2024年12月8日星期日

GPUInstancing学习笔记

参考连接：<https://github.com/vanCopper/Unity-GPU-Instancing>

<https://docs.unity3d.com/cn/2019.3/Manual/GPUInstancing.html>

github网页中的shader文件存在问题，会导致批处理时无法正确的访问到对象的颜色，具体修改参考unity官方文档。

没有什么算法，仅是一些知识点以及shader语法

unity中的批处理：

Static Batching：

StaticBatching会在Build阶段提取多个使用相同材质的，且不会变换矩阵不会发生变化的模型的Vertex buffer和index buffer。然后将顶点数据变换到世界空间下，存储到最终Static Batching所使用的Vertex buffer中，并记录每个子模型的Index buffer在Static Batching所使用的Index buffer中的位置。在绘制阶段，会一次性提交合并后的模型顶点数据，最终引擎会根据子模型的可见性确定需要绘制的子模型，设置一次渲染状态，在调用多次Draw Call分别绘制子模型。

所以静态批处理并没有减少draw call的数量，但在绘制阶段避免了多次数据提交和渲染状态的切换。

Dynamic Batching

动态批处理会在运行时将使用同一材质的模型进行合并渲染，也就是将符合条件的Game Object放在一个Draw Call中绘制。使用动态批处理有一些限制：

i：模型最高900个顶点属性，300个顶点。假如我们的Shader中每个顶点使用了Position、Normal、UV，那么模型就只能有300个顶点，如果使用了Position、Normal、UV0、UV1、Tangent，那么顶点数就要减少到180个（900/5）

ii、Game object之间有镜像变化的不能合批；

iii、使用多通道shader的对象不能合批；

iv、拥有光照贴图的对象不能合批；

v、对象接收实时阴影无法合批。

动态批处理在降低DrawCall的同时会导致CPU性能消耗，所以仅在合批操作的性能消耗不小于合批，动态批处理才有意义。

# unityshader

## #pragma multi\_compile\_instancing

//用于开启GPUInstancing支持，当在Shader中添加这行代码之后，会在unity 编辑器中的inspector界面看到Enable GPUInstancing选项

图形用户界面, 文本, 应用程序

描述已自动生成

## UNITY\_VERTEX\_INPUT\_INSTANCE\_ID

在shader输入、输出结构体中声明一个对象ID

struct appdata

{

//声明一个instance id

UNITY\_VERTEX\_INPUT\_INSTANCE\_ID

};

struct v2f

{

UNITY\_VERTEX\_INPUT\_INSTANCE\_ID // 仅当您要访问片元着色器中的实例化属性时才需要。

};

如果想要访问实例化的属性如颜色，则必须要在输出结构体中添加UNITY\_VERTEX\_INPUT\_INSTANCE\_ID

## 声明实例化属性

即如果希望每个对象拥有不同的状态，如颜色，则需要在此定义这些属性 ；

UNITY\_INSTANCING\_BUFFER\_START(Props)

UNITY\_DEFINE\_INSTANCED\_PROP(fixed4 , \_Color)

//其他属性

UNITY\_INSTANCING\_BUFFER\_END(Props)

如上述的代码，定义了类型为fixed4，名字是\_Color的实例化属性。

## 实例化属性的使用

v2f vert(appdata v)

{

v2f o;

UNITY\_SETUP\_INSTANCE\_ID(v); // 获取实例 ID

UNITY\_TRANSFER\_INSTANCE\_ID(v, o); // 将实例 ID 传递给片段着色器

o.vertex = UnityObjectToClipPos(v.vertex);

o.uv = TRANSFORM\_TEX(v.uv, \_MainTex);

return o;

}

fixed4 frag(v2f o) : SV\_Target

{

UNITY\_SETUP\_INSTANCE\_ID(o); // 片段着色器中也需要获取实例 ID

fixed4 instancedColor = UNITY\_ACCESS\_INSTANCED\_PROP(Props, \_Color); // 获取该实例的颜色

return instancedColor; // 返回实例化的颜色

}

1. 在顶点着色器中的UNITY\_SETUP\_INSTANCE\_ID(V);

这行代码在顶点着色器中为每个实例设置一个唯一的ID

实例ID是一个整数值，用来唯一标识当前渲染的实例。在GPUInstancing中模式下，多个相同的材质会共享同一个绘制调用，当每个实例都会有一个唯一的ID，以便区分不同的实例并访问它们的独特属性，如位置、颜色等。

1. UNITY\_TRANSFER\_INSTANCE\_ID(v,o);

这行代码将实例ID从输入的v传递到o中，确保片段着色器或其他阶段的着色器可以访问到该实例的ID。

1. 在片元着色器中的UNITY\_SETUP\_INSTANCE\_ID(o);

确保每个像素都能正确的访问到当前的唯一实例ID，从而可以在片元着色器中修改对应的实例属性如Color；

以上三行代码缺一不可。

如果确定不用修改对象的属性如颜色，则仅需要1中的代码即可。

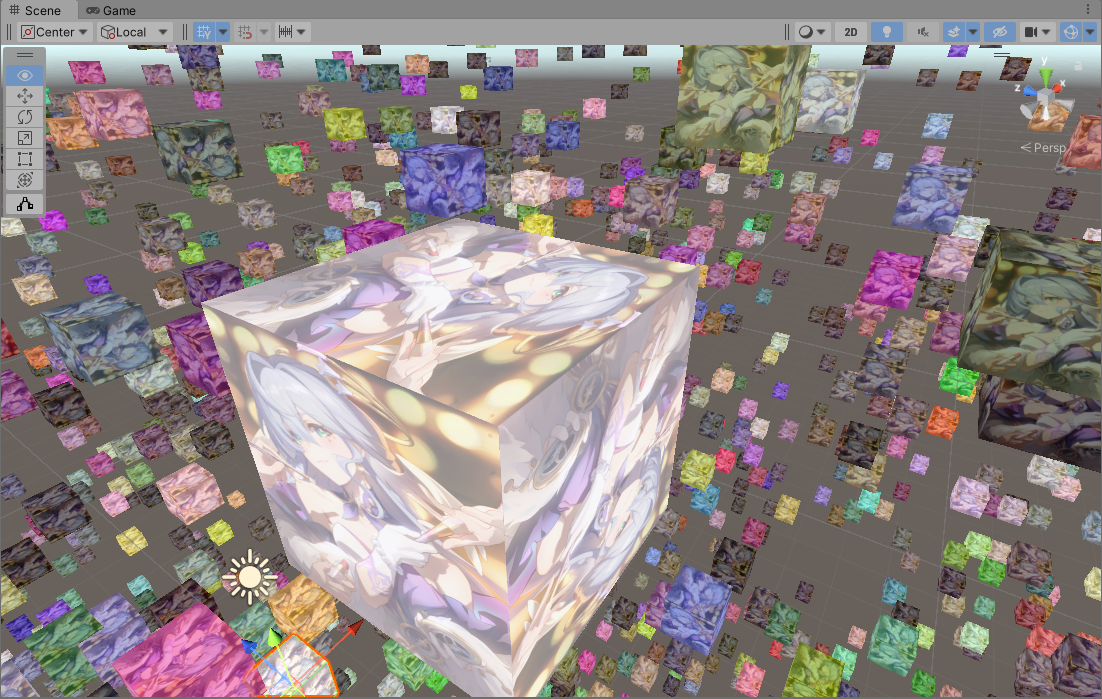
## 在unity C#脚本中修改对应的实例属性

MaterialPropertyBlock block = new();

block.SetColor(“\_Color” , Random.ColorHSV());

intanceObj.GetComponent<MeshRenderer>().SetPropertyBlock(block);

上述代码修改了shader中的实例化属性\_Color。



图形用户界面, 应用程序

描述已自动生成