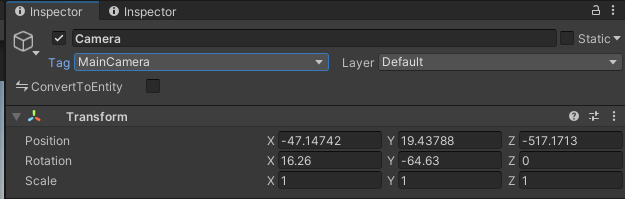
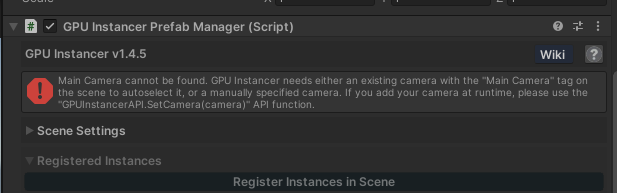
**注意主摄像头必须设置为MainCamera,**

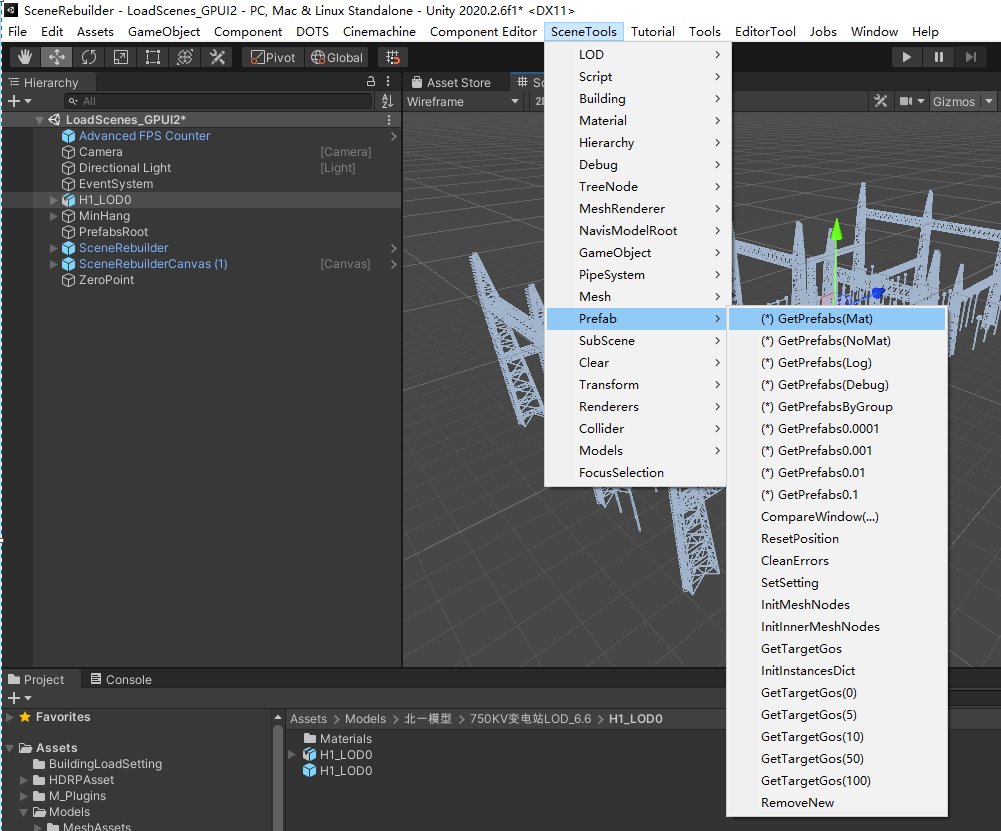


**不然运行时会出现提示，并且不会显示GPUI渲染的模型。**

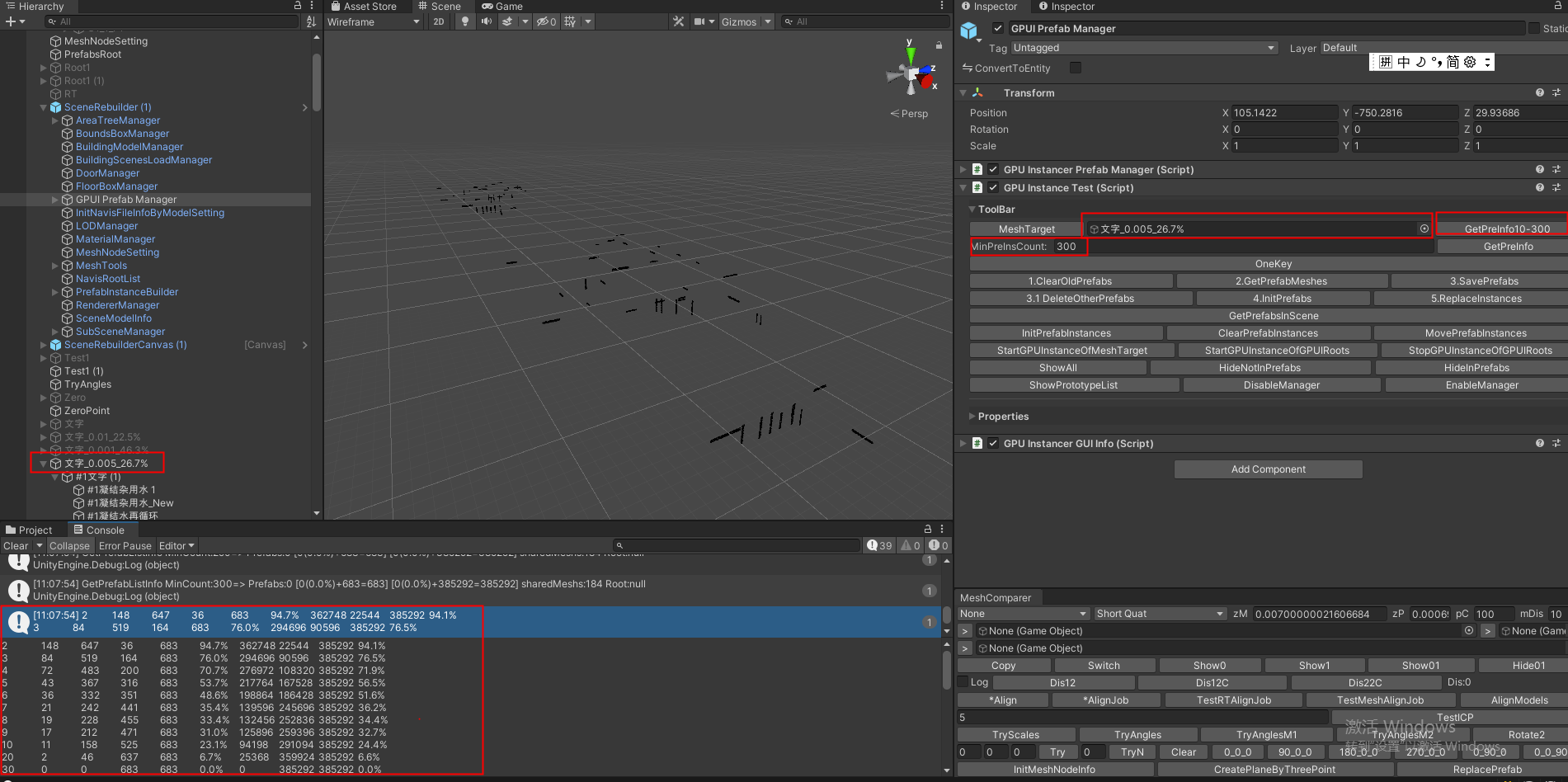


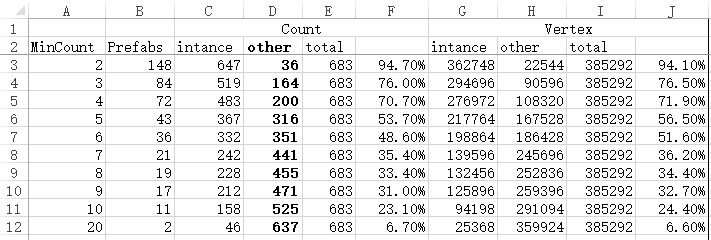
一、操作说明（从新处理）

1. 将模型进行过复用处理

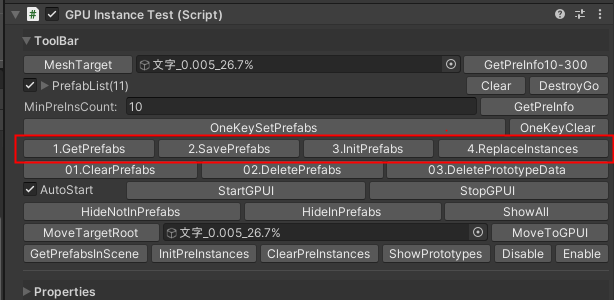


1. 将需要处理的模型的对象根节点拖到GPUInstanceTest.MeshTarget中，点击GetPreInfo10-300，判断一下不同的MinPreInsCount设置的情况下，有多少模型放入GPUI渲染。





1. 设置一个MinPreInsCount，这个参数指的是一个模型的复用数量，大于这个数值的模型设置成GPUI渲染，其他的模型继续默认渲染。复用率高的模型放到GPUI渲染中，复用率的模型就不用放了。比如这里可能应该设置为3或者4吧，先设置10测试一下。
2. 分步设置操作

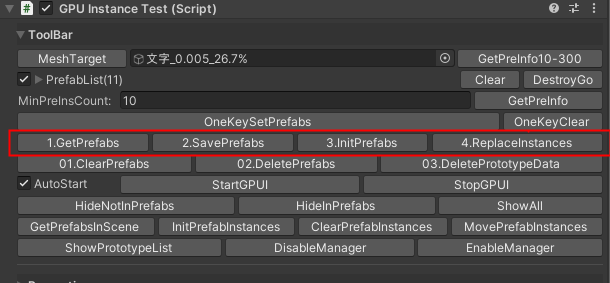


* 1. 1.GetPrefabes

MinPreInsCount设置太高的情况下，或者说目标模型的复用数量很低的情况下，没有效果，会提示



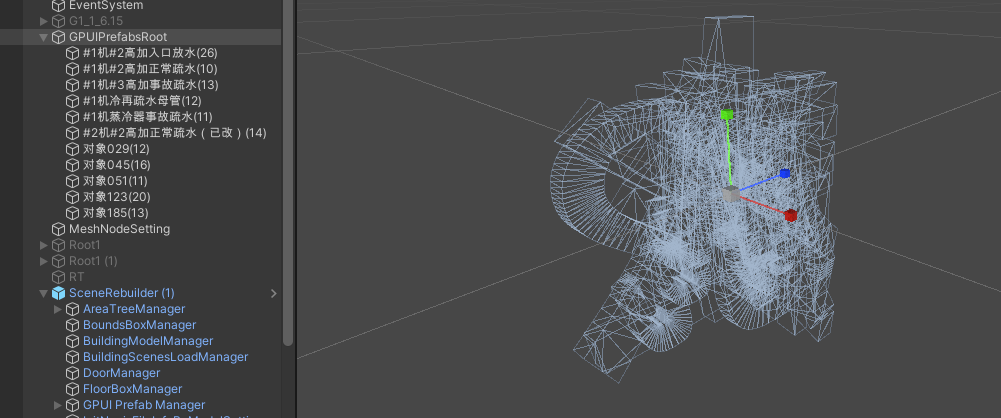
设置的数值合适的情况下，会把大于该值的模型获取并列出来。





这个信息代表有11个模型的复用数大于10，这些模型的总数是158，占原来模型的总体数量的23.1%。

同时会在场景中创建一个PrefabsRoot根节点，拷贝一份这些模型，放到该节点下面。

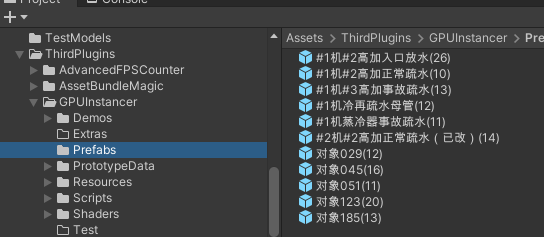


重复点击的情况下，不会有影响，会判断是否已经存在，没有存在则添加的。

* 1. SavePrefabs

将上述的模型创建预设文件，放入文件夹中。因为GPUInstance功能似乎是基于预设的。

预设文件会被保存在Assets/ThirdPlugins/GPUInstancer/Prefabs/



文件名称是模型Mesh名称(复用数量)。

同时会删除场景中GPUIPrefabsRoot下面的模型，PrefabList内关联的变成了这些预设文件了。

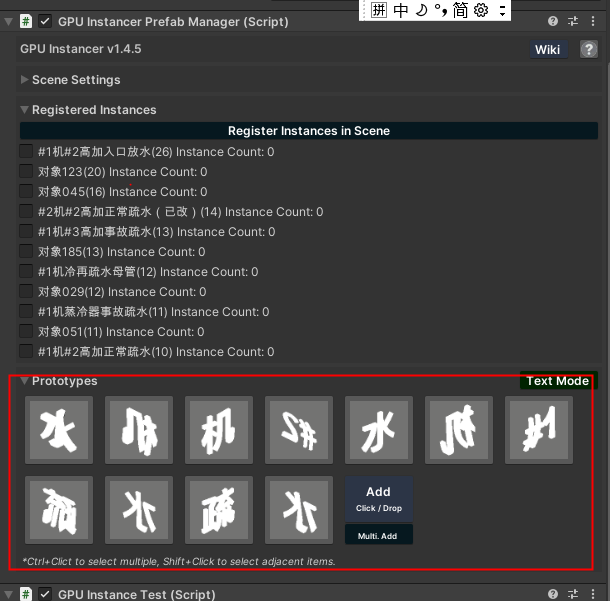
* 1. DeleteOtherPrefabs

一般不用操作，清理以前的老的预设文件。

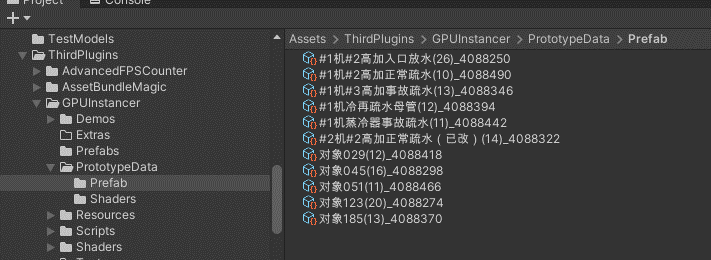
* 1. InitPrefabs

将这些预设文件放入GPUInstance脚本GPUInstancerPrefabManager。

设置的结果就是Prototypes列表中加上了刚刚的模型预设，Instances中也有。

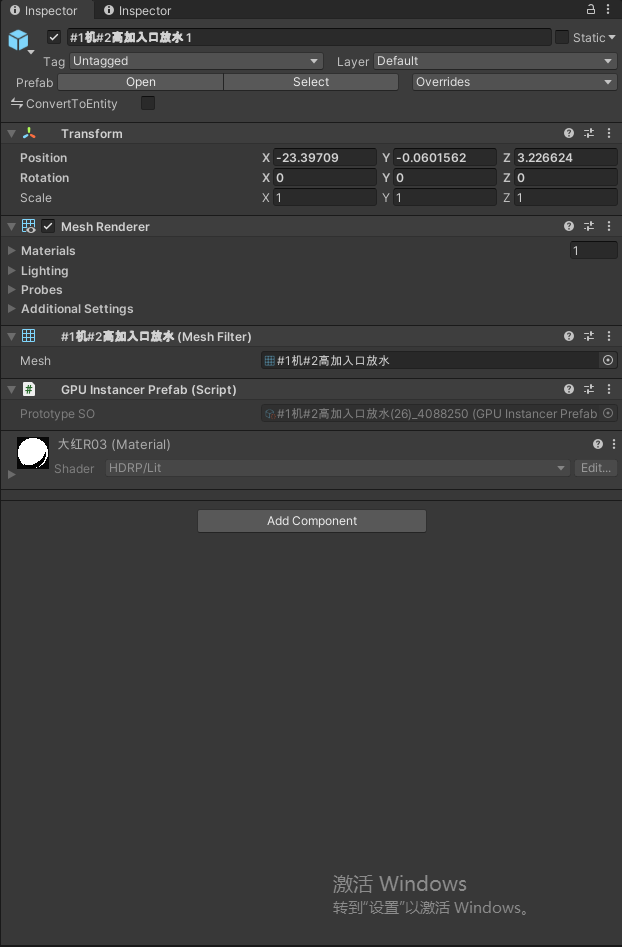


在GPUInstancer/PrototypeData/Prefab/文件夹中也有了相应的文件。



* 1. ReplaceInstances

将GPUInstancePrefab脚本关联到模型上，替换后的模型属性。GPUIntance启动时会将这些GPUInstancerPrefab模型隐藏掉并在GPUI中渲染出来。

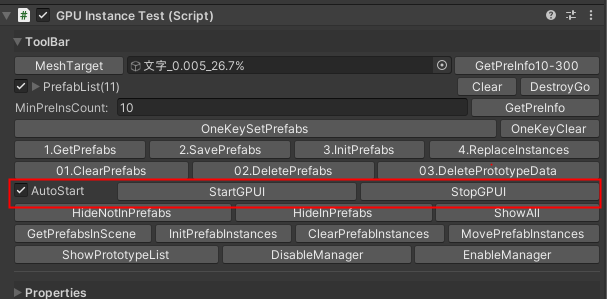


1. 一键设置操作

OneKeySetPrefabs,将前面的4步自动执行一遍。

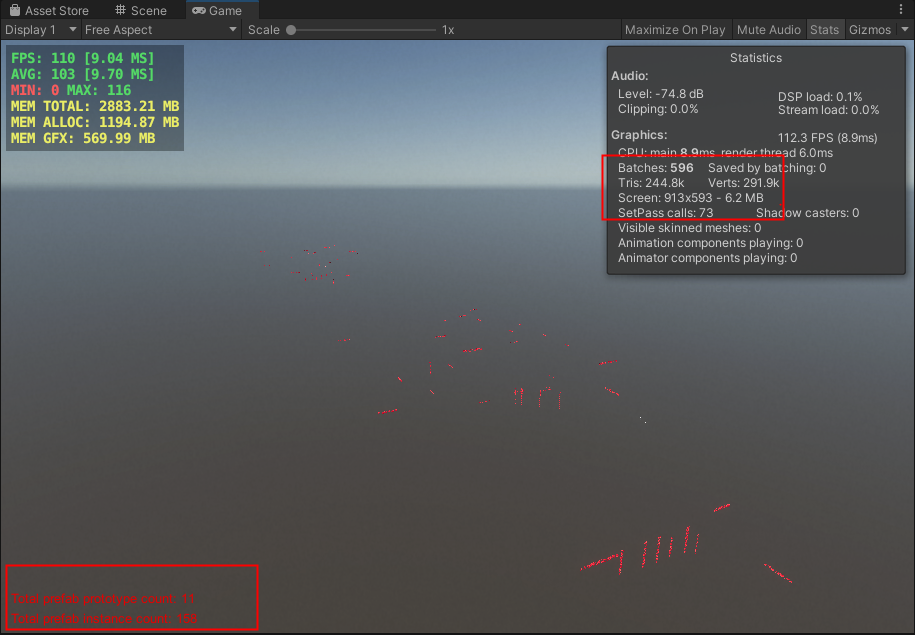
1. 运行中启动/停止GPUInstancer渲染

点击StartGPUI/StopGPUI按钮，或者设置为启动就调用，或者程序中代码调用开始/停止。



启动后：





1. 重置清理工具
   1. ClearPrefabs：

清空PrefabList列表，GPUInstancerPrefabManager. ClearPrefabsAndPrototypes()

* 1. DeletePrfabs：

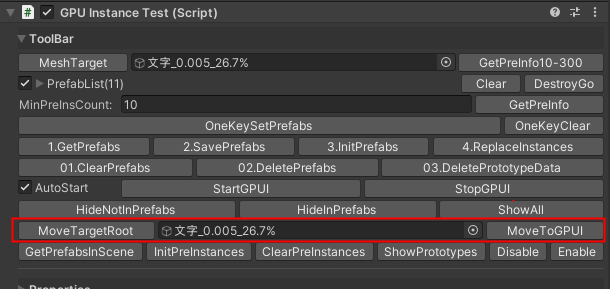
删除预设文件。Assets/ThirdPlugins/GPUInstancer/Prefabs/。

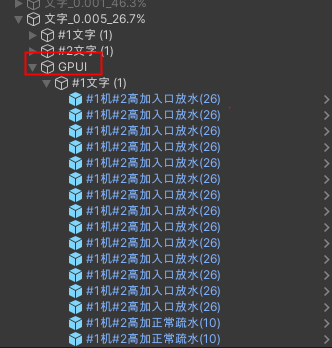
* 1. DeletePrototypeData：

删除GPUI的Prototype文件。GPUInstancer/PrototypeData/Prefab/

1. 模型处理工具
   1. MoveToGPUI：

将模型移动到GPUI节点下统一区分进行处理，保持原来的路径。



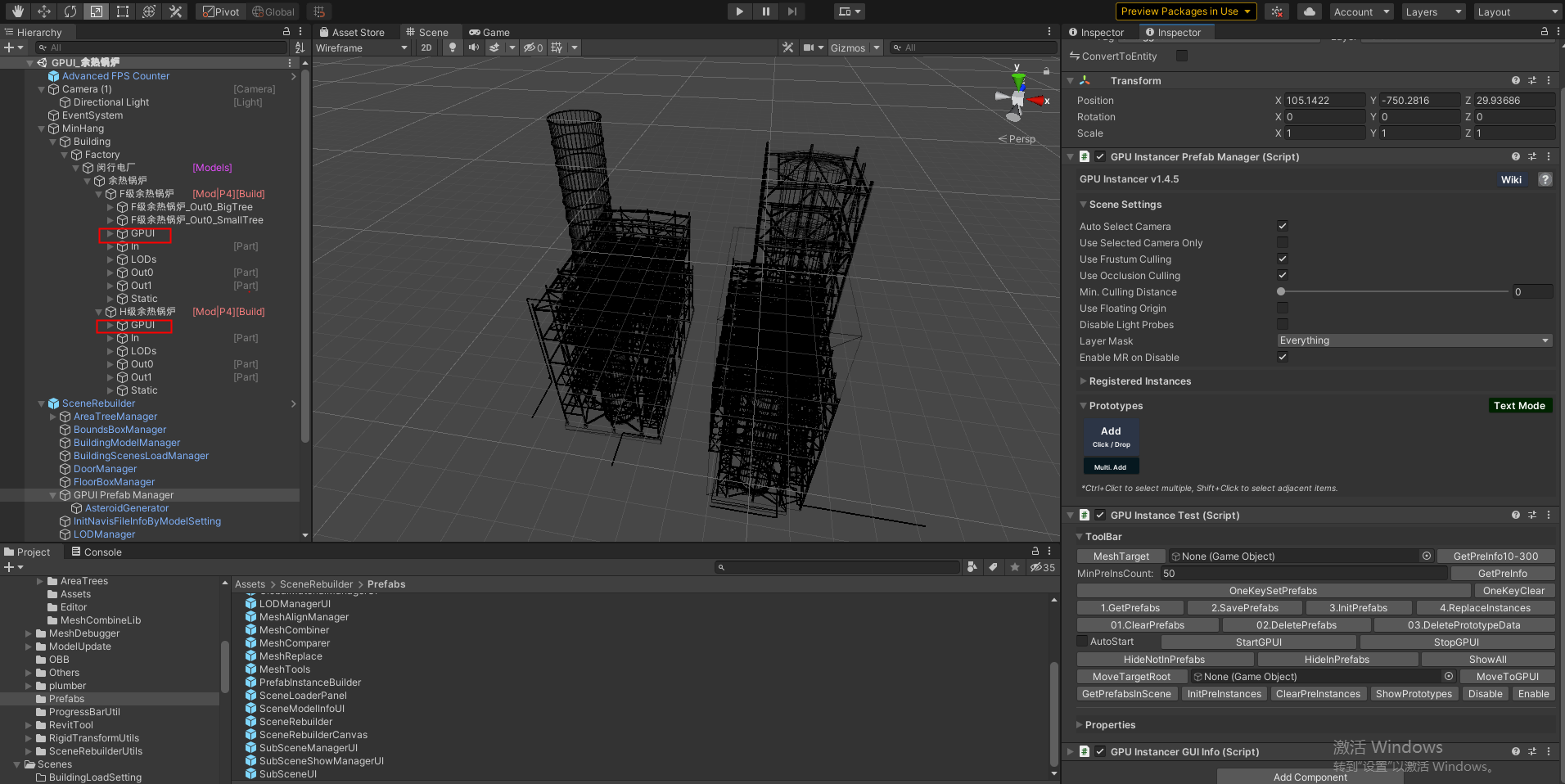


* 1. 。。。

1. 。。。。
2. 。。。。。

二、操作说明（恢复Prefabs设置）

上下文：已经做过处理，场景移动到另一台电脑上，GPUInstance没同步过来。



H级余热锅炉和F级余热锅炉大于50个复用模型的模型都已经分别移动到不同的GPUI节点下了。

1. 设置GPUIRoots。
2. 获取Prefabs
3. 保存Prefabs
4. 设置到GPUInstanceManager
5. 替换Instances。

四、性能参数说明

官方对于Draw call batch的解释：

<https://docs.unity3d.com/Manual/DrawCallBatching.html>

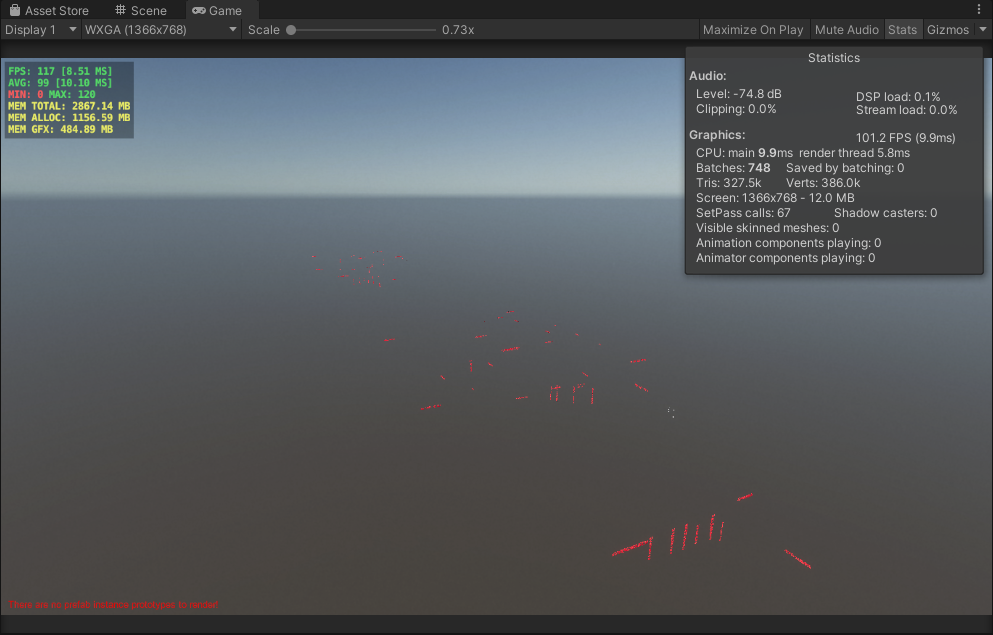
先来看Setpass Call，通俗来说，它大致代表的是摄像机照射范围内，所有GameObject所包含的Material种类数量。（不完全准确，但是这样说易于理解）。假如有30种material，那这个SetpassCall的数值也会离这个30比较接近。这个值越大，肯定会使draw call 越来越大。但是setpasscall很小的时候，也有可能draw call的值比较大。

Batches,是unity中合批次。具体的概念这里就不赘述，网上有很多资料。这个值是和Draw call 比较接近的。有时候在项目中，我们的setpass call 数值不高，但是batches出奇的高，结果也会导致draw call 过高而影响游戏的CPU性能。那这个值为什么会过高呢？ 主要是因为需要渲染的一些游戏物体不满足Unity的Dynamic batch的要求。当需要渲染的Gameobject过多，而且这些gameobject又不能被Unity 内置的Batch功能自动合并时，每多一个这样的Gameobject，就会多一个Batch，进而就会导致Draw call过高，造成卡顿。这种情况是要尽量避免的。（这里不讨论Static Batch）。那么怎么判断哪些gameobject 是可以被unity 进行动态batch的呢？有如下几点（这里图快，只贴英文）：

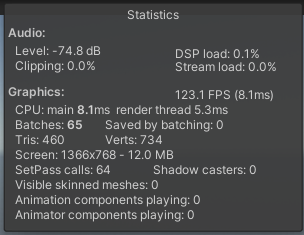
**因此优化Draw call 的时候，我们重点要将Bathes 的数值降低**

三、性能测试（一）

1. 默认渲染情况下的性能参数。



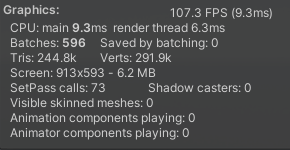
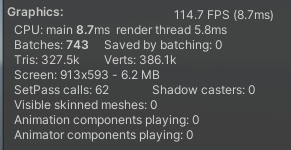
无模型时是



模型复用前后参数是一样的， 模型的复用在不结合GPUI的情况下只是减少了内存占用。

1. GPUI渲染的性能参数。

前后



这里数量少还看不出区别来。参数上Batches和Tris少了，SetPassCalls多了。

四、性能测试（二）