1.1 HLOD System简介

首先，HLOD System主要的目标是为了减少Draw Call。然后，进行更多的Batch批处理，从而大大提高渲染性能，减少面数和纹理，这样我们相应地节省了内存，并提升了加载时间。

HLOD System只针对当前所在的地方进行加载，它会流式加载网格和纹理，在后台进行异步的操作。

本次HLOD是基于官方AutoLOD代码的扩展和改进制作出来的，链接：https://github.com/Unity-Technologies/AutoLOD，链接是AutoLOD的文章，可以先看看。下面将详细介绍HLOD原理和实现。

1.2 HLOD与LOD对比

HLOD与传统LOD差异对比如表所示。

LOD

HLOD

减面

√

√

减少Batches、纹理数量

×

√

减少内存使用

×

√

CPU性能提升

×

√

磁盘空间

=

+

2.1 系统支持

BVH划分LOD Group

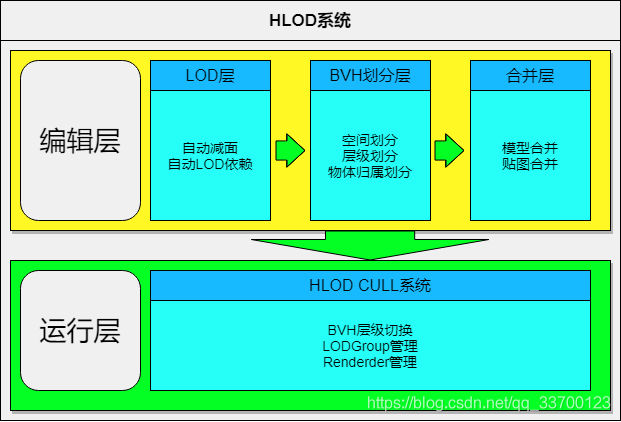
根据BVH划分进行合并模型和贴图

HLOD CULL系统

2.2 系统概述

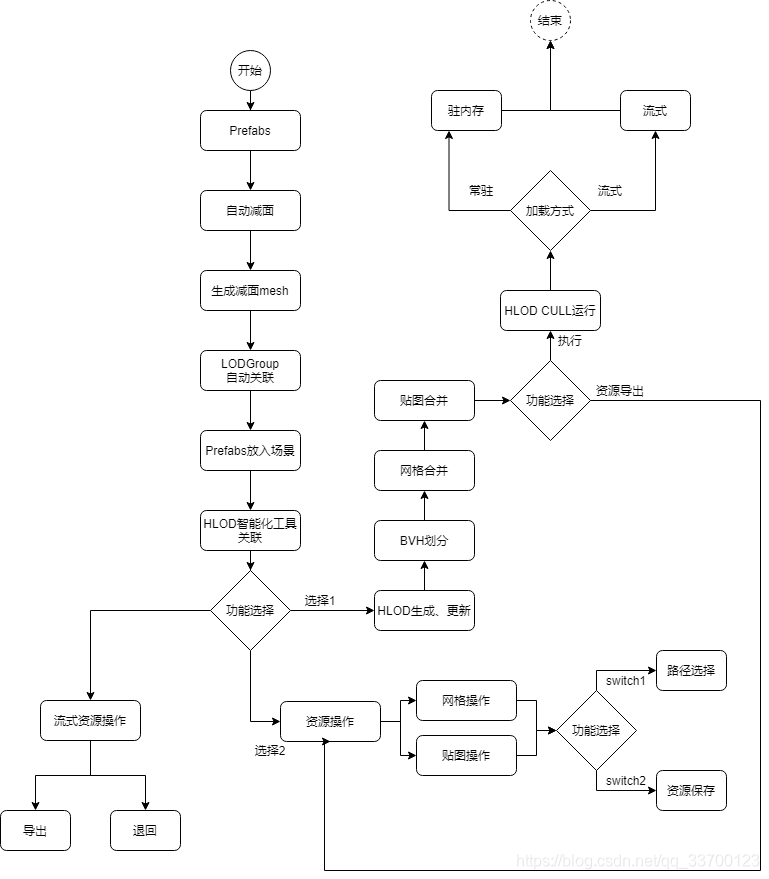
1.系统架构

系统主要由编辑层和运行层组成，编辑层负责每个预制体的LODGroup生成、BVH划分、网格、贴图合并，同时自动做好运行层所需要的关联。运行层负责该系统中Renderder、LODGroup管理及BVH层级切换，系统架构如图所示。



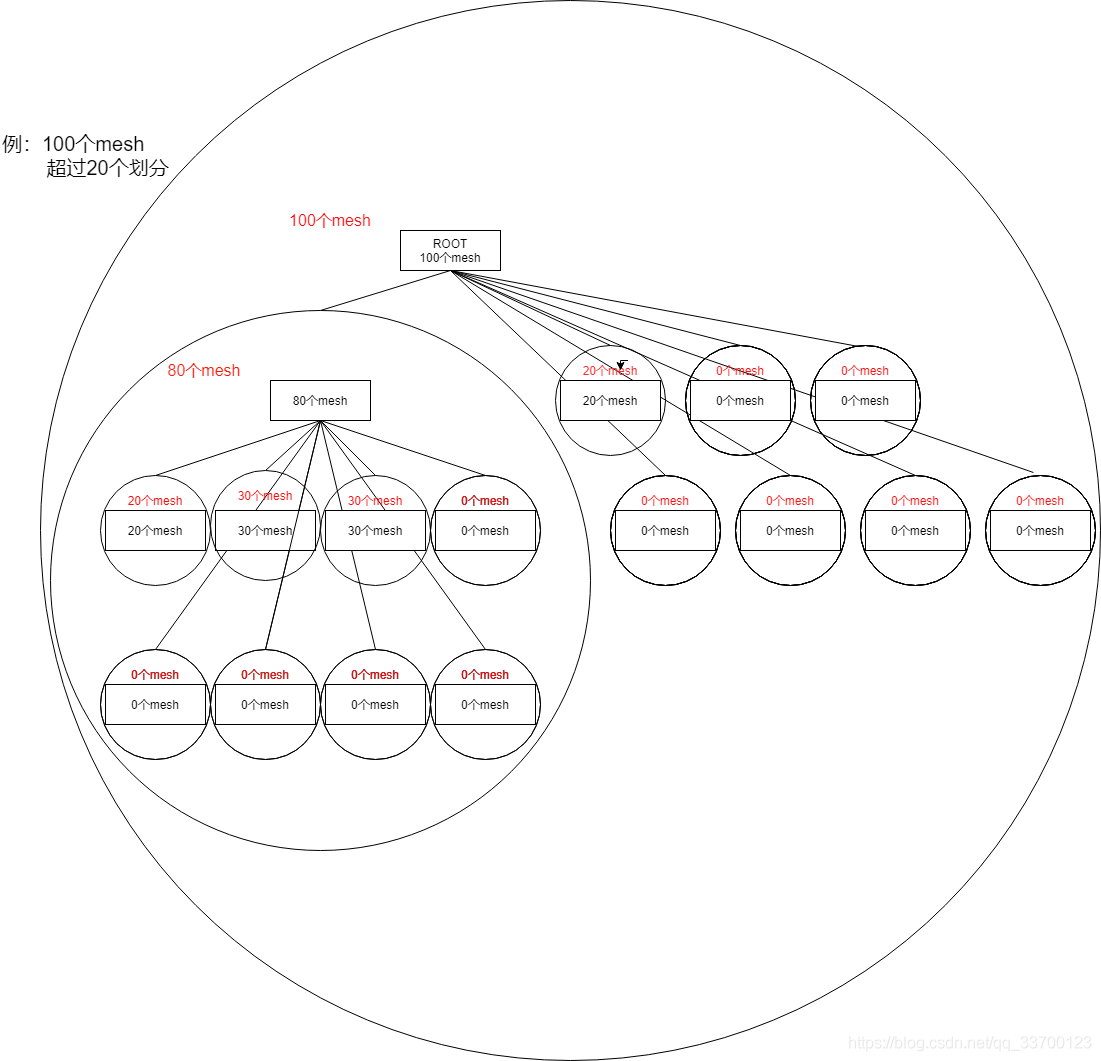
2.系统流程

本套系统拥有一条完整流程，其系统流程如图所示。

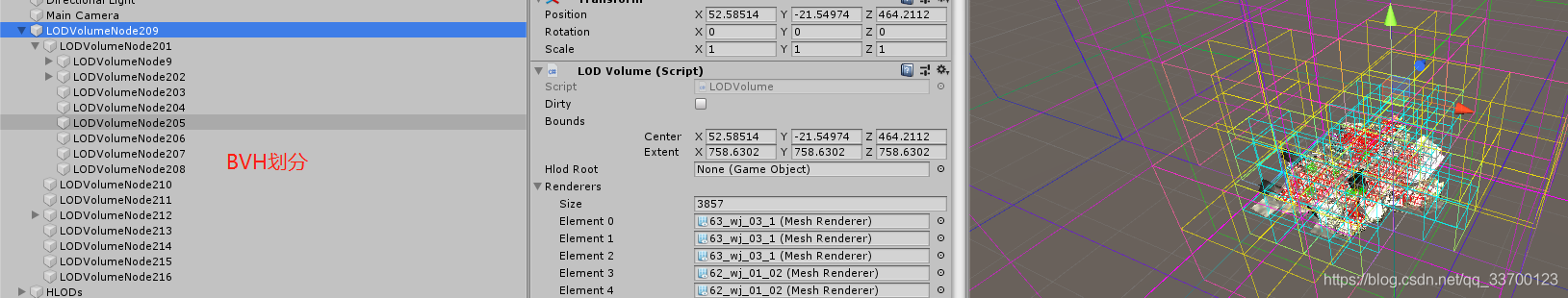


2.3 BVH划分LOD Group

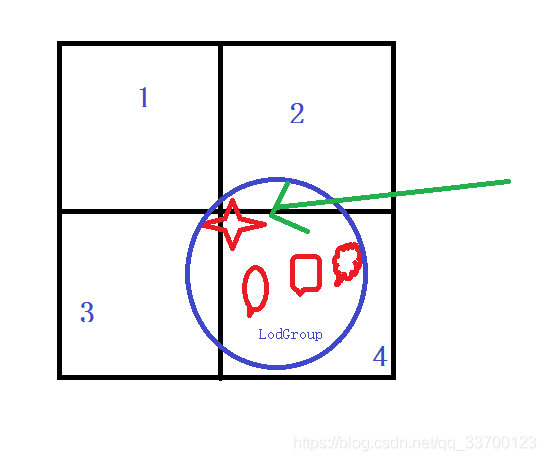
八叉树对LOD Group进行划分到各个区域，划分条件由每个区域超过n个mesh开始划分，划分依据由LOD Group中心点作为划分点，可设置剔除实际包围盒超过指定大小的mesh，划分规则如图2-4所示。



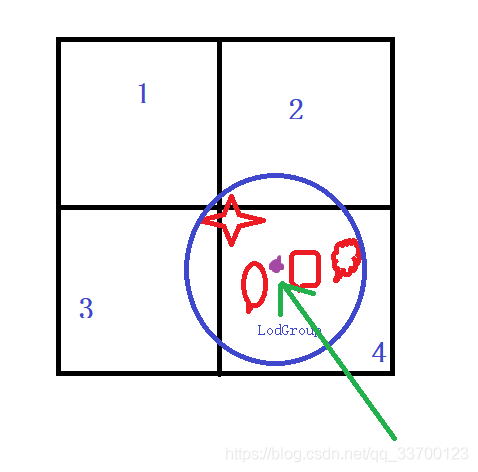
划分后效果如图所示。



这里划分方式对AutoLOD进行了改进，AutoLOD划分方式如下图所示，下图是BVH划分的同一级别中其中的4个区域，圈内是一组LodGroup，AutoLOD在进行BVH划分规则是只要该组LodGroup有任何模型与区域接触，那么该组LodGroup就会被算入该区域，图中4角星与2、3、4区域同时有相交，因此在模型合并的时候这3个区域都会将该组LodGroup下的模型合并。假设HLOD切换到了该层级且同时显示2、3、4节点的合并模型，那么这个LodGroup合并的模型就会被显示了3份，这样的效果是不允许的，解决办法就是同一个层级每个区域不能出现相同的LodGroup。



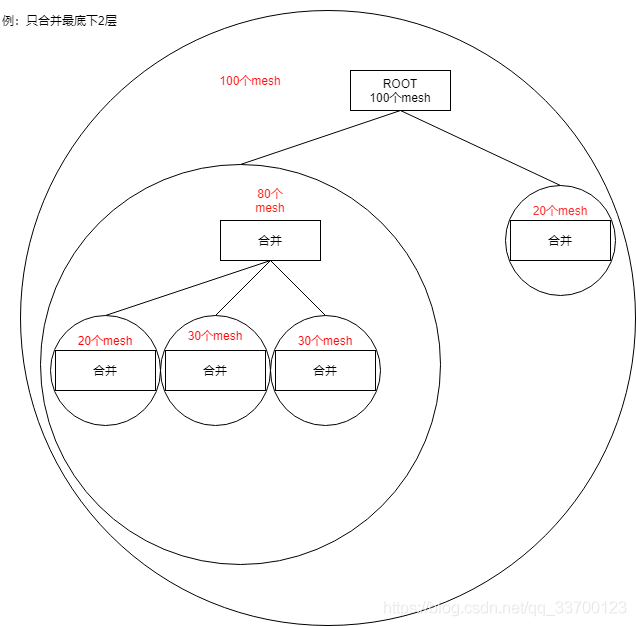
本次HLOD采用的解决办法是使用LodGroup的中心点进行划分，这样就可以保证一个LodGroup最多能被一个区域包含，如下图所示，箭头指向的点就是LodGroup的中心点，它只有4这个区域包含。



2.4 根据BVH划分进行合并模型和贴图

1.合并原理

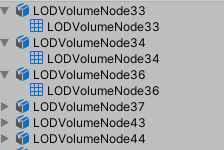
根据2.3的划分，可以设置合并几层的模型(从最底下开始计算)，如下图所示的为合并2层，其中第一层(最底层)有三个区域合并，第二层有两个区域合并。这里节点比2.3少了，是因为没用的节点会被剔除掉，如果这里设置只合并一层也就是最底层，那么上面两层也会被剔除掉。



2.合并的网格

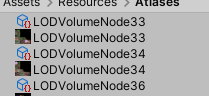
网格每生成一层就会多一倍以上磁盘大小，如果重复的模型多了，那么合并后的网格磁盘大小将会成倍增加，合并后的网格如下图所示。

例如：（300\*300M场景），原始网格6M磁盘空间，合并原始网格两层后多出20M空间(fbx)。



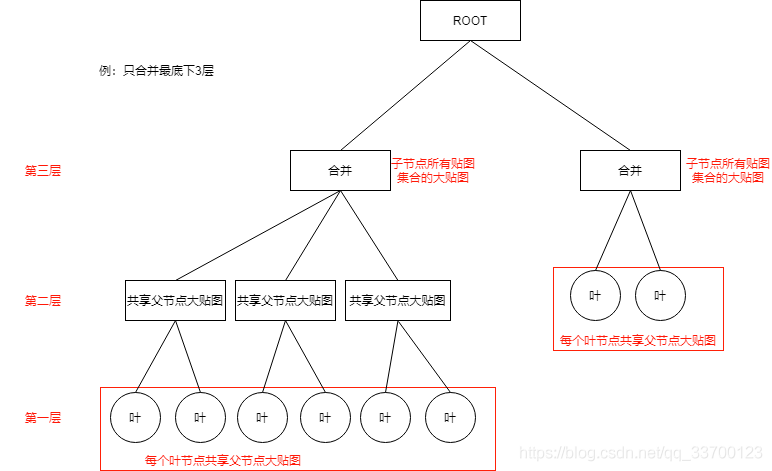
3.合并的贴图

如下图，贴图目前只保留了MainTex贴图，默认使用Standard物理光照shader（带阴影），支持GPU Instancing。



贴图合并规则如下图所示，设置合并层次，比如图中设置3层，那么第三层是所有子节点合集的大贴图（不重复）。

例如：300\*300M场景，原始贴图大小26M，合并原始网格两层后多出50M，多出这么多主要是因为把整个场景合并，原始贴图很多是共用的，导致合并后内存上升问题，所以合并时选择模型和贴图复用性低的模型合并比较好。

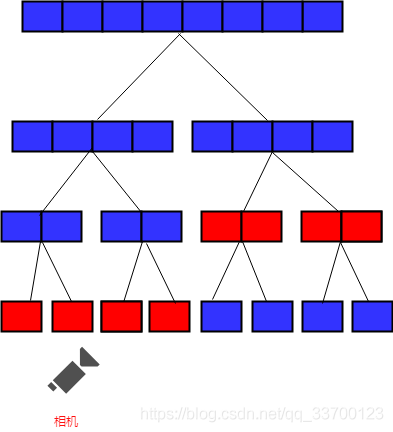


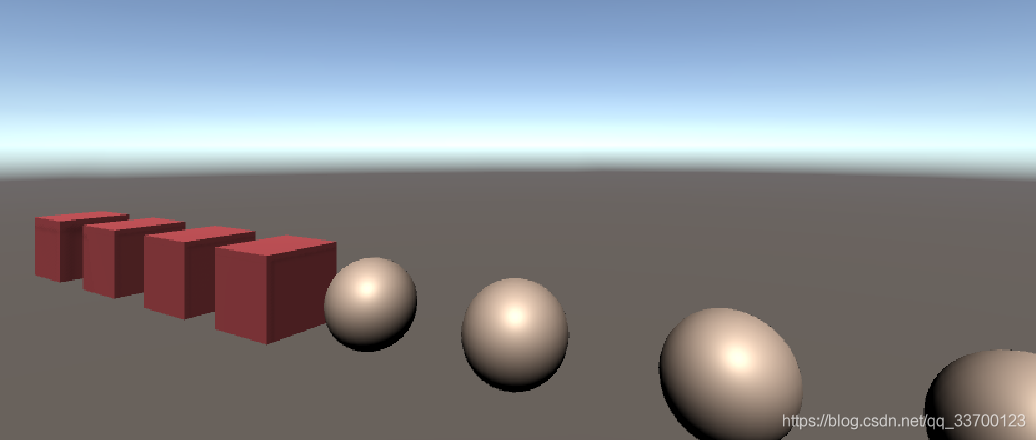
2.6 HLOD CULL系统

1.如何工作

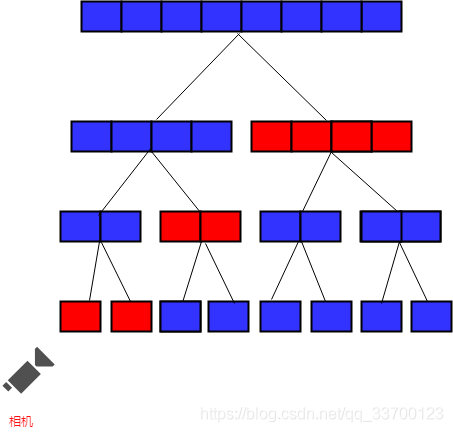
当上述步骤做好后，在BVH的根节点上会有个HLOD CULL脚本，用于控制当前管理的HLOD的切换。

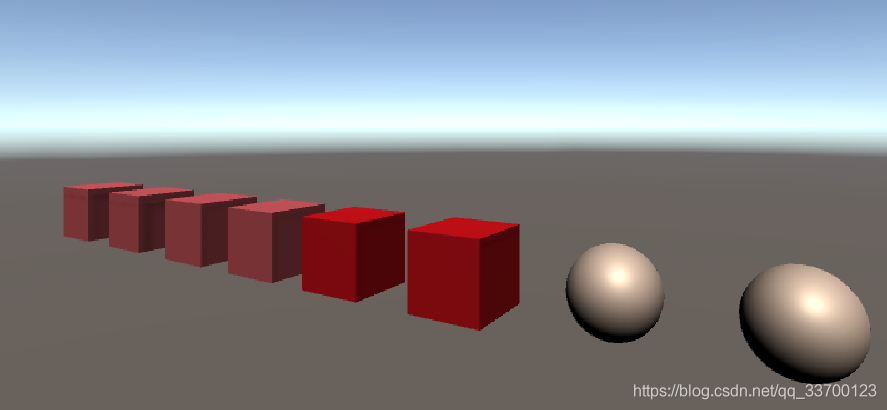
当摄像机靠近部分精细模型时，HLOD切换状态如图2-10所示（红色为当前显示的层级，蓝色为不显示层级）。



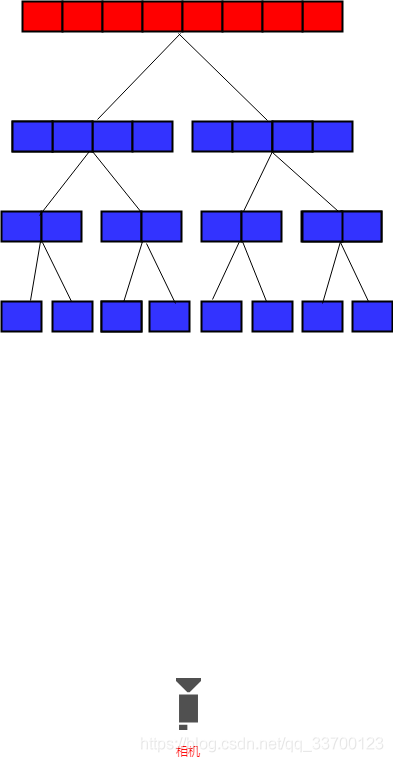


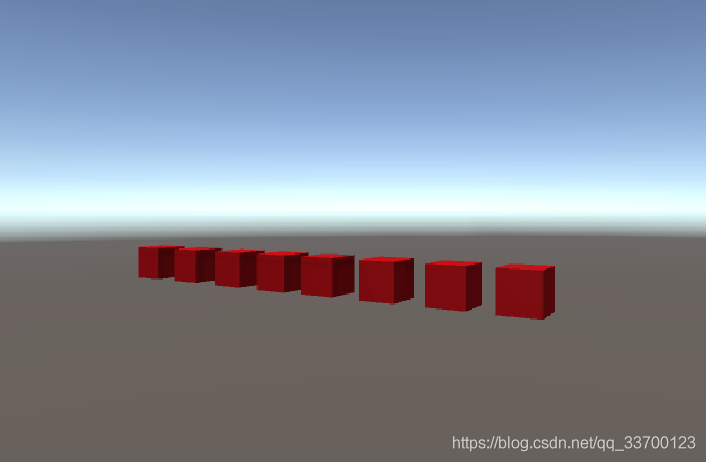
当摄像机靠近少部分精细模型时，HLOD切换状态如下图所示。





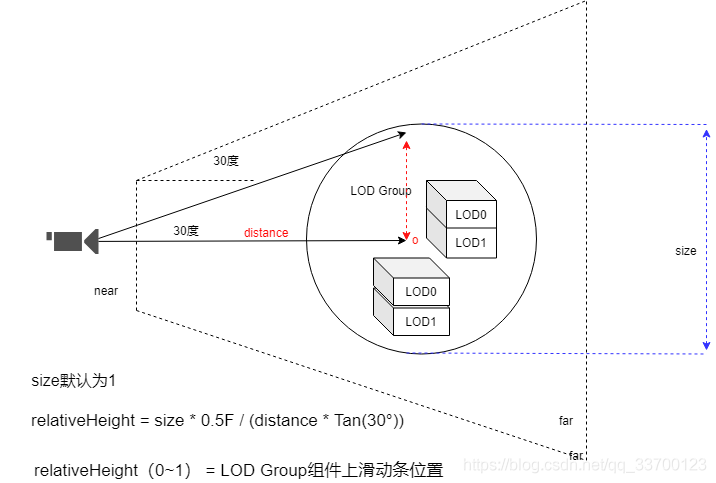
当摄像机距离精细模型比较远时，HLOD切换状态如图2-12所示。



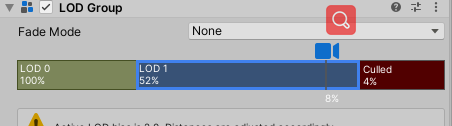


2.计算原理

首先是精细度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2-14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叫起来包围盒大小。

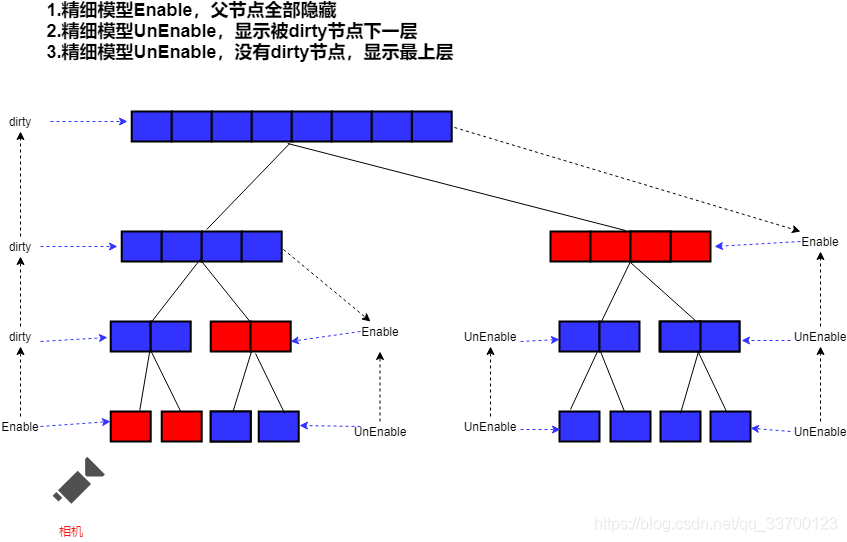


relativeHeight表示



3.工作原理

如图2-15所示，LODGroup的计算只会计算最精细的模型，只要有一个精细模型被激活那么该节点的精细模型都会被激活，父节点的所有HLOD被dirty并隐藏。如果精细模型不激活，那么直到找到父节点被dirty或已经是最顶层情况激活当前层HLOD。

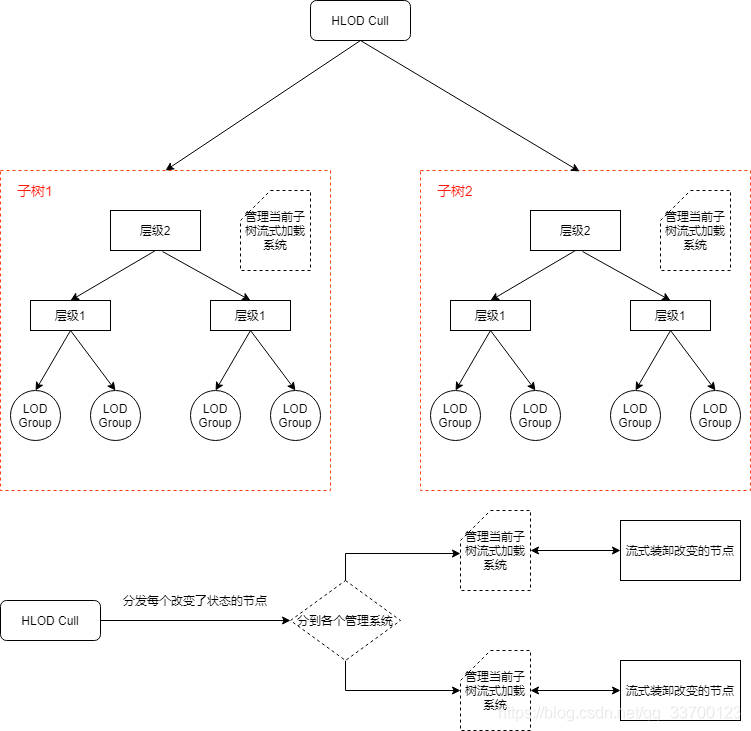


HLOD流式加载

3.1 流式系统设计

流式加载的设计主要针对移动端内存占用过高问题，利用流式加载可以做到极大降低移动端运行常驻的内存。设计如图3-1所示。

首先，一个HLOD System里面有多颗子树，每颗子树都会带有一个流式管理器，该管理器负责当前子树的所有节点流式加载，而HLOD Cull系统负责通知每颗子树哪些节点状态出现了变动。



3.2 流式资源加载设计

如下图，流式加载有两种模式，经过大量测试，总结出了各自优缺点。

1.装完再卸载

当前子树下，所有需要加载的节点加载完毕后再卸载需要卸载的。

优点：可以保证模型常在视区

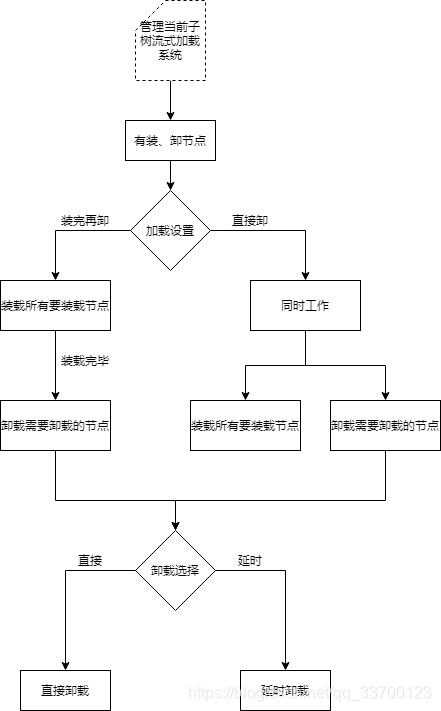
缺点：经常会出现内存峰值，经常会卡帧

2.直接卸

当前子树下，卸载不等待其他节点加载完就卸载

优点：极大避免卡帧问题，少许出现内存峰值问题。

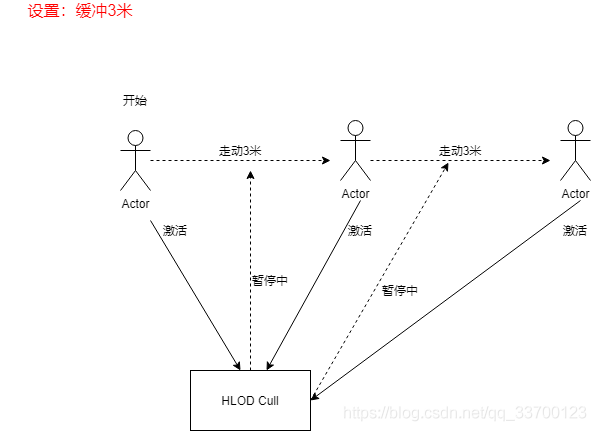
缺点：不可保证模型常在视区，加载的模型内存大可能会出现闪烁现象。



3.3 流式加载距离缓冲设计

经常会出现玩家在加载边沿处来回走动，这会造成资源不断的来回装卸，因此加入距离缓冲策列。

设定一定距离的缓冲，当触发流式切换后，要再次激活流式切换需要走出设定的缓冲距离才会切换，设计如图3-3所示。



四、适用

HLOD Stream应用场景：

1.大城镇，很多房屋需要处理很多Bathces的情况

2.需要看得远，远处看得见轮科且数量较多的情况使用

3.物件密集并且无法使用GPU Instancing的地方使用

4.只要有很多Batches的地方而无法优化掉的都可以考虑使用

五、问题

1.贴图合并只保留MainTex贴图，默认使用Standard物理光照shader（带阴影），支持GPU Instancing。

2.相同的预制体的网格合并时内存会翻倍（这个跟静、动态合批一样）

3.每生成一层HLOD所需要的网格内存会多一倍以上

4.不同子树相同贴图会出现重复贴图合并现象。

5.没有实现按照材质球合并

————————————————

版权声明：本文为CSDN博主「IChessChess」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/qq\_33700123/article/details/113835834