<http://thomasmountainborn.com/2016/05/25/materialpropertyblocks/>

<http://www.jianshu.com/p/eff18c57fa42>

在unite2017 shader性能与优化专题中提到了这样的优化，建议就是使用材质属性块，其原话是：

Use MaterialPropertyBlock is faster to set properties using a MaterialPropertyBlock rather than maerial.SetFloat(); Material.SetColor();

MaterialPropertyBlock的官方文档是这样说的，材质属性块被用于Graphics.DrawMesh和Renderer.SetPropertyBlock两个API两个API，当我们想要回执许多相同材质但不同属性的对象时可以使用它。例如你想改变每个绘制网格的颜色，但是它却不会改变渲染器的状态。

我们来看看Renderer这个类，它包含了material、sharedMaterial这两个属性；GetPropertyBlock、SetPropertyBlock两个函数，其中两个属性是用来访问和改变材质的，而两个函数是用来设置和获取材质属性块的，我们知道，当我们操作材质共性时，可以使用sharedMaterial属性，改变这个属性，那么所有使用此材质的物件都将会改变，而我们需要改变单一材质时，需要使用material属性，而在第一次使用material时其实是会生成一份材质拷贝的，即material(Instance)

可以用SetpropertyBlock方法来实现多个物体共享材质，可以有部分属性不一样的情况

它不像sharedMaterial所有的共同拥有，也不同material每个都会自己拷贝一份，它可以实现有部分属性不一样，但是不会每个Renderer都拷贝一份

这种方式被用在unity的地形上，里面的树就是通过这种方式实现不同树共享材质的，却可以有不同的颜色风力等，这种方式可以在一定程度上简化材质数量，提高效率。

public class TestMaterial : MonoBehaviour

{

private const int COUNT = 1000;

private GameObject cube;

private GameObject[] listObj = new GameObject[COUNT];

private GameObject[] listProp = new GameObject[COUNT];

private MaterialPropertyBlock prop;

void Awake()

{

prop = new MaterialPropertyBlock();

for (int i = 0; i < COUNT; i++)

{

listObj[i] = GameObject.CreatePrimitive(PrimitiveType.Cube);

listObj[i].name = "listObj";

listObj[i].transform.parent = GameObject.Find("Obj").transform;

listProp[i] = GameObject.CreatePrimitive(PrimitiveType.Sphere);

listProp[i].name = "listProp";

listProp[i].transform.parent = GameObject.Find("Prop").transform;

}

}

void Update()

{

if (Input.GetKeyDown(KeyCode.DownArrow))

{

var startTime = Time.realtimeSinceStartup;

for (int i = 0; i < COUNT; i++)

{

float r = Random.Range(0.0f, 1.0f);

float g = Random.Range(0.0f, 1.0f);

float b = Random.Range(0.0f, 1.0f);

listObj[i].GetComponent<Renderer>().material.SetColor("\_Color", new Color(r, g, b, 1));

}

var endTime = Time.realtimeSinceStartup;

Debug.Log("meterial time = " + (endTime - startTime));

}

if (Input.GetKeyDown(KeyCode.UpArrow))

{

var startTime = Time.realtimeSinceStartup;

for (int i = 0; i < COUNT; i++)

{

float r = Random.Range(0.0f, 1.0f);

float g = Random.Range(0.0f, 1.0f);

float b = Random.Range(0.0f, 1.0f);

**listProp[i].GetComponent<Renderer>().GetPropertyBlock(prop);**

**prop.SetColor("\_Color", new Color(r, g, b, 1));**

**listProp[i].GetComponent<Renderer>().SetPropertyBlock(prop);**

}

var endTime = Time.realtimeSinceStartup;

Debug.Log("materialpropertyblock time = " + (endTime - startTime));

}

}

}

初始化：



按下下箭头





按下上箭头





会发现material在初始化的时候会动态创建material，而materialpropertyblock这虽然改变了属性，但是并没有创建新的material

别人的结论：



从结果对比来看，确实使用材质属性块要快于使用材质，其消耗将近是操作材质耗时的四分之一。同时不管是材质还是材质属性块，第一次操作比后面的操作耗时要大，尤其是材质，可见在第一次使用材质改变属性操作时，其拷贝操作消耗还是非常大的。同时我也通过profiler的memory模块，切换进Detailed选项，对其进行采样，可以发现在Sence Memory下面会有material的拷贝（材质操作导致，而材质属性操作不会）

**正如官方文档介绍材质属性块一样，unity地型引擎正是使用材质属性块来绘制树的，所有的树使用的是相同的材质，但是每棵树有不同的颜色，缩放和风因子。对于大场景大世界来说，我们肯定是采取动态加载来处理地图的，这个时候我们可以配合Gpu Instance来进一步的提高性能，使用Gpu Instance一是可以省去实体对象本身的开销，二是能够起到减少Drawcall的作用，同时还能减少动态合批的cpu开销，静态合批的内存开销；可谓一举多得，遗憾的只能在ES3.0以上的设备上使用。对于一些游戏中存在自定义皮肤颜色玩法的，材质属性块的优势就可以发挥出来了，你想当你100个不同玩家同屏时，如果使用材质操作颜色属性的话，那么首先就存在100份材质拷贝的实例，其次，材质操作属性本身就比材质属性块操作要慢那么点，在性能优化中一毫秒的优化就是胜利，这了一毫秒那里一毫秒加起来就不得了了。**