData.uv，并对 uv 坐标进行缩放偏移。

在fragment shader中，程序返回 _MainTex 中位于 FragmentData.uv 位置的颜色值。



2. 光照模型

1. 环境光&漫反射

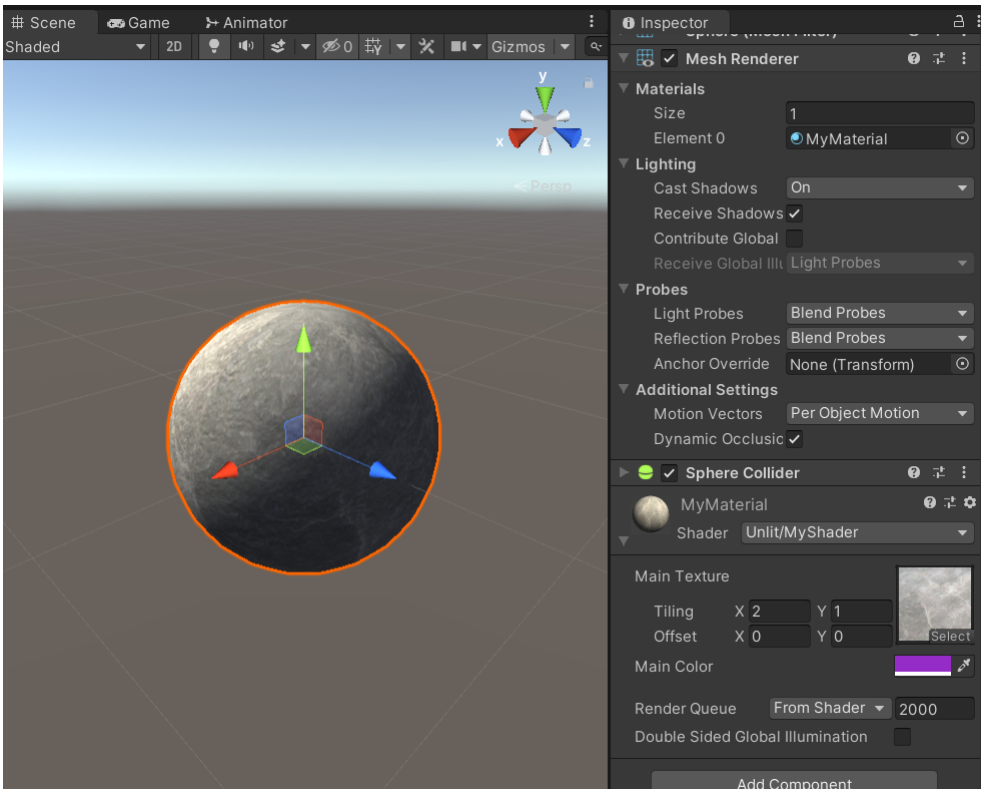


Lambert 光照模型中认为漫反射光的光强仅与入射光的方向和反射点处表面法向夹角的余弦成正比。因此，我们可以将公式写为如下形式：

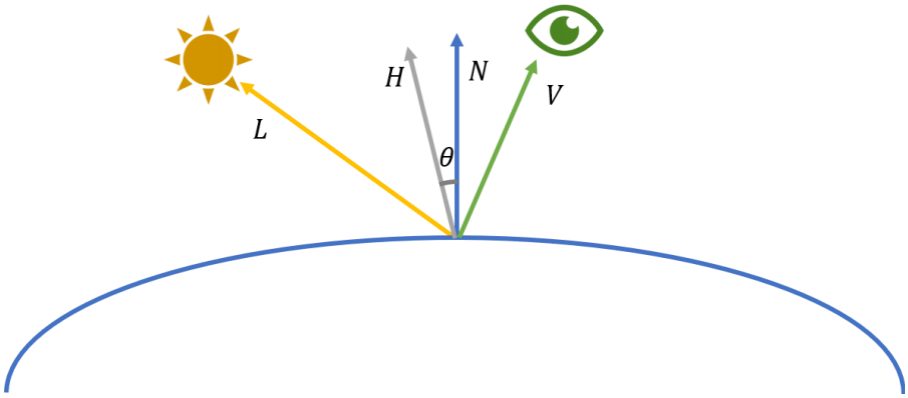
$$I_{diffuse} = I_{in} * cos\theta = I_{in}(L \cdot N)$$

在 Fragment Shader 中，首先需要计算入射光的方向以及颜色，然后依照公式计算漫反射光） DotClamped 函数可以将负数的点乘结果截断至0，以防负数产生

的错误光照效果），在漫反射光中考虑物体本身的颜色（diffuse）并加入环境光（ambient）使未被光照的地方不至于过暗 。



2. BlinnPhong模型（镜面反射）



Blinn Phong 模型引入了一个新的向量 *H* 的概念，它是光入射方向和视线方向之间角平分线的方向，也叫半矢量（Half Vector）。可以通过以下公式求解

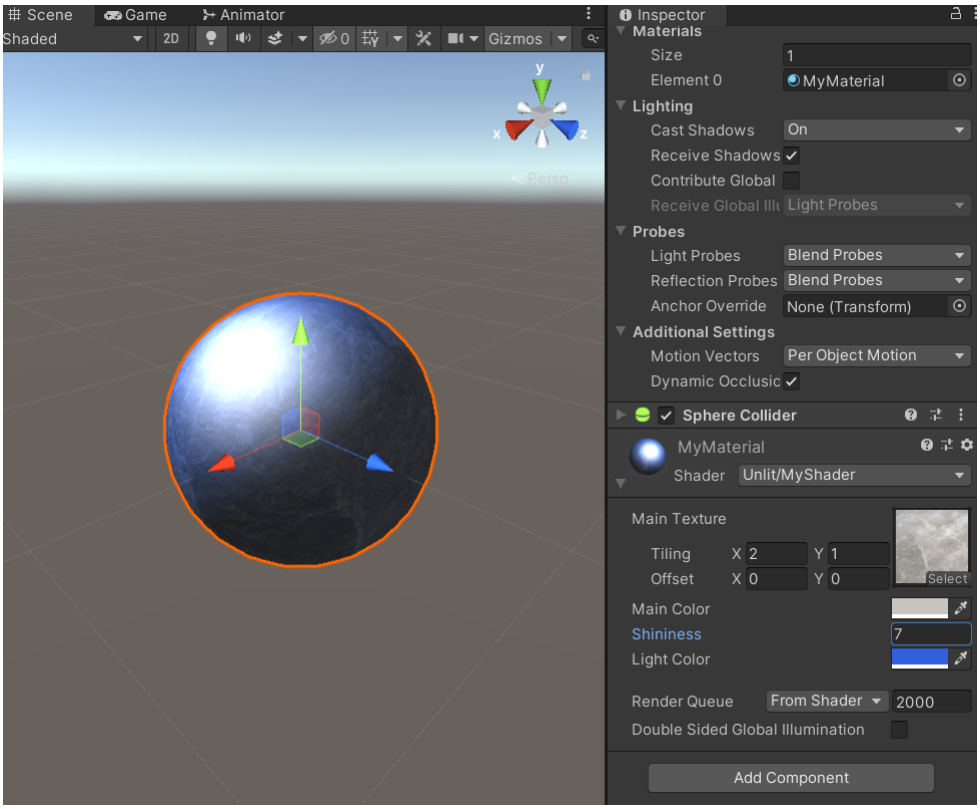
$$H = \frac{L+V}{||L_V||}$$

通过法线和半矢量之间的点积可以衡量镜面高光的亮度。直观的理解，当点积越大，表示两个向量越接近，即法线方向越接近光入射方向和视线方向的角平分线，换句话说，视线方向也就越接近镜面反射的出射光方向，此时镜面反射自然会增大。具体的，有如下公式

$$I_{specular} = (N \cdot H)^{n_{shininess}}$$

在 FragmentData 中，我们会额外需要一个顶点在世界坐标系中的位置 worldPos ，用以计算视线方向。在 Vertex Shader 中，通过将局部坐标系乘上坐标系转换矩阵，计算 FragmentData.worldPos 。

为了调节高光效果，在 Properties 中添加浮点数 _Shininess 属性，对应公式中的乘方系数。**通过GUI自定义光照颜色为蓝色，调节 _Shininess 为 7** ，可以得到较好的镜面反射光效果如下图。



3. 自定义Shader

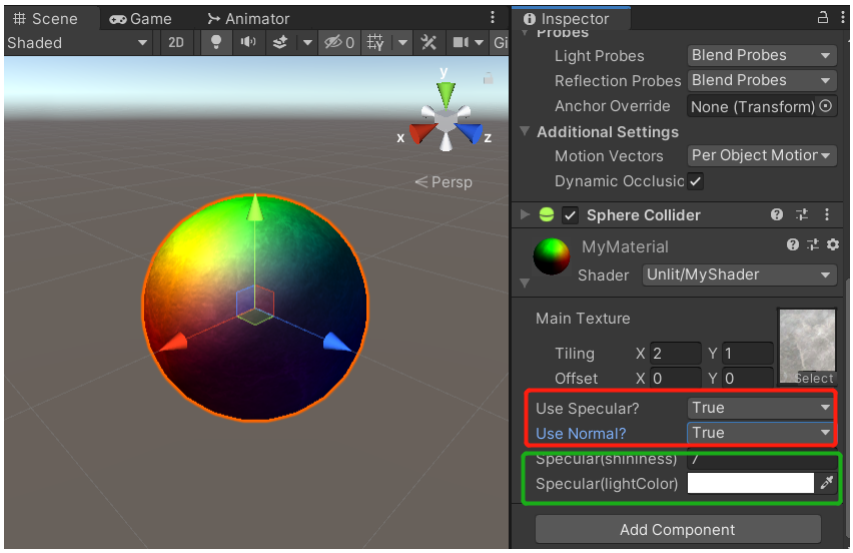
实现一个 Shader，其中能通过 GUI 面板下拉栏选择不同的渲染效果，包括**法线可视化**和**BlinnPhong镜面反射**，在BlinnPhong镜面反射选项中提供光照强度和光线颜色两个参数设置：

```
//GUI下拉框
        SpecularChoice specularChoice = SpecularChoice.False;
        NormalChoice normalChoice = NormalChoice.False;

        if (target.IsKeywordEnabled("USE_SPECULAR"))
            specularChoice = SpecularChoice.True;
        if (target.IsKeywordEnabled("USE_NORMAL"))
            normalChoice = NormalChoice.True;
        EditorGUI.BeginChangeCheck();
        specularChoice = (SpecularChoice)EditorGUILayout.EnumPopup(new
GUIContent("Use Specular?"), specularChoice);
        normalChoice = (NormalChoice)EditorGUILayout.EnumPopup(new
GUIContent("Use Normal?"), normalChoice);
        if (EditorGUI.EndChangeCheck())
        {
            if (specularChoice == SpecularChoice.True)
                target.EnableKeyword("USE_SPECULAR");
            else
                target.DisableKeyword("USE_SPECULAR");
            if (normalChoice == NormalChoice.True)
                target.EnableKeyword("USE_NORMAL");
            else
                target.DisableKeyword("USE_NORMAL");
        }
        //GUI镜面反射参数隐藏（_Shininess和_LightColor）
        if (specularChoice == SpecularChoice.True)
        {
            MaterialProperty shininess = FindProperty("_Shininess",
properties);
```

```
            GUIContent shininessLabel = new
GUIContent(shininess.displayName);
            editor.FloatProperty(shininess, "Specular(shininess)");
            MaterialProperty lightColor = FindProperty("_LightColor",
properties);
            GUIContent lightColorLabel = new
GUIContent(lightColor.displayName);
            editor.ColorProperty(lightColor, "Specular(lightColor)");
        }
```

效果如下：



4. 高级Shader

1. 描边Shader（Outline）

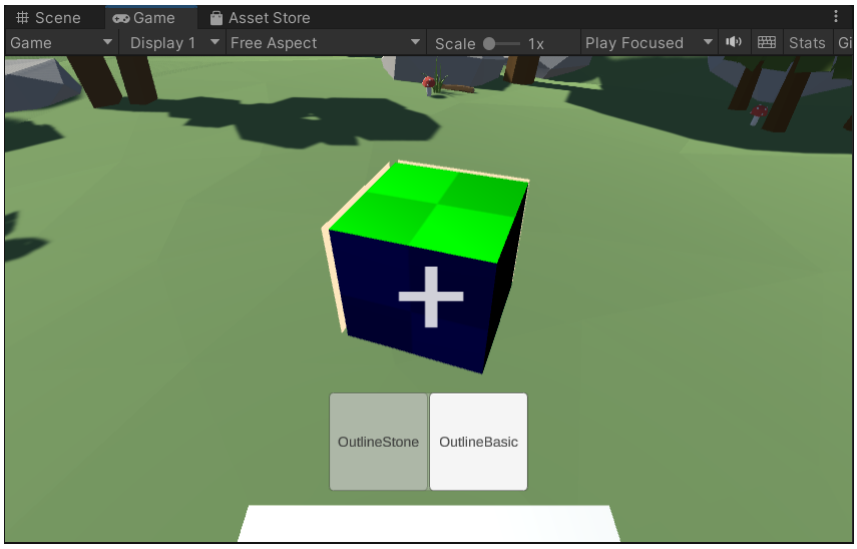
OutlineShader是一个surface shader，有不同灯光和渲染方式。表面着色器函数为surf，使用自定义光照模型。

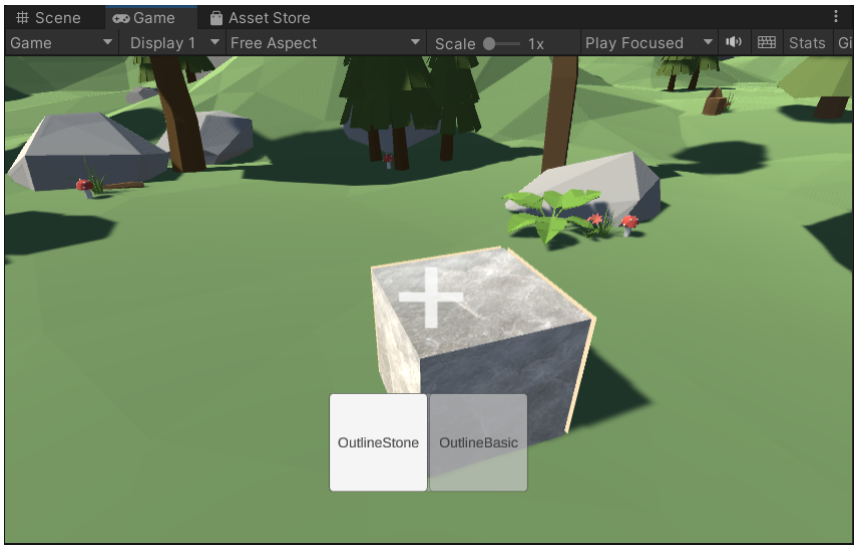
第一个pass先用surface shader渲染无描边的物体由于surface shader自动生成vertex和fragment shader，因此只需要在SurfaceOutputStandard中控制灯光和渲染参数。

第二个pass绘制描边。在vertex shader中改变绘制点的位置，沿着法线方向扩展长度。需要的参数是OutlineColor、OutlineThickness分别表示外框颜色和厚度。fragment shader直接返回外框颜色。

对于3中的自定义shader，只需添加第二个描边pass。

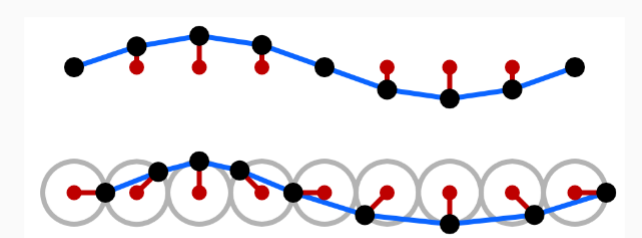
效果如下：





2. 海浪Shader（Gerstner Waves）

每个表面点绕着一个固定的锚点绕圈运动。当波峰接近时，该点向它移动。波峰通过后，它会滑回来，然后下一个波峰出现。结果是水在波峰中聚集，在波谷中扩散，同样的情况也会发生在我们的顶点上。



Sine vs. Gerstner wave.

有时候因为振幅相对于波长太大，所以表面点的在表面上方形成环路。如果这是真实的水，那么波就会破碎，因此使用另外一种模型。这个模型涉及表面压力和波的形状的关系，这样可以调节steepness而不是振幅。

$$\text{Then we have } P = \begin{bmatrix} x + \frac{s}{k}\cos f \\ \frac{s}{k}\sin f \end{bmatrix}, \text{ which simplifies our tangent to } T = \begin{bmatrix} 1 - s\sin f \\ s\cos f \end{bmatrix}$$

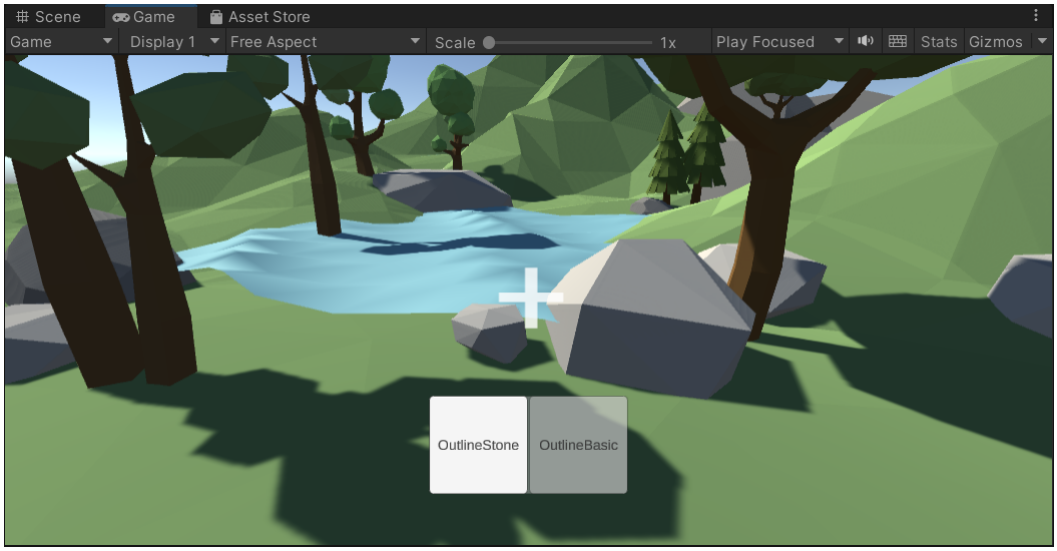
修改波动方向只需要修改xy以及对应法向量：

$$P = \begin{bmatrix} x + D_x\frac{s}{k}\cos f \\ \frac{s}{k}\sin f \\ z + D_z\frac{s}{k}\cos f \end{bmatrix} . T = \begin{bmatrix} 1 - D_x^2 s\sin f \\ D_x s\cos f \\ -D_x D_z s\sin f \end{bmatrix}$$

最后通过振幅叠加实现多波：

$$P_x = x + \sum_{i=1}^{n} D_{ix} \frac{s_i}{k_i} \cos f_i, \text{ and } f_i = k\left(D_i \cdot \begin{bmatrix} x \\ z \end{bmatrix} - ct\right)$$

效果如下：



5. 场景建模+操作说明

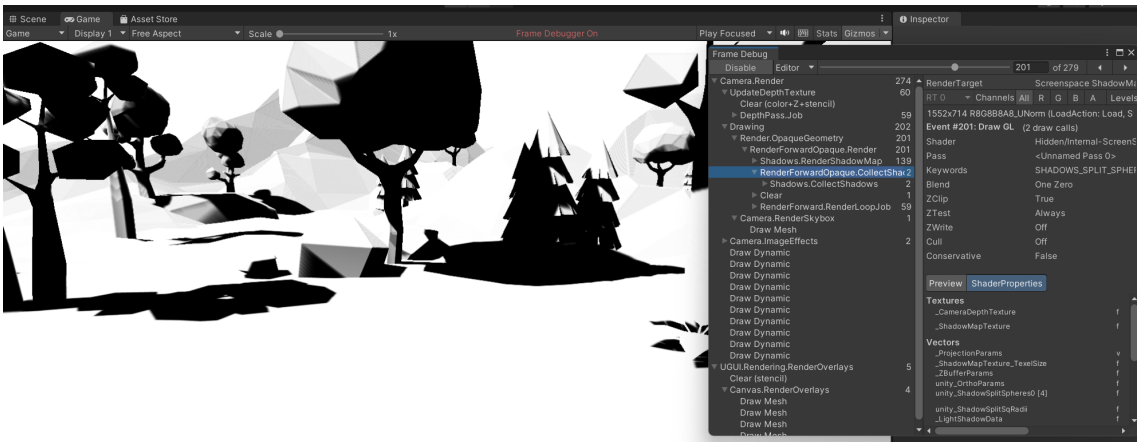
- 引入美术素材包建模游戏场景（森林天空盒），素材包URL：https://assetstore.unity.com/packages/3d/environments/landscapes/low-poly-simple-nature-pack-162153#description
- 按照作业要求搭建一个类似minecraft的游戏，实现一个支持人物移动、从物品栏选择物品、放置物品于地面的3D游戏：
  - [x] 鼠标左键放置自定义Cube
  - [x] 鼠标右键删除放置的自定义Cube
  - [x] 鼠标移动转换视角
  - [x] 键盘↑↓←→控制玩家移动
  - [x] 共有OutlineBasic和OutlineStone两种材质的自定义Cube，通过鼠标滚轮循环切换放置Cube
  - [x] 屏幕中央十字瞄准Cube位置，选中的Cube会触发描边效果
  - [x] 用waves shader渲染场景中的固定湖面

6. Debug工具使用（Unity Frame Debugger ）

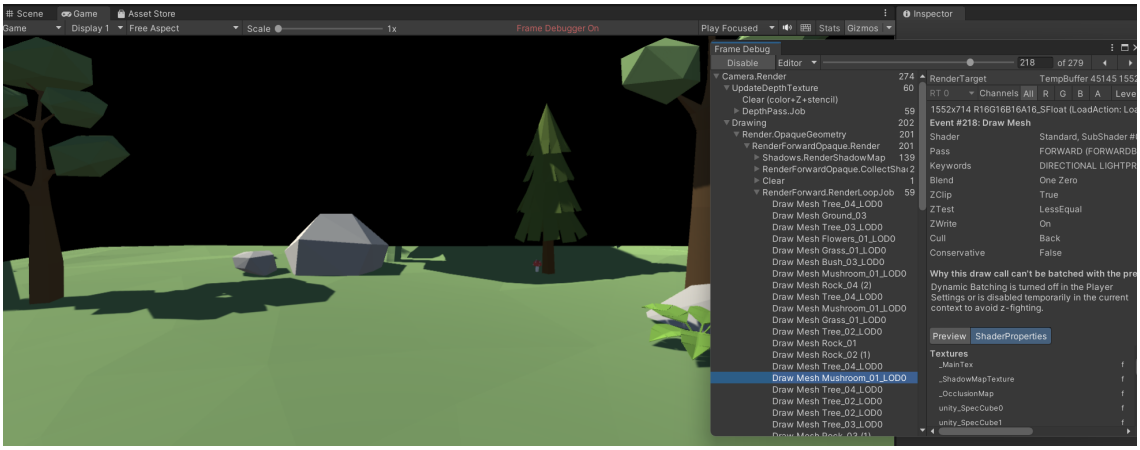
主列表以层级视图形式显示绘制调用（以及其他事件，例如帧缓冲区清除）的序列，并在其中标识绘制调用的来源。列表右侧面板提供有关绘制调用的更多信息，例如几何体细节和用于渲染的着色器。

单击列表中的某项将显示该场景（在 Game 视图中）截止到并包括该绘制调用的情况。工具栏中的左右箭头按钮用于在列表中向前和向后移动一步，您也可以使用箭头键来实现相同效果。此外，窗口顶部的滑动条可让您在绘制调用中快速拖动来迅速定位要关注的事项。如果绘制调用对应于游戏对象的几何体，则会在主 Hierarchy 面板中突出显示该对象以便于识别。

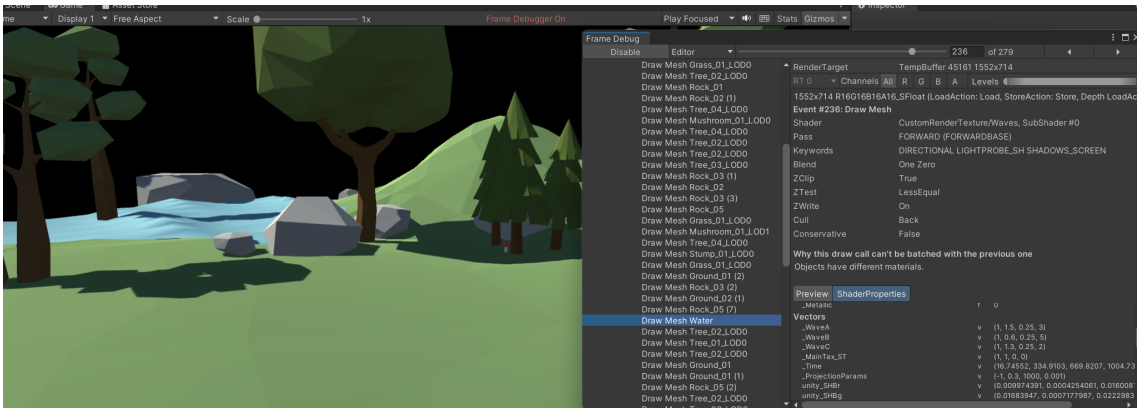
点击左边的不同步骤可以看到渲染的过程。绘制流程为线框，黑白，彩色，UI：



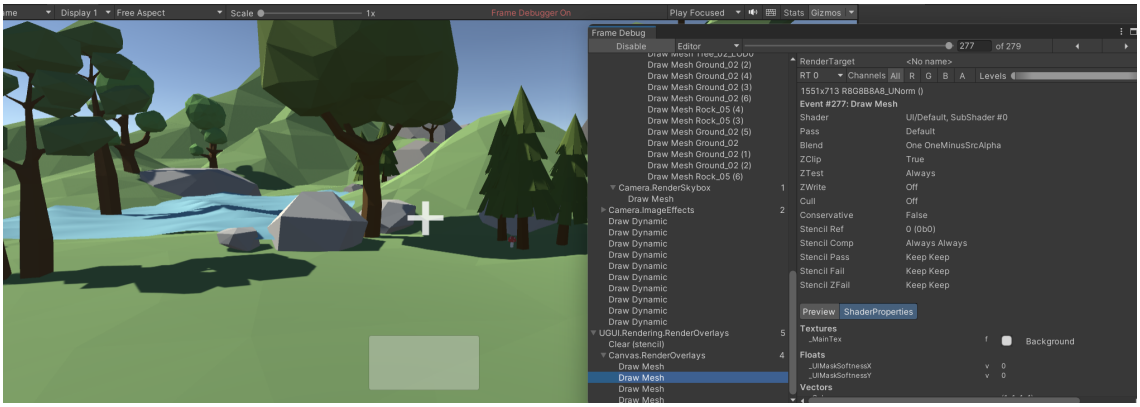
所有物体的绘制有明确先后顺序：



如water的绘制，右边可看到shader的各项参数值：



最后是UI绘制：



7. 参考文档

【4.1】https://www.ronja-tutorials.com/post/020-hull-outline/

【4.2】https://catlikecoding.com/unity/tutorials/flow/waves/

【6】https://docs.unity3d.com/2019.2/Documentation/Manual/FrameDebugger.html