**GPU Instancing**

**GPU Instancing的作用是什么？**

使用GPU Instancing能够在一个Draw Call中渲染多个具有相同网格的物体。

**GPU Instancing的工作原理是什么？**

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；另一个就是用来描述模型在世界坐标下的位置信息的数组。

**GPU Instancing的实现步骤**

1. 要支持GPU Instancing，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。

HLSLPROGRAM  
 #pragma multi\_compile\_instancing

2. 需要知道渲染对象的索引，该索引是通过顶点数据提供的。UnityInstancing.hlsl中定义了UNITY\_VERTEX\_INPUT\_INSTANCE\_ID宏来简化这个过程，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。

//用作顶点函数的输入参数  
struct Attributes   
{  
 float3 positionOS : POSITION;  
 UNITY\_VERTEX\_INPUT\_INSTANCE\_ID  
};  
//顶点函数  
float4 UnlitPassVertex(Attributes input) : SV\_POSITION  
{  
 UNITY\_SETUP\_INSTANCE\_ID(input);  
 float3 positionWS = TransformObjectToWorld(input.positionOS);  
 return TransformWorldToHClip(positionWS);  
}

3.使用UNITY\_INSTANCING\_BUFFER\_START和UNITY\_INSTANCING\_BUFFER\_END替换CBUFFER\_START和CBUFFER\_END。

UNITY\_INSTANCING\_BUFFER\_START(UnityPerMaterial)  
UNITY\_DEFINE\_INSTANCED\_PROP(float4, \_BaseColor)  
UNITY\_INSTANCING\_BUFFER\_END(UnityPerMaterial)

4. 需要在片元函数中提供对象的索引，通过使用UNITY\_TRANSFER\_INSTANCE\_ID(input，output)将对象索引输出，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。

//用作片元函数的输入参数  
struct Varyings   
{  
 float4 positionCS : SV\_POSITION;  
 float2 baseUV : VAR\_BASE\_UV;  
 UNITY\_VERTEX\_INPUT\_INSTANCE\_ID  
};  
Varyings UnlitPassVertex (Attributes input)   
{  
 Varyings output;  
 UNITY\_SETUP\_INSTANCE\_ID(input);  
 UNITY\_TRANSFER\_INSTANCE\_ID(input, output);  
 float3 positionWS = TransformObjectToWorld(input.positionOS);  
 output.positionCS = TransformWorldToHClip(positionWS);  
 return output;  
}

5. 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。