# **Unity优化技巧（下）**

作者：[韩宇飞](http://gad.qq.com/user/index?id=2044697)

链接：<http://gad.qq.com/article/detail/39888>

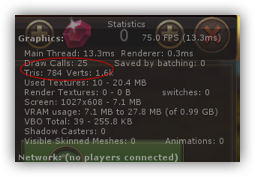
**本文首发于知乎专栏：**[MACK的游戏开发笔记](https://www.zhihu.com/people/han-yu-25-35)**，欢迎各位关注。**

最近比较慢最后一篇写的有些晚了。最后介绍一下GPU，内存，闪存耗电和网络。因为介绍的比较广，每个点都比较简单，后续可能会针对某些部分做更深的介绍。

**GPU**

* **DC**

DrawCall实际上优化的CPU的时间，但因为DC的优化一般都是材质mesh合并，所以放到了GPU的部分。每次在准备数据并通知GPU渲染的过程称为一次Draw  
Call。渲染一次拥有一个网格并携带一种材质的物体便会使用一次Draw Call。可以理解为调用一次DC就换一种画笔在画板上画一个物体。



* **多线程**

限于篇幅，虽然多线程是CPU的部分但都放在GPU这部分介绍。随着PC的主频达到瓶颈，手机也开始朝多核并行开发的方向前进了。所幸Unity这方面做的比较好，大部分开发者可以不用考虑多线程的开发方式。

* + 多线程渲染：虽然叫多线程渲染其实节约的也是CPU时间。虽然CPU和GPU是并行工作的，但是因为CPU提交了渲染数据和指令之后，需要等显卡同步会阻塞逻辑的执行。多线程渲染就是把这个等待时间和逻辑的执行并行了起来。Unity5.5版本在PC和IOS上都默认开启了多线程渲染，但在Android设备上单独提供了多线程选项，这是因为Android系统设备太多碎片化严重，开启多线程会导致一些设备出错闪退等。亲测我们游戏开启多线程渲染后，能提高10%的性能，所以我们是默认开启的。王者荣耀的解决方案比较完善，他们会针对每个机型做大量测试，还和硬件厂商有合作，只针对测试通过的设备开启多线程。所以也会看到一些比较有趣的广告，xx手机对王者荣耀做了优化支持多核，其实只是一个配置参数而已。
  + GPU骨骼更新：利用GPU计算骨骼降低CPU的开销，但是Android的一些设备因为GPU非常弱（广告只介绍CPU），所以开启之后很多设备性能反而下降了，这个配置默认关闭。
  + 网络多线程：我们是开了一个线程进行网络协议的收发处理，这也是比较常见的。
  + 多线程并行计算：Unity5.6的位置更新其实就是开了线程计算，所以新版本的Unity设置位置和旋转的性能大幅提升。此外音频的计算相对独立也都是在其他线程里计算的。优化时也可以考虑把一些运算量比较大又相对独立的部分放到线程里计算，例如AI。
* **面数**

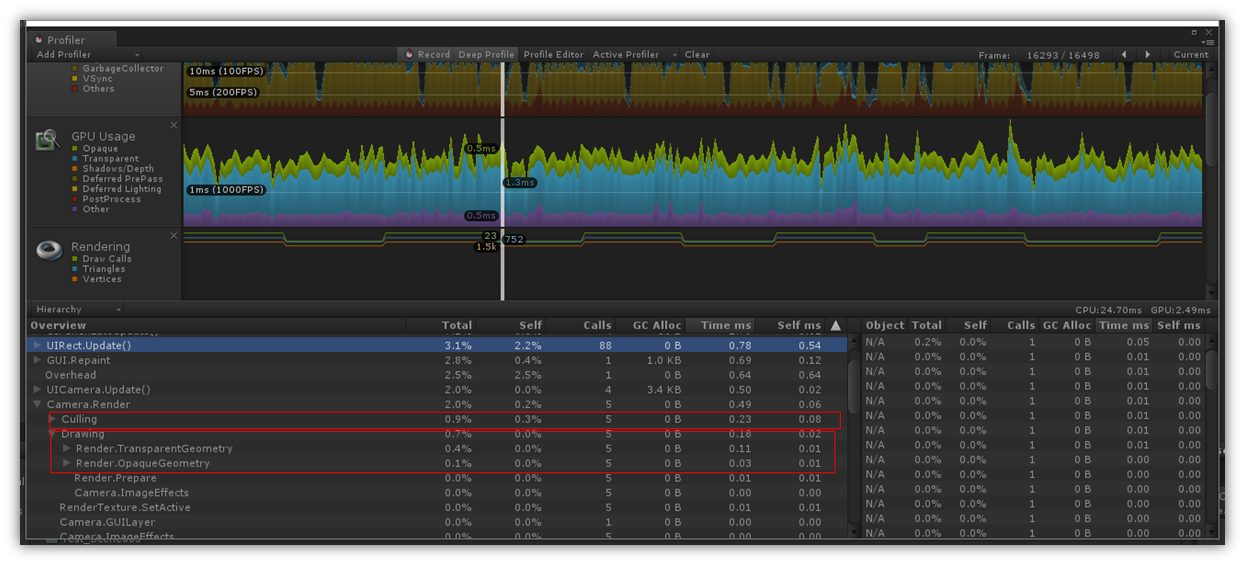
DC<200,面数小于10w是Unity建议。我在14年的测试，在红米1上，当dc超过100性能开始直线下降，面数超过6w面性能开始直线下降。

我们目前场景的标准是整个场景少于10个DC，由美术或插件合并输出，这样虽然不是十分灵活，但因为静态合并会增加Loading时间和内存，动态合并也会增加内存和合并CPU开销，所以还是采取这种优化方式。角色2000面左右，低配1个DC，高配3个DC（描边，实时阴影，主角还有额外的一个因为墙后半透效果）

* **LOD**

对GPU的优化也可以通过LOD进行，可以通过模型LOD，骨骼LOD，粒子LOD，材质LOD的方式，地形LOD等等，例如不同配置开启不同的效果，开启后处理等。

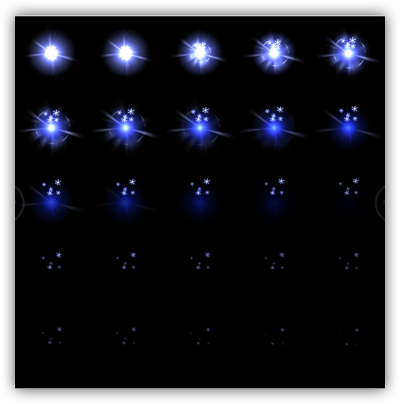
* **遮挡**
  + 遮挡剔除：顾名思义就是被遮挡看不见的地方不渲染，例如墙后的物体。遮挡剔除可以CPU计算也可以GPU计算。Unity自带了OcclusiongCulling，但5.x不建议使用。
  + UI遮挡：例如全屏UI可以隐藏背景，节省电量。
  + 场景拆分：我们因为是俯视角，能遮挡的东西很少，只是采取了场景分割，小物件LOD的方式。
  + 入口：一般用于室内，早起的Quick使用了二叉树和入口的优化方式。
* **半透明**

相对不透明半透明开销巨大，在PC和手游上都是，还会破化渲染管线的优化。另外使用alpha通道的贴图压缩也很困难（特别是IOS上的PVR格式Alpha像素压缩之后损失巨大，ETC，DDS，PVR等格式Alpha一个通道的压缩比就等于其他3个通道了）。半透明物体不写Z无法做像素级遮挡，需要做混合操作，半透明需要单独排序。以下有一些比较通用的优化点；

* + 少用/减小面积：尽量少用，要用尽量减少占用屏幕的面积，减少像素填充率。
  + 这里要说的是PowerVR的平台AlphaTest比AlphaBlend开销更高。因为会影响HSR（Hidden Surface Removal）优化,类似EarlyZ，和其他平台不同。
* **Culling**

这个其实也是优化的CPU。每个相机都会针对场景图做Culling，通过视锥和包围盒剔除这个摄像机看不见的物体，减少实际渲染物体个数。因为Culling操作比较耗时，也可以通过减少摄像机下对象的个数或者手工开关分层来做优化。这里需要注意的是，合并方式也会影响Culling，例如把整个游戏所有的树的都合并成一个DC，DC是下降了，但是只要有一棵树在摄像机里，所有合并的树模型都会被渲染，增大了渲染的带宽和负载需要权衡使用。例如手游穿越火线就是把整个场景优化成了3个DC，一个DC渲染所有地形，一个渲染所有的墙，一个渲染所有的箱子等。

* + **粒子**

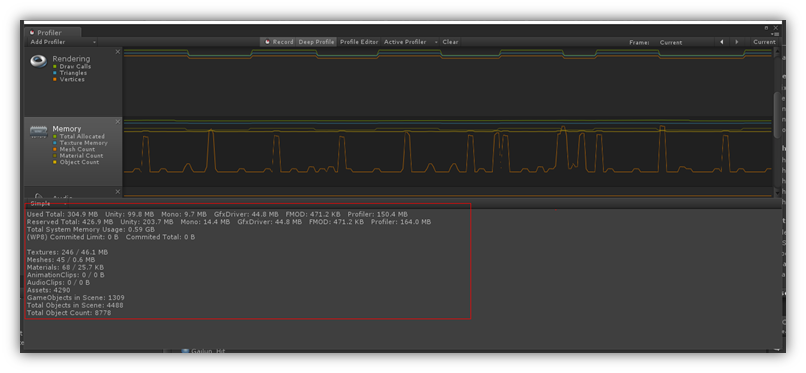


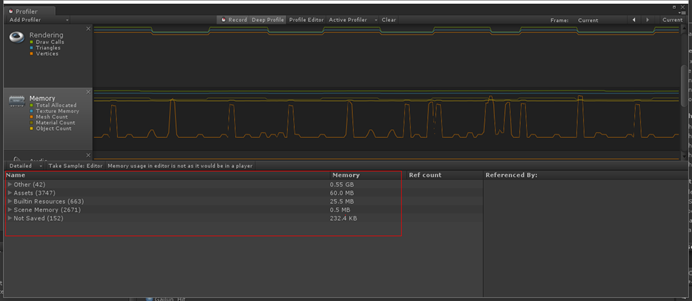
* + 减小屏幕覆盖面积
  + 避免使用Alpha
  + 合并材质和mesh
  + LOD：根据机型或者距离降低粒子发射器的个数和效果
  + 序列帧：对一些俯视角游戏使用序列帧做特效也能大幅提升效率。这个是一个典型的空间换时间的优化，会增大内存，使用受视角影响如果压缩较大会损失一些效果，要有选择的使用。之前在做一款3d战斗卡牌游戏时特地写了序列帧录制工具。
* **其他**



* 渲染设置：阴影，雾，抗拒齿，垂直同步，各项异性，多线程渲染，GPU计算骨架，顶点受骨骼影响，软粒子等等。每个项目要求不同。
  + 降低渲染的分辨率：缩小Framebuff分辨率，减少ps开销和内存显存，但是会模糊，王者荣耀等很多主流游戏在Android上都降了分辨率。
  + 智能动态调节：根据玩家配置和游戏环境实时调整配置，低配设备或者战斗降低配置进行限帧，高配插电或者低开销场景，动态提升配置，提高限帧。有一篇文档有更详细的介绍。
  + 后处理：Unity里可以通过看Graphics.Blit性能了解后处理的开销。后处理一般是像素级别的计算，手机设备上分辨率有普遍比pc高，使用的时候更需要注意。
  + 不使用多维子材质材质：我在真机上实测，三个子材质，50个物体，使用多维自材质12帧，330个dc，343面。拆开之后20帧，154dc，134面。多维子材质Unity无法动态合并。

**内存**

****



其实在手机上内存的优化才是最重要的，绝大部分闪退都是内存不足导致，都闪退了其他优化还有什么用呢？内存分析工具有很多，使用Unity的Profile，memoryprofile，XCode，，UWA。一般简单快速分析用Profile，具体内存资源使用memoryprofile和uwa，内存泄露代码级内存优化使用xcode。

* + Mono托管堆：逻辑代码的堆内存分配 ，一旦分配，不会返还给系统。代码级别比较难查。以前在PC上自己写过通过钩子的方式记录内存分配的工具，Unity推荐编译成IL2cpp使用XCode的工具分析，也可以自定义采样点逐帧查看或者通过二分法。
  + GfxDriver：显存资源。
  + FMOD音频资源：比较单纯，profile就可以看出来了。
  + Texture，Mesh，Animaiton，Audio等等：推荐使用memory或者uwa。

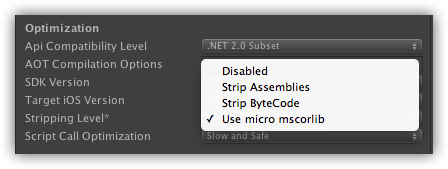
下面介绍一些具体的优化方式。

* + 压缩贴图ETC/PVR：贴图是占用资源最大的部分。我对3D贴图基本上都会压缩，2DUI贴图部分压缩。在Android上尽量使用etc格式，IOS上使用pvr格式，非半透有1/8的压缩比。这两种压缩格式类似DX的dds，可以直接被显卡渲染，即降低内存又能减少包大小，提升加载速度（JPG等格式虽然压缩比高但是需要解压成32位色再渲染，增加加了内存和显存还有额外的解压开销）。需要注意以下几点：
    - 如果压缩效果不好还可以减成16位色。
    - 关Mipmap：UI或者俯视角游戏不需要Mipmap可以关闭减小1/3的体积
    - 如果使用ETC，PVR，DDS压缩贴图必须是2的幂（因为区块颜色索引的压缩算法，如果不是2的幂压缩后也会补成2的幂），正方形（PVR还必须是正方形否则会自动补成正方形浪费空间）。处于渲染效率和显存碎片的考虑贴图的大小建议最大1024，最小64，最大不能超过2048。
  + 使用九宫，对称贴图：提高贴图复用率
  + Shader：利用Shader合并贴图通道，实现灰度图等。Alpha通道存alphatest和高光，贴图一个通道存阴影一个通道存ao等，alpha通道存在贴图的其他通道便于压缩等等。这点在做轩辕传奇的时候大幅使用。
  + 压缩动画减少关键帧：前面有介绍
  + 及时卸载**：**在进出场景时，或者打开UI界面等对性能不敏感的时候，卸载资源并调用Resource.UnloadAsset清理引用资源和destroy，System.GC.Collect清理系统资源，前面有介绍。AssetBundle加载时生成、卸载时销毁，这也是比较大的一个坑：）
  + 在代码级别上避免不必要的堆内存分配：可以通过静态代码分析检查，另有一篇文章详细介绍。
    - 避免频繁New Class：使用内存池。
    - Constainer：新版本Unity Constainer已经优化。
    - 控制Log输出：我们使用条件限制，Release自动屏蔽一些不重要的日志
    - For代替Foreach：新版本Unity已经优化。
    - String连接：减少字符串拼接，使用StringBuilder等等。
    - delegate：因为内部的链表和装箱拆箱操作，使用频率较高时GC也很高。
    - 合理的使用Lambda表达式：例如Unity的粒子系统5.6版本以前GC较高就是这个原因。
    - 频繁的临时变量或者list生成，建议定义一个全局的list每次都用该list来计算。
    - 需要注意的是类申请在堆上，结构申请在栈上，有时可以使用结构。
    - 等等，这方面优化点和文章较多不再累述。
  + 内存泄露： Unity是基于引用计数的，一般内存泄漏是资源被Hold住无法释放，内存增长趋势明显、反复切换场景内存膨胀。针对这种情况可以自己写工具输出每个场景的资源日志。也可以使用XCode分析一段时间的内存。
  + 表数据。表一般不会卸载，15年帮一个项目做优化，发现光表数据就占用40M。建议使用二进制反序列化使用不要直接使用字符串，并且不在内存中做多份缓存，表数据非常巨大的情况下可以考虑使用完删除。
  + 冗余顶点数据:UI贴图Mesh把color，normal等不适用都导出，静态合并会导致内存增大。
  + 抗锯齿/Rendertexture：开了抗锯齿会增大内存，高分辨率会增大内存，后期处理可以交换使用Rendertexture不要创建多份。
  + GameObject数量：小于1w，节点书过多也会导致加载更新缓慢内存膨胀，可以写工具监控并作为测试时的一项性能指标
  + 其他

**闪存**

在PC上也就是硬盘。具体表现在包大小，资源加载速度等。闪存的优化和内存大部分共同，但是也有例外，例如使用jpg就是减少包大小增大内存消耗CPU的方法。

* + 压缩和内存一样。贴图，动画，导表等等。
  + 在移动平台上可以打开code strip功能来减少代码带来的内存和容量消耗。对于使用了反射的类，可以使用link.xml配置解决。code strip是Unity提供的一个优化功能，他会预判断代码的执行路径，将没有使用的函数去掉。



* + 部分SDK很大，可以和第三方协商减少重复包含的库。
  + l2cpp 中会包含 ARMv7 和 ARM64 位两个版本的代码，因此，会有两份代码的体积。
  + 动态下载：类似微端。但在手机上并不推荐，可能会导致使用玩家流量。
  + 冗余资源：导出包的时候利用插件或者自己写的工具分析资源。在prefab中搜索资源是否被引用。
  + 贴图mesh动态生成：对于一些规则画贴图和模型可以通过计算生成出来，例如以前使用过的著名的substance，还有之前端游优化地形的时候只保存地形的高度，通过多留的方式在GPU中还原成地形信息，可以减少1/3等等。
  + 其他：包体的优化相对CPU，GPU，内存没有那么重要，但是对游戏的推广非常重要，也值得花精力去优化。

**网络**

* + 减小包体和压缩：在二进制上进行一些重用和合并减小包体，对协议包做压缩。
  + 合包：按一定频率进行合包操作，把几个包合并在一起降低发送频率，减小包头。MMORPG为了防止包过大可以采取裁包，当玩家同屏较多时抛弃一些不重要的协议。最近的项目为了防止超过MTU，会做一些负载均衡，当操作过多的时限制每帧的操作数，把一些操作放到后面的包广播。
  + 网络同步框架和策略。最近开发的一款射击MOBA游戏，就使用了帧同步。对于大量小兵的RTS和MOBA类游戏，帧同步的协议量可以比状态同步小n倍，我们的延迟和流量已经优化到和王者荣耀类似，而状态同步的全民超神比帧同步的王者荣耀高3倍左右。具体帧同步和状态同步的优缺点会写更详细的文章讨论。

**耗电**

耗电可能不会被大家所重视，但其实在手游上这是一个非常重要的指标。如果游戏打一两个小时就没电了，那出门在外谁敢玩呢？火遍全国的王者荣耀就在省电上做了非常多的优化，我们游戏也是。那么有哪些优化方案呢？

* + 降低CPU/GPU：CPU和GPU非常影响耗电，因为手机CPU和GPU封装一个SOC上的，因此主要体现在手机CPU占用上，当CPU低于25%会非常省电当CPU高于80%耗电就会从呈指数上升。和CPUGPU的优化是重合。
  + 内存闪存：内存闪存的使用频率也会影响耗电，和内存闪存的优化重合。
  + 网络：和网络的优化重合。
  + 限帧：限帧是比较常用的优化手段，手游一般会限帧30帧，早期苹果系统强制限帧30帧后来才开放的。王者荣耀已经把限帧作为一个广告手段了，哪个设备支持都会做一番宣传。但是限帧会降低手感需要权衡。
  + 省电模式：当检查到玩家没有插电的时候，配合限帧，降低画质和效果，降低更新频率和LOD等。
  + 其他

**总结**

* + **扁鹊三兄弟VS过早的优化是万恶之源**

优化有两个理论，Donald Knuth曾说过“过早的优化是万恶之源”，因为让正确的程序更快，要比让快速的程序正确容易得多。但与之相反的就是扁鹊三兄弟的段子，最好的医生是再出问题之前就先避免问题的，特别是游戏等到后期美术资源代码都非常庞大的时候再优化成本非常高甚至来不及优化。

在这个方面上我认为两种观点都有道理，对软件开发来说更倾向中后期优化，在产品开发程度在50%的时候开始测试崩溃率，在完成度80%的时候启用兼容性测试。而对游戏来说因为美术的原因又比较特殊，个人感觉前期应该先验证玩法，效果。等到立项阶段就开始测试并制定各种规范，因为美术的优化后期成本非常高。最后在游戏的中后期做有针对性的持续优化。

* + **持续的过程**

优化是一个持续的过程，因为游戏开发是一个变化非常大的过程。随着开发和需求的变更，会不断出现性能瓶颈。优化一般会持续到游戏的整个生命周期。

* + **没有万金油，具体问题具体分析**

优化其实没有通用的方法，例如预加载是空间换时间，资源压缩是时间换空间。具体的优化方案要根据需求根据项目更具优先级来制定。