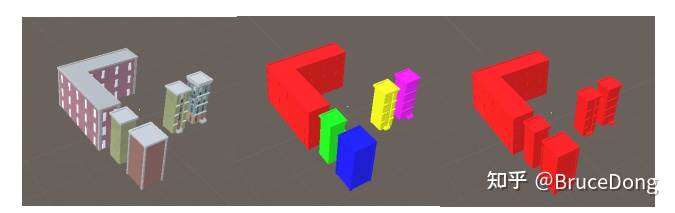
### **概述**

这项技术是用一个网格代替多个网格，当距离足够远的时候合并静态网格。这样可以有效减少场景渲染的drawcall数量，提高性能。PS:根据距离判断不一定合理，有方案根据占屏幕空间大小。下图能简单阐述HLOD的情况。



1. 减少Batches(Draw Calls)。
2. 根据HLOD级别简化三角形、贴图、材质球。
3. 减少内存的使用，减少加载时间。仅仅加载当前等级、流式&异步加载Mesh/Texture。

|  |  |  |
| --- | --- | --- |
|  | LOD（per Object） | HLOD(per Group) |
| 简化三角形面数 | √ | √ |
| 减少DC | × | √ |
| 减少内存 | × | √ |
| 提升性能(Culling/LODCalculation) | × | √ |

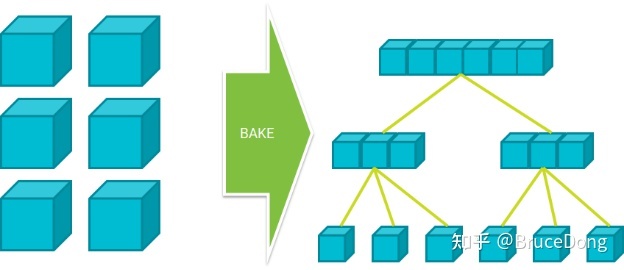
### **当前的HLOD解决方案**

* **Amir Ebrahimi’s AutoLOD ( unity )**○ [https://blogs.unity3d.com/kr/2018/01/12/unity-labs-autolod-experimenting-with-automatic-performance-improvements/](https://link.zhihu.com/?target=https://blogs.unity3d.com/kr/2018/01/12/unity-labs-autolod-experimenting-with-automatic-performance-improvements/" \t "https://zhuanlan.zhihu.com/p/_blank)  
  目前停留在了实验阶段的一个解决方案，后面有空会翻译一下这篇文章。
* **Megacity ECS Demo ( unity )**  
  object和HLOD都做成预制体，嵌套预制体、异步加载场景。优点是它可以管理到每一个object，但是一旦某一个object有修改，必须要重新构建所有与之相关的预制（也就是说需要重新制作）。
* **Unreal 4 HLOD**  
  ○ [http://api.unrealengine.com/INT/Engine/HLOD/index.html](https://link.zhihu.com/?target=http://api.unrealengine.com/INT/Engine/HLOD/index.html" \t "https://zhuanlan.zhihu.com/p/_blank)

### **它是如何工作的**

一、创建一个HLOD树

下图展示HLOD的工作原理

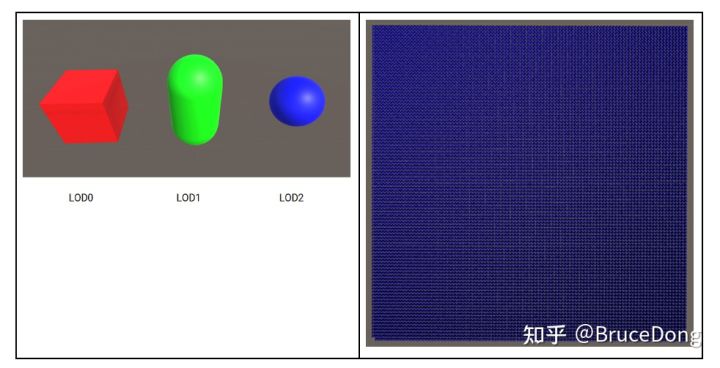


1、通过四叉树将空间切割成N个层级

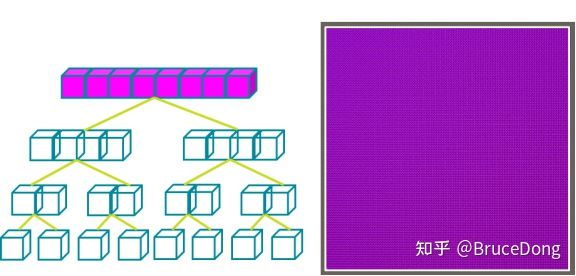
2、叶子节点对应着原始的网格，随着等级的提升网格是逐步被合并的。

二、HLOD生成实例

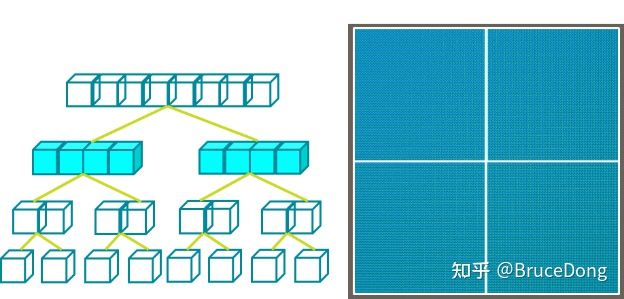
假设我们现在有100\*100个object，如下图所示。



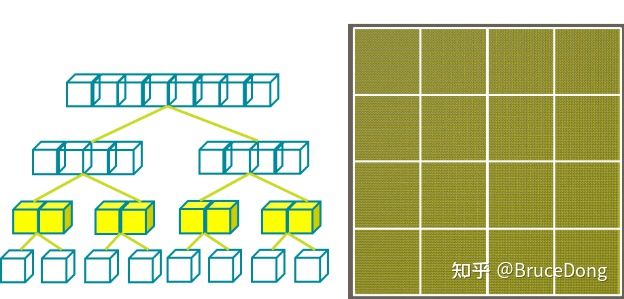
1、HLOD根节点的mesh就会和下图类似。PS：左图画的有点问题，不是二叉树分割。



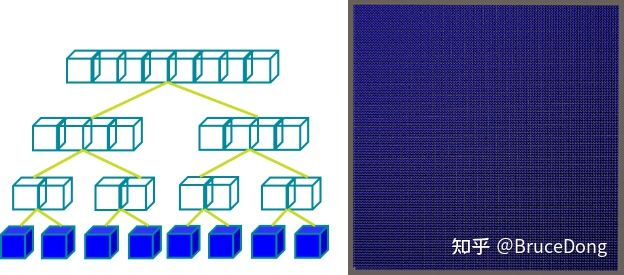
2、使用四叉树划分空间，根据划分的空间合并网格，合并在一起的网格为一个节点。



3、四叉树算法继续向下划分



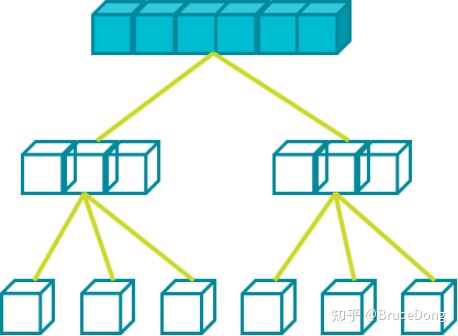
4、叶子节点即是最原始的网格。



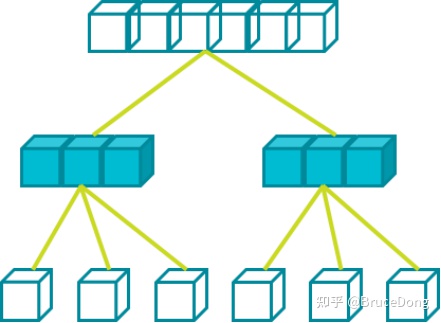
### **使用HLOD树**

我们完成上述的HLOD树形结构的划分之后，节点的显示主要依赖于节点和相机的距离。

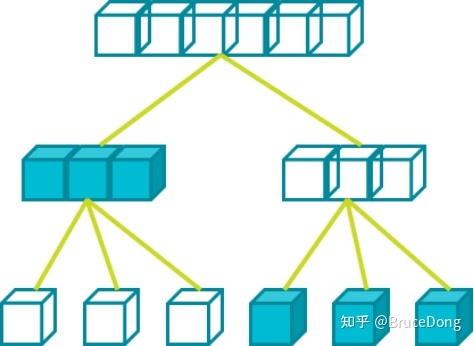
* 下图展示的是当相机远离节点的时候HLOD树的显示。显示HLOD根节点的合并过的网格，同时隐藏孩子节点。



* 下图展示的是当相机靠近HLOD树的时候，根节点被隐藏，显示第二层级的合并节点。

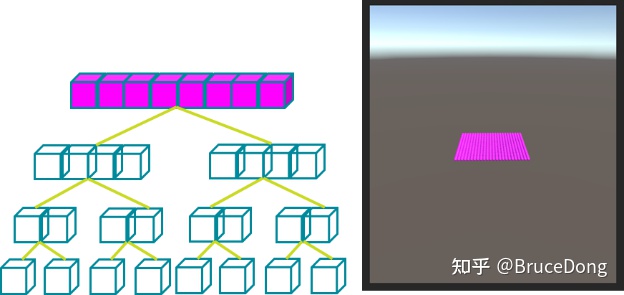


* 下图展示的是当相机更加靠近右侧叶子节点时，显示叶子节点，隐藏叶子节点的父节点。

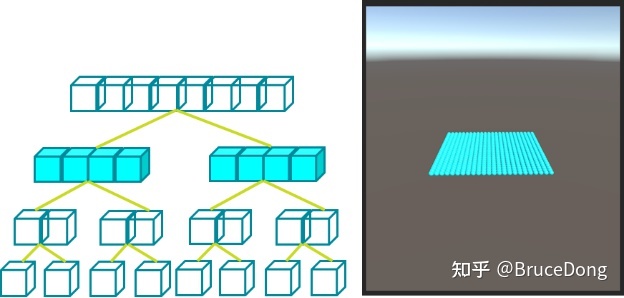


### **HLOD运行时举例**

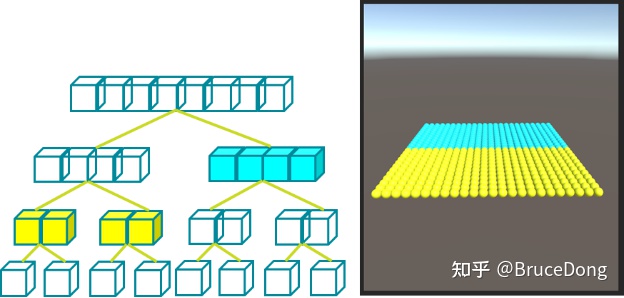
1、如果相机距离HLOD树足够远，显示所有网格合并在一起的根节点。



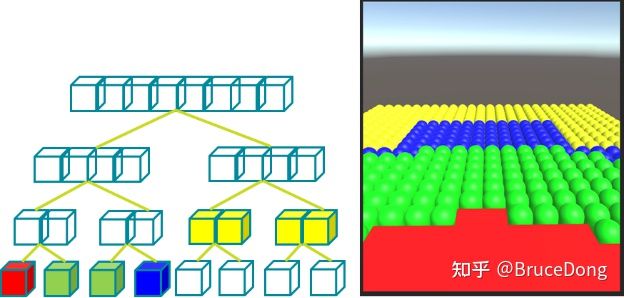
2、当相机靠近时，隐藏根节点，显示第二层级的网格节点。



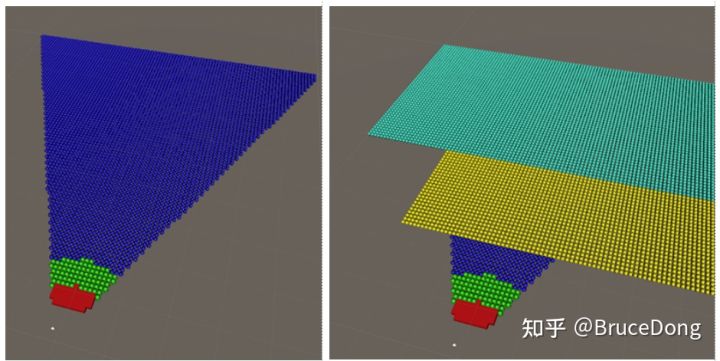
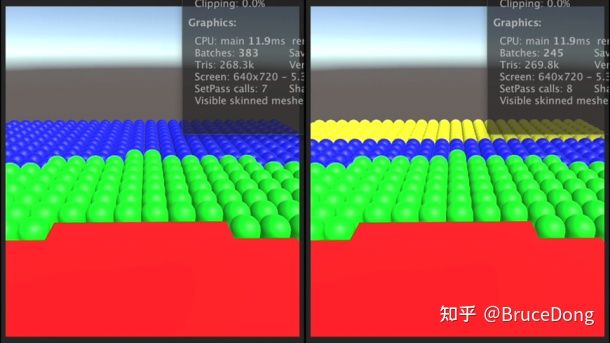
3、随着相机的越来越近，会隐藏当前节点，继而显示父节点。



4、当相机足够近的时候，显示原始的网格。远处的节点依然是显示HLOD网格，这时候就能看到原始网格和HLOD网格混用的情况。

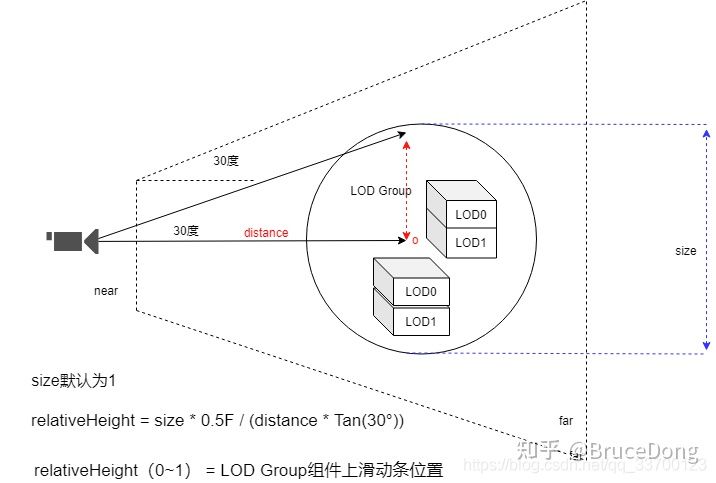


|  |  |
| --- | --- |
| Without HLOD | With HLOD |



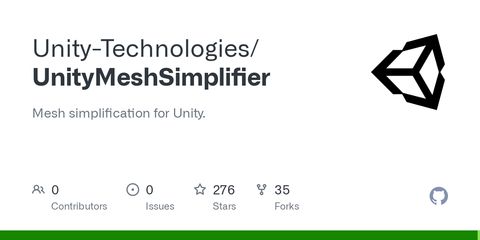
## **计算原理**

HLOD模型是否需要显示计算，根据距离LOD Group的距离、屏幕占比与摄像机FieldOfView计算出relativeHeight，这个数值对应如图所示的摄像机位置，如果这个数值不指向最精细模型，那么就显示合批模型。size取物体在世界坐标下所有物体加起来包围盒大小。

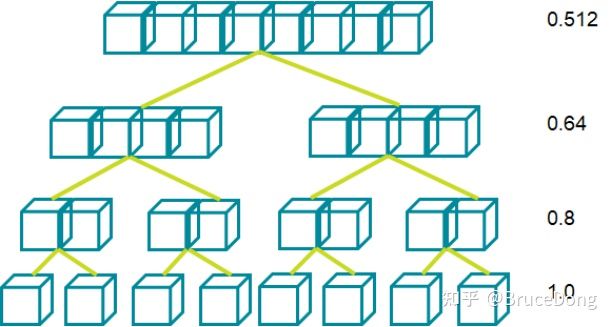


## **关于减面**

当创建HLOD网格的时候，会根据等级减面。这个事儿可以用插件来完成，unity提供一个减面插件UnityMeshSimplifier

[GitHub - Unity-Technologies/UnityMeshSimplifier: Mesh simplification for Unity.​github.com/Unity-Technologies/UnityMeshSimplifier](https://link.zhihu.com/?target=https://github.com/Unity-Technologies/UnityMeshSimplifier" \t "https://zhuanlan.zhihu.com/p/_blank)

rate这个参数是会在每一级上累乘的，举一个例子，将rate设置为0.8，实际每层减面的比例如下图所示，随着层级的上升0.8是幂次方的关系。

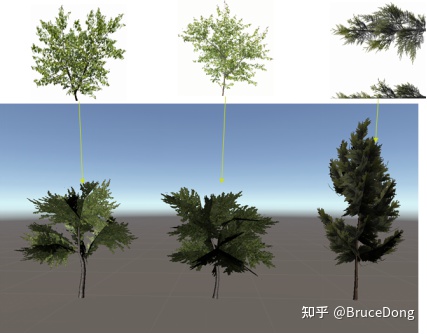
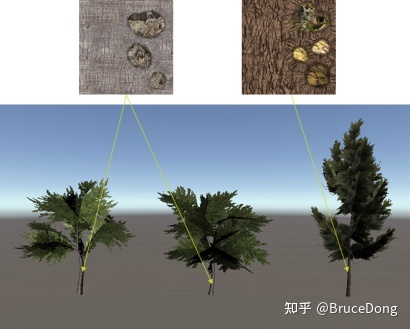


## **合批**

HLODSystem有两个内置的合批工具，如果你想用自己的合批工具，你可以打包加入进去。

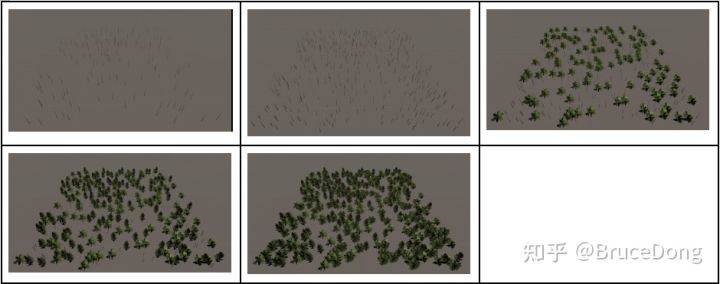
* MaterialPreservingBatcher：将相同材质球的网格进行合批，这种方式使用原始的材质球。
* SimpleBatcher：即使材质球不同，这种方式会合并所有的网格。它会将贴图合并成一张然后创建一个新的材质球。

例子中用到的材质资源有以下五种：



MaterialPreservingBatcher

将相同材质的网格进行合并，下图展示400棵树在这种合批工具下的显示。

相同材质球的网格合成一批

SimpleBatcher

首先使用TextureAtlas工具将贴图合并，然后将每个节点的mesh都合并成一个mesh。贴图合并如下图所示，这样的话400棵树就是一个批次渲染完成。



## **控制器**

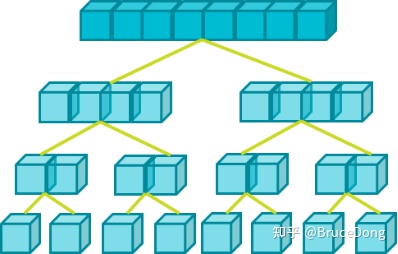
HLOD组件会生成一个流式控制器，这实际控制着HLOD的工作。HLODSystem里面有两个流式类型，分别创建一个控制器控制。

## **不支持**

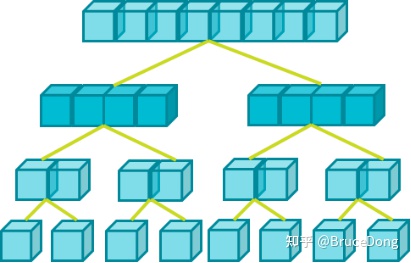
这个流式类型意味着并不支持HLOD树节点的流式。构建的时候会创建一个默认的组件。它会在场景中产生HLOD树节点且禁用它。如果这个HLOD树节点需要展示，那么这个HLOD树节点会被启用，不需要的HLOD节点会被禁用。

下图展示上述的行为。

透明的节点是指已经被加载到内存中，但是并没有展示的节点。不透明的节点表示正在展示的节点。



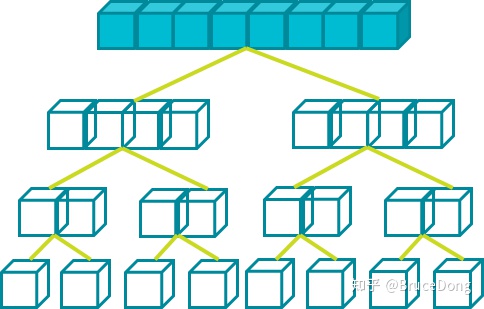
首先根节点是可见的，但是它所有的孩子节点同时也被加到内存中。



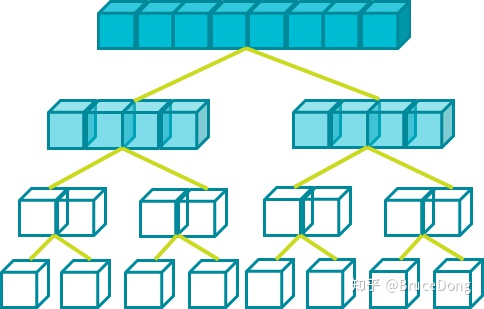
当它的孩子节点可见时，它的根节点不可见消失了，但是数据依然在内存中。

## **AddressableStreaming**

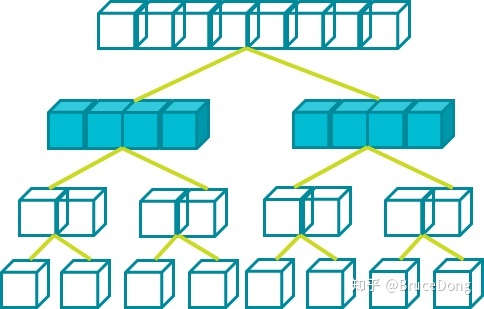
使用Addressables动态流式加载HLOD树节点。当构建的时候会创建一个AddressableStreaming 的组件。HLOD树节点在需要的时候被加载到内存。在加载完成之后，显示这个节点，同时将不需要的节点从内存中移除。



最开始只有根节点可见并且加载到内存中。



当它的孩子节点需要被显示的时候，孩子节点就被加到内存中。

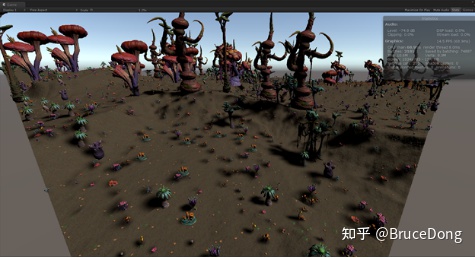
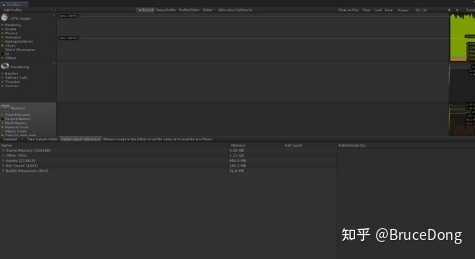
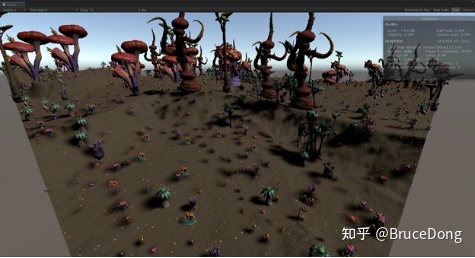
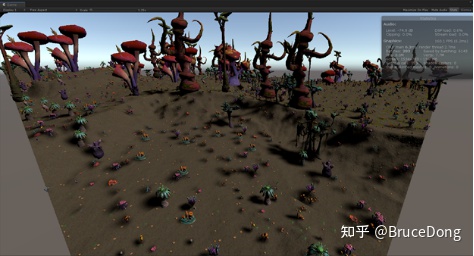
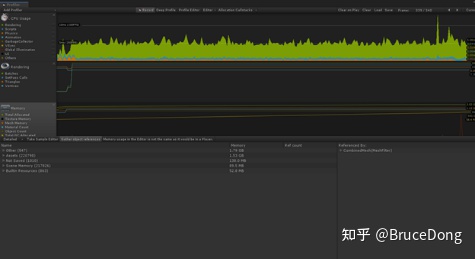


当加载完成时，这个节点就显示出来了，同时不必要的节点会从内存中删除掉。

## **性能测试**

使用了47中预制，共使用了40000次

|  |  |  |  |
| --- | --- | --- | --- |
|  | 静态批处理 | GPU Instancing | HLOD |
| Batches | 2181 | 626 | 283 |
| FPS | 14.5 | 15 | 160 |
| 三角形数 | 10.6M | 10.6M | 8.0M |
| 顶点数 | 8.9M | 8.9M | 7.7M |
| 内存 Asset+Scene | 3.65GB | 536MB | 1.62GB |

静态批处理性能1静态批处理性能2GPU Instancing性能1GPU Instancing性能2HLOD性能1HLOD性能2