**1 自定义渲染管线**

* 1. **自定义渲染管线**

本系列教程参考了Jsper Flick的《Custom SRP》系列文章，可以去他的网站学习更多技术。

本课程参考书籍《Unity Shader入门精要》和《Unity3D 内建着色器源码剖析》等。

推荐雨松的《URP从原理到应用》教程。

**1.1.1 前置工作**

1. 通过Player Setting将Color Space由Gamma空间切换为Linear线性空间；
2. 从PackageManager下载扩展包；

**1.1.2 新建渲染管线资产**

**1.1.3 创建渲染管线实例**

**1.2 正式渲染**

ScriptableRenderContext是SRP用于渲染的最底层接口之一，还有一个接口叫做CommandBuffer。

**1.2.1 相机渲染**

**1.2.2 绘制天空盒**

通过调用ScriptableRenderContext渲染接口的DrawSkybox()来绘制一个天空盒。通过ScriptableRenderContext发送的渲染命令都是缓冲的，最后需要通过调用Submit()方法来正式提交渲染命令。

但现在我们还无法控制相机，通过设置相机的Transform旋转发现毫无作用，Scene窗口右下角的Camera Preview视图也没有任何变化。因为我们还需要设置视图-投影变换矩阵，我们通过context.SetupCameraProperties方法来设置矩阵和相机的其他属性。

**1.2.3 CommandBuffer**

某些任务，比如绘制天空盒，可以直接调用context的专用方法发出命令，而其它命令需要通过单独的命令缓冲区（CommandBuffer）间接发出，我们需要这样一个缓冲区来绘制场景中其它几何图形。CommandBuffer是一个容器，它保存了这些将要执行的渲染命令。

执行缓冲区命令是通过context.ExecuteCommandBuffer(buffer)来执行，这个操作会从缓冲区复制命令但不会清除缓冲区，我们如果要重用buffer，一般会在执行完该命令后调用Clear()清除。

**1.2.4 清除渲染目标**

在内存中对应分配着存储帧数据的缓冲区，包括写入颜色的颜色缓冲（Color Buffer）、写入深度值的深度缓冲（Depth Buffer） 以及基于一些条件丢弃片元的模板缓冲（Stencil Buffer），最后还包括自定义的缓冲区，这几种缓冲一起称之为帧缓冲。

为了保证下一帧绘制的图像正确，我们通常要清除渲染目标，清除旧的数据。该操作通过调用buffer.ClearRenderTarget方法来完成。

**1.2.5 剔除**

我们只需要渲染在相机视野内的物体，视野外的物体需要剔除掉。这一步主要通过camera.TryGetCullingParameters方法得到需要进行剔除检查的所有物体，正式的剔除是通过context.Cull()实现的，

最后会返回一个CullingResults的结构，里面存储了我们相机剔除后的所有视野内可见物体的数据信息。

**1.2.6 绘制几何体**

调用context.DrawRenderers方法进行正式绘制。它需要三个参数：CullingResults，DrawingSettings和FilteringSettings。

DrawingSettings用于设置是哪个Shader的哪个Pass进行渲染。

在SRP中，旧的着色器大部分基本不能再使用，但没有光照的内置着色器Unlit被保留了下来，我们需要获取Pass名字为SRPDefaultUnlit的着色器标识ID。

SortSettings的作用是确定相机的透明排序模式是否使用正交或基于距离的排序。

FilteringSettings，用于过滤给定的一组可见对象以便渲染。

**1.2.7 透明和不透明几何分开绘制**

一般情况下，我们应当遵守 不透明物体->绘制天空盒->绘制透明物体 的绘制顺序。

1.3 编辑器渲染