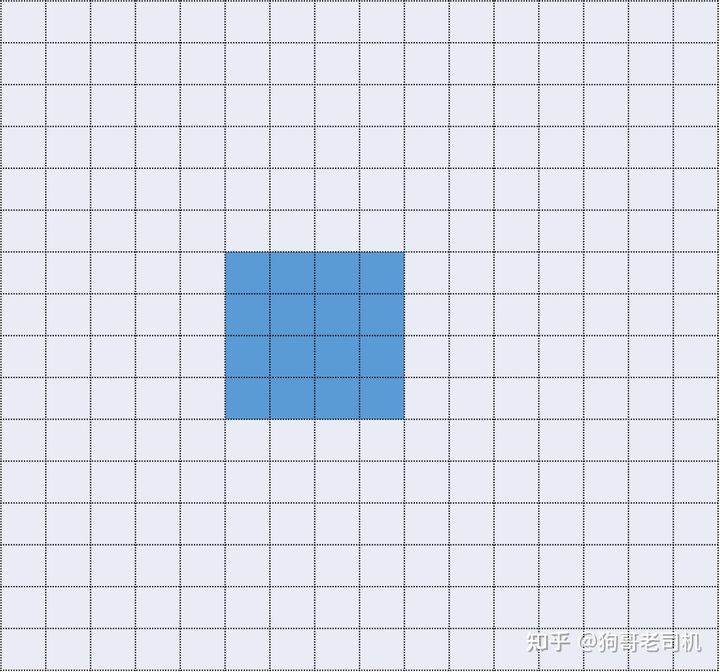
大世界最重要的毫无疑问是地形了，地形也是一项比较古老，且一直在迭代更新的图形学技术。地形系统主体技术要点，一般围绕着LOD来展开。最近一些年，随着DrawInstance和GPU Pipeline的流行，地形系统又在这两个方向做了进一步发展，这俩技术非常契合地形系统，简直就是为地形而生。

Unity的整套地形系统（包括植被）只能说是一坨浓shit，在有DrawInstance功能前，几乎不能在移动上使用，大家一般采用转成Mesh的方式在游戏中使用。转成Mesh始终不是长久之策，发挥不了地形极致LOD的优势。要想做大世界，自己写一套高效的地形系统是必不可少的。

大概思路依然是合理的材质和模型LOD，结合DrawInstance，在性能和效果之间进行平衡。我这里提供一个在移动平台验证过的可行解决方案，可以参考，也可完全按照这个方案来，至少在几万到几十万的量级不会出问题。

通过模型，可以分两部分，GPU地形和远景四叉树，以下图每个格子是512的大地图为例，灰色部分为远景四叉树，蓝色部分为GPU地形。



## **GPU地形**

GPU地形的实现，我之前写过一篇博客，可以参考一下

[狗哥老司机：Unity实现GPUDriven地形182 赞同 · 16 评论文章](https://zhuanlan.zhihu.com/p/352850047" \t "https://zhuanlan.zhihu.com/p/_blank)

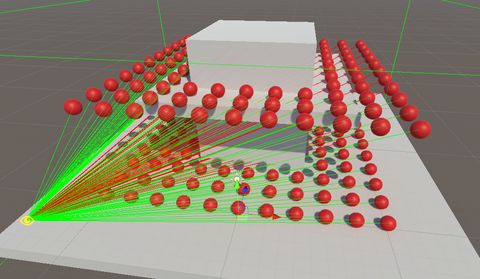
这里主要交待一下配置和注意事项。

GPU地形整体大小是2048\*2048，更新粒度是512，也可以是256，但是256的话，资源文件可能就太多了，Unity在资源文件爆炸后，import的速度会变得很慢。实际测下来，512的粒度没有明显问题。LOD分5级就足够了，因为我们整体大小才2048，LOD0的Mesh对应密度比例是1:1，LOD4已经是16\*16了。

跟模型有关的数据建议用保存成二进制文件，因为保存成unity文件，如Asset或者贴图，需要把这些资源放到Asset目录下，二进制的数据文件是可以放到目录外的，打包的时候进入Bundle就ok，在editor模式下，可以直接通过文件访问，一切为了节省Import耗时。二进制还有好处是，可以整合各种数据，如每个块各个LOD等级的一些配置信息（坐标，高度差等），挖洞信息等。一个2Wx2W的大世界数据能控制在500M左右。

原博客使用的是Hiz来剔除，在实际使用过程中，Hiz由于有一帧延迟，不太适合做地形的剔除，最好还是采用PVS，并且地形的排布比较整齐，比较好做PVS，天然省去了PVS里字典映射的部分。

本来想写个用DXR烘焙PVS的教程，法线知乎上已经有了，直接贴上。DXR是个不错的框架，很适合做一些离线自动化工具。

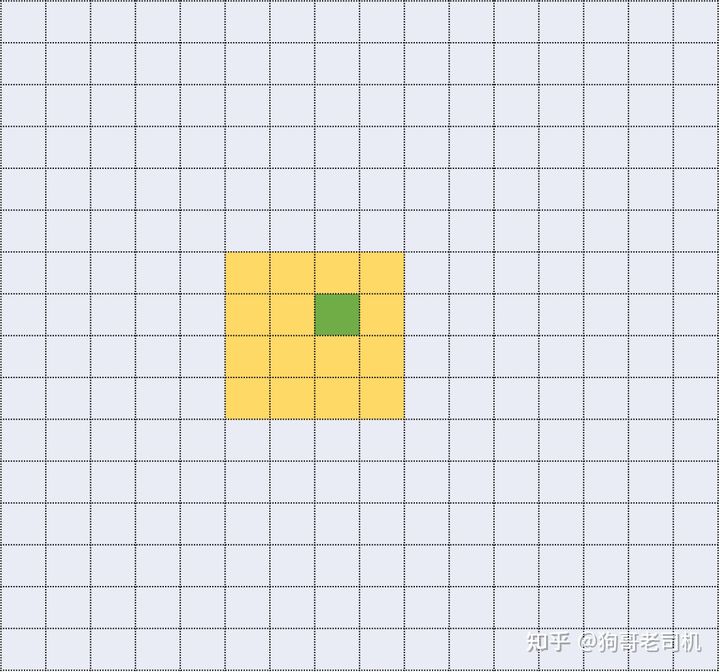
[zhing2006：使用Unity DXR加速PVS烘焙67 赞同 · 7 评论文章](https://zhuanlan.zhihu.com/p/88905817" \t "https://zhuanlan.zhihu.com/p/_blank)

这里，着重说一下，PVS的数据组织和加载使用。

gpu地形只显示2048x2048的地形，那么全量的PVS数据是多少呢？LOD0是一个4x4的Mesh，显示的实际空间是4x4，那么LOD0的内存数据就是(2048/4)\*(2048/4)/1024 = 256KB，所有LOD加起来不超过512KB。数据结构直接用一维数组就ok了，因为是个全量均匀排布，那么Offset = (Pow(4，level) - 1)/3，这里level=0表示最低的即2048x2048，需要有个换算，还有就是可能低等级没有到0，再减去无用Index就得到正确结果了。其实从一个地块的数据量，就能大概推算出整个大世界地形的PVS数据有多少，实际还能进行压缩，这些对于包体的大小是完全能接受的。

硬盘数据我们为了省去索引烦恼需要全量存放，是一颗完整的树，但是这里的内存还能压缩，因为LOD0是最多的，但是离中心点较远的位置其实是不需要的，可以在LOD0只加载64米以内的数据，LOD1则是124米，以此内推。这样是能把数据压缩到非常小。索引的话就需要做进一步换算了，拿个纸笔应该能很快推算出来。

GPU地形可以再拆分两种材质

绿色为RVT材质，黄色为中景离线烘焙贴图材质

**近景RVT材质**

关于RVT的实现，我之前写过博客，这里就不展开了

[狗哥老司机：Unity实现移动平台超大地形RVT202 赞同 · 28 评论文章IMG_259](https://zhuanlan.zhihu.com/p/300731406" \t "https://zhuanlan.zhihu.com/p/_blank)

在这里由于使用GPU地形的原由，粒度会下降到4x4，这样我们可以把尺寸缩到512，rvt在把尺寸缩到512后，即使VT的size控制到2048，也能达到一个比较清晰的画面，不仅能节约很大的内存，而且地表的精细度也能得到很大的提升。

**中景离线烘焙贴图材质**

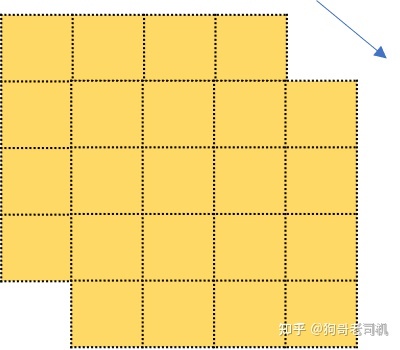
我们为每一块512地形，生成混合完成的Albedo和Normal，这里有三点需要注意。

第一，烘焙的时候注意把投射RVT的模型也渲染到上面，比如路面，贴花这些，直接烘焙到中景贴图上。

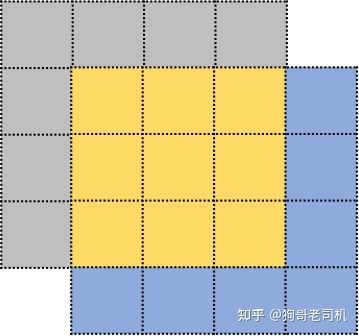
第二，Normal直接取地形的顶点法线即可，无需把layer上tangent转到世界空间，相隔太远，太碎的normal反而会造成噪点，起反作用，我们只保留顶点法线的结果就ok。

第三，烘焙时，地形的Layer可以适当放大，远处的纹理可以通过放大Tile，让纹理的细节显现出来，具体放多大，取决于RVT远处纹理的缩放比。

在材质准备好后，由于是512位单位的，我们需要显示2048\*2048（中间扣掉一块512为RVT材质），我们在运行时，可以采用TextureArray或者动态合并VT的方式。以合并VT的方式为例，我们可以忽略中间RVT那个块，直接生成2048\*2048的完整贴图，直接平铺上去就OK，要处理的是，我们应当尽可能复用之前的数据。如：



我们只需把右下一圈，填到左上即可，如下图，将蓝色区域，填在灰色区域，然后存一个偏移即可。



这种处理方法在整个大世界的资源加载非常常见，一定要有这个意识。  
由于整个GPU地形使用两种材质，那么在GPU地形生成Mesh的IndirectDraw就要分开生成，分两个drawcall提交。

## **远景四叉树模型**

2048以外的地方，其实也可以用GPU地形，只是资源的形式要分两级，2048以外的数据粒度更大一点，保证IO友好，甚至可以做四叉树的IO，不把各个等级的数据放在一起，从技术上讲是能完全可行的，关键问题在于，出来的模型质量很差，因为需要把LOD等级提到很高，基本至少要128\*128用一个数据表示，疏密调节的灵活度在这里是远远不如模型。

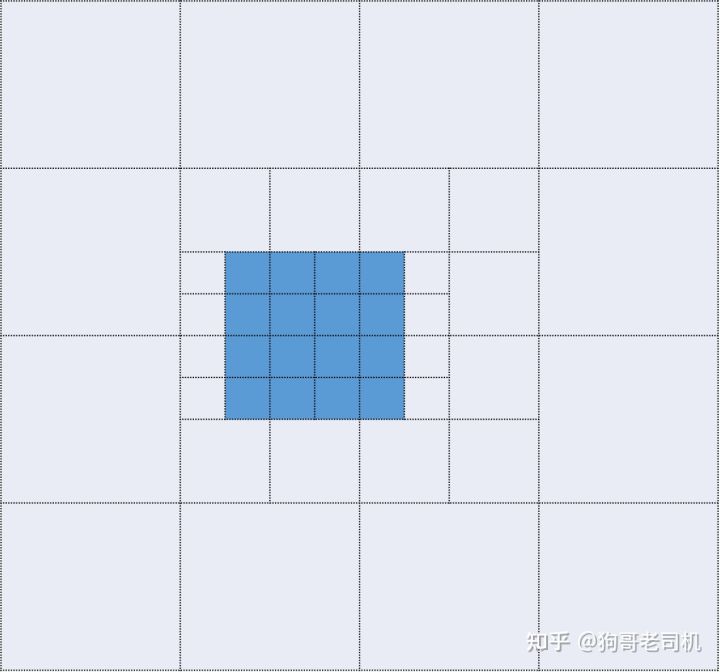
在确定生成远景模型后，最理想的情况是，全地图生成一个静态模型直接渲染，一个2Wx2W的大地图，做完极致优化，1~2W面就能达到预期效果。这里建议用Houdini自动生成，可以把山脊那部分的面加多，海拔低的盆地和平原面数可以砍得狠一点。但是我们面临了几个问题。第一，我们要能灵活抠掉中间2048\*2048的空隙，粒度是512。第二，远景模型跟GPU地形之间存在接缝问题。

第一个问题，难的不是扣出中间部分（vs输出V\_Position为无穷大或者蜕化顶点到边上），难的是2048的矩形边，要求扣出来是一个四四方方的，那么就需要在每个512上都要卡一条直线，卡线会使模型顶点数直接翻倍。

第二个问题，可以把模型往下降降，根据视角特性，能解决大部分问题，但是恰恰地形接缝问题太过突兀（漏出天空盒，有人考虑把天空盒的底面用地形颜色图，依然没法解决所有问题），必须得保证100%没问题。那么经典的解决接缝问题，蜕化边，加裙子，蜕化边问题，由于GPU地形和远景地形不在一个系统，不好蜕化边，只剩加裙子了，加裙子又需要很多顶点。

为了解决这两个问题，最简单的一个办法是，切成每个512\*512的地块，且生成裙边，然后控制地块的显隐，再通过staticbatch渲染。这里引申出了两个新问题，一个是面数过多，切块加裙边，面数基本翻3倍以上，另一个是虽然是staticbatch，但是实际drawcall数偏高。实际裙边仅仅在2048的那个矩形需要，其他地方造成很大的浪费，能缓解第一个问题，但是还是由于切边导致面数翻番。

我们引出一种新的解决办法，四叉树生成各个等级LOD的模型，然后根据显示矩形，如图所示：



这样的好处在于：极大减少了因为切割添加的面，drawcall相比全量lod0少得多。

**一些想法**

还是想通过蜕化横跨中间矩形三角形，把矩形内的点推到矩形上，然后对于矩形内剩下的三角形，在vs的输出，设置SV\_Position的xy为无穷大来挖空中间的部分，似乎是一种比较完美的解决方案。由于四叉树的远景显示，性能没太大问题，暂时没尝试这种办法，后面如果有机会，可以试试这种办法。

远景的材质，直接使用一张Diffuse贴图就ok，但是这张贴图，不要使用工具自动生成，因为就这一张贴图，值得画一些时间在PS里精雕，特别是可以在游戏场景中对着来画，一边画，一边对照游戏中的效果，在山脊的地方，勾勒出山体的线条，把法线，AO这些都直接画到Diffuse上，会起到意想不到的效果。