Draw Call

着色器语言

最常见的三种着色器语言分别是微软的HLSL，OpenGL的GLSL和NVIDIA的CG。CG语言已经停止更新很多年了，基本被废弃了。现在SRP的着色器代码库使用的是HLSL。

Draw Call和Set Pass Call

要想CPU和GPU既可以并行又可以独立工作，要使用一个命令缓冲区（Command Buffer）。命令缓冲区的命令有很多种类，而Draw Call就是其中一种，其它命令还有Set Pass Call等等。Set Pass Call代表了改变渲染状态，当切换材质或者切换同一材质中Shader的不同Pass进行渲染时都会触发一次Set Pass Call。Set Pass Call比Draw Call更耗时，所以SRP不再支持多Pass。

SRP Batcher

早期的Unity只支持动态批处理和静态批处理，后来又支持了GPU Instancing，SRP出现时支持了一种新的批处理方式SRP Batcher。

SRP Batcher不会减少Draw Call的数量，但可以减少Set Pass Call的数量，并减少Draw Call开销。

CPU不需要每帧都给GPU发送渲染数据，如果这些数据没有发生变化则会保存在GPU内存中，每个绘制调用仅需包含一个指向正确内存位置的偏移量。

SRP Batcher是否会被打断的判断依据是Shader变种，即使物体之间使用了不同的材质，但是使用的Shader变种相同就不会被打断，传统的批处理方式是要求使用同一材质为前提的。

SRP Batcher会在主存中将模型的坐标信息、材质信息、主光源阴影参数和非主光源阴影参数分别保存到不同的CBUFFER（常量缓冲区）中，只有CBUFFER发生变化才会重新提交到GPU并保存。

GPU Insatncing

如果能将数据一次性发送给GPU，然后使用一个绘制函数让渲染流水线利用这些数据绘制多个相同的物体将会大大提升性能。这种技术就是GPU多例化（GPU Instancing）技术。

使用GPU Instancing能够在一个绘制调用中渲染多个具有相同网格的物体，CPU收集每个物体的材质属性和变换，放入数组发送到GPU，GPU遍历数组按顺序进行渲染。

假设需要渲染100个相同的模型，每个模型有256个三角形，那么需要两个缓冲区，一个是用来描述模型的顶点信息，因为待渲染的模型是相同的，所以这个缓冲区只存储了256个三角形（如果不存在任何的优化组织方式，则有768个顶点）；另一个就是用来描述模型在世界坐标下的位置信息。例如不考虑旋转和缩放，100个模型即占用100个float3类型的存储空间。

如何支持GPU Instancing？

1. 要支持GPU Instancing，首先需要在Shader的Pass中添加#pragma multi\_compile\_instancing指令，然后在材质球上就能看到切换开关了，这时Unity会为我们的Shader生成两种变体。
2. 在Common.hlsl文件中include SpaceTransforms.hlsl之前，我们将SRP源码库中的UnityInstancing.hlsl文件Include进来，我们需要用到里面的一些定义好的宏和方法。
3. UnityInstancing.hlsl通过重新定义一些宏去访问实例的数据数组，它需要知道当前渲染对象的索引，该索引是通过顶点数据提供的。UnityInstancing.hlsl中定义了UNITY\_VERTEX\_INPUT\_INSTANCE\_ID宏来简化了这个过程，但它需要我们存在一个顶点输入结构体，我们定义它并将positionOS的定义放进来，然后在结构体中加入UNITY\_VERTEX\_INPUT\_INSTANCE\_ID宏，最后该结构体对象作为顶点函数的输入参数。然后在顶点函数添加UNITY\_SETUP\_INSTANCE\_ID(input)代码，用来提取顶点输入结构体中的渲染对象的索引，并将其存储到其他实例宏所依赖的全局静态变量中。
4. 目前我们还不支持每个物体实例的材质数据，且SRP Batcher优先级比较高，我们还不能得到想要的结果。首先我们需要使用一个数组引用替换\_BaseColor，并使用UNITY\_INSTANCING\_BUFFER\_START和UNITY\_INSTANCING\_BUFFER\_END替换CBUFFER\_START和CBUFFER\_END。
5. 我们还需要在片元函数中也提供对象的索引，通过在顶点函数中使用UNITY\_TRANSFER\_INSTANCE\_ID(input，output)将对象位置和索引输出，若索引存在则进行复制。为此我们还需定义一个片元函数输入结构体，在其中定义positionCS和UNITY\_VERTEX\_INPUT\_INSTANCE\_ID宏。
6. 在片元函数中也定义UNITY\_SETUP\_INSTANCE\_ID(input)提供对象索引，且现在需要通过UNITY\_ACCESS\_INSTANCED\_PROP(UnityPerMaterial, \_BaseColor)来访问获取材质的颜色属性了。

动态合批

动态批处理的原理是每一帧把可以进行批处理的模型网格进行合并，再把合并好的数据传递给CPU，然后使用同一个材质进行渲染。好处是经过批处理的物体仍然可以移动，这是由于Unity每帧都会重新合并一次网格。

动态批处理有很多限制，比如在使用逐对象的材质属性时会失效，网格顶点属性规模要小于900等等，该技术适用于共享材质的小型的网格。

透明

在Unity中我们通常使用两种方法来实现透明效果：第一种是透明度测试（Alpha Test），这种方法其实完全无法得到真正的半透明效果；另一种是透明度混合（Alpha Blend）。

透明度测试会使得硬件底层的优化技术Early-Z失效。

Shader feature

使用shader feature可以让Unity根据不同的定义条件或关键字编译多次，生成多个着色器变体。然后通过外部代码或者材质面板上的开关来启用某个关键字，加载对应的着色器变种版本来执行某些特定功能，是项目开发中比较常用的一种手段。下面我们的目标是添加一个控制透明度测试功能是否启用的开关。