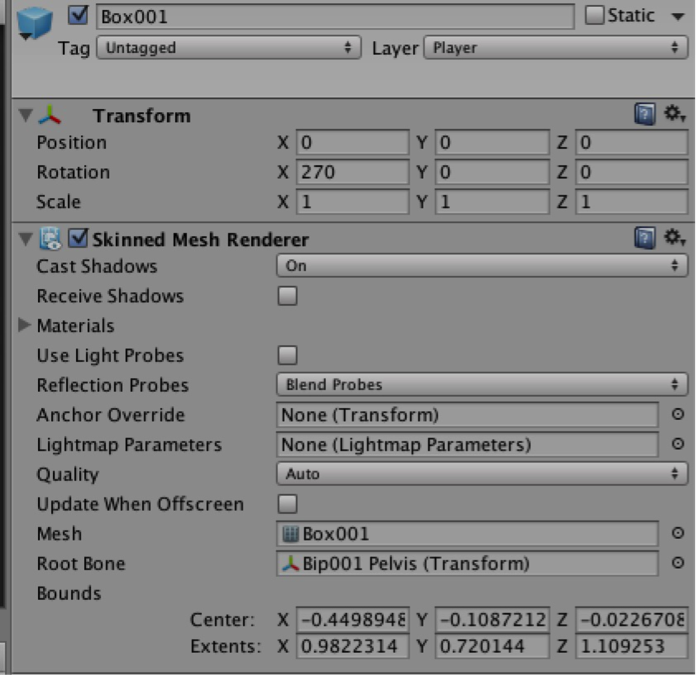
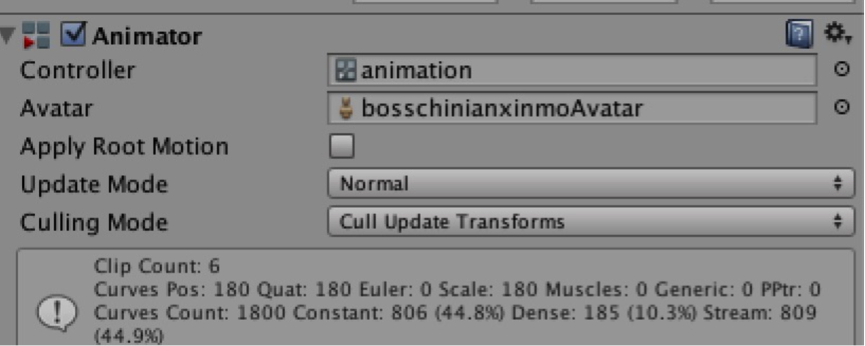
<https://blog.uwa4d.com/archives/QA_Rendering.html>

# [关于Unity渲染优化，你可能遇到这些问题](https://blog.uwa4d.com/archives/QA_Rendering.html)

**Draw Call**  
**半透明物体渲染**  
**多层纹理渲染**  
**Graphics.PresentAndSync**  
**VBO**  
**相机后处理特效**

## ****一、Draw Call相关****

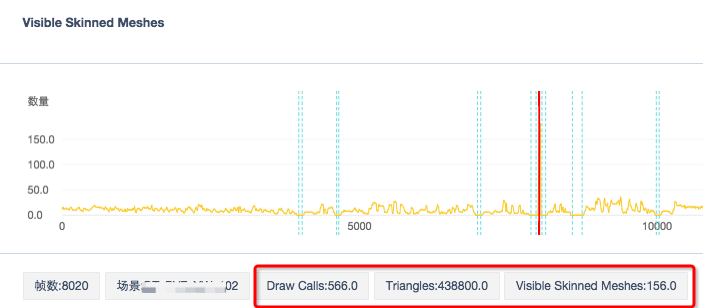
**Q1：移动游戏场景中，相同的怪物，Draw Call会动态合并吗？如下设置可行吗？**  
  


默认情况下，带蒙皮的Mesh是不支持动态合批的。如果场景中相同材质的蒙皮网格数量很多，可以考虑通过插件MeshBaker来进行合并，具体方法大家可以参考[好插件让你事半功倍！](http://blog.uwa4d.com/archives/Resource_Plugin.html)

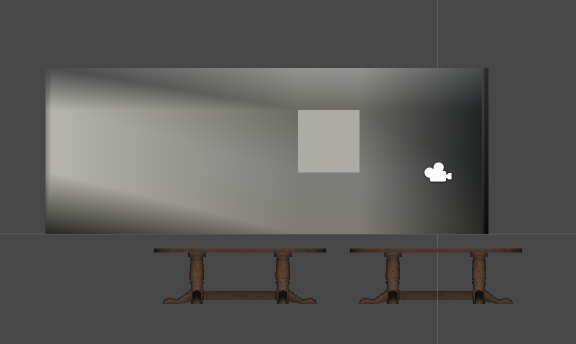
**Q2：Draw Call和Setpass Call，这两个指标主要是看哪一个？关于这点众说纷纭，很多地方都是说看SetPass Call，但是在UWA的性能测试中，还是把Draw Call当成唯一指标。**



在 Unity 5.x 中，SetPass Call与 Draw Call相比，SetPass Call的指标与性能相关性更大（比如Static Batching的开启不影响Draw Call数，而SetPass Call通常会明显下降）。但 SetPass Call在某些情况下也同样存在问题，比如往一个场景中添加任意个相邻且材质相同的大网格物体（使Dynamic Batching失效）时，SetPass Call并不会变化。因此在UWA中，我们所使用的是类似Profiler 中 Total Batches 这一项指标，通常该数值与 Frame Debugger 中的数值基本一致，因此可以通过该工具来查看每个Batch的内容，从而更有针对性地进行优化。

**Q3：我UWA报告中“渲染模块”界面中看到大部分场景中的三角面片数是正常的，但在某一帧时DrawCall、Traingle、蒙皮网格数骤然提升，请问这可能是什么原因导致的？这三个参数的变化曲线是否有规律？为什么我在切换场景的时候也会有400多的DrawCall呢？**  


从图中看，这种情况发生在场景切换处。这种情况的发生原因很可能为一次性动态加载大量GameObejct，然后再手动Deactive当前并不需要使用的GameObject。研发团队可以查看该峰值处的场景名称和相关截图，从而来进一步定位发生该问题的根本原因。

**Q4：关于Static Batching, 场景中物件组合成大的Mesh，那么判断子Mesh要合入大的Mesh中的依据是什么？材质？勾选Static?**  
  
**我有一个模型A并勾选Static，使用材质A，怎么看到也和其他材质的Mesh合并到一块去了(Combined Mesh)？**  
  


勾选Static的GameObject下的Mesh都会被合入CombineMesh（无论什么材质），且每个Mesh都作为SubMesh存在。在Unity 5.3之前，对于渲染顺序相邻且材质相同的SubMesh则会动态将其索引数组拼合，从而合成一个Draw Call。而Unity 5.3之后则不再拼合索引数组，因为在不切换材质时产生多个Draw Call的开销并不大，而这多个Draw Call会被统计为一个Batch。

**Q5：Unity对Dynamic Batching的数量是否有限制？或者说对Saved by Batch的数量是否有限制？**

Unity对于任何Mesh的面片都有65536的个数限制，拼合后的面片数也是如此。

**Q6：请教，角色分部件换装可行吗？比如衣服裤子分开，都是用Skinned Mesh Render，有没有办法合并降低Draw Call？**

可以通过合并网格的方式来达到降低Draw Call的效果，具体可查看Asset Store中的换装例子：Character Customization。但是，在角色换装时需要注意以下几点：  
（1）装备与角色必须是共用一套骨骼的；  
（2）各装备之间所用的材质必须相同。  
开发者需要注意，只有同时满足以上两个条件时，才能达到只使用少量Draw Call来进行动态换装的效果。

**Q7：请问，Canvas里的东西移出了屏幕后，DrawCall没降低，那么它还会每帧去绘制吗？**

DrawCall没降低，说明CPU依然将这部分的网格提交到了GPU。因此虽然UI元素已经不可见，但其CPU开销（包括切换渲染状态，提交VBO等）依然是在的，只是对GPU不会造成明显影响，因为最终并没有进行像素的渲染。

**Q8：能否对提升NGUI的渲染效率提供一些思路？**

开发团队可以从以下几点入手：

1. 通常一个Panel会产生1个或多个Draw Call，以一个Panel为单位，Draw Call 的数量通常由当前 Panel 中使用的Atlas、Font的数量所决定。
2. 要降低UI渲染时的 Draw Call数量则需要对 Atlas 的制作进行合理的规划，即在保证使用较少的 Atlas 的同时，还需要保证 Atlas之间不会存在交叉遮挡。
3. 要注意UI Texture的使用，每个UITexture自身会占用一个Draw Call，同时如果其Depth值穿插在了其他来自相同Atlas的UISprite中，还会导致Draw Call的打断，造成不必要的额外Draw Call。
4. 另外还可以借助Panel Tool和Draw Call Tool来对UI部分的Draw Call进行分析，前者可以显示每个UIPanel包含了多少个Draw Call，而后者可以显示每个Draw Call由哪些UIWidget组成。"

**Q9：关于场景中玩家和NPC名字的DrawCall的问题。我们项目中是使用TextMesh挂到场景单位上，但是这样每个名字就占了一个DrawCall，请问有没有好的办法优化呢？**

游戏中的HUD的做法一般有两种，一种是如上的做法，另一种则是通过NGUI/UGUI来制作HUD。第二种的实现方法大致如下：

1. 计算屏幕中角色在屏幕中的位置；
2. 根据屏幕中的位置来计算各自HUD的位置，并根据HUD的数量分别放置在一个或几个Panel/Canvas下。  
   第二种方法的优势是尽可能用少的Draw Call数来渲染角色的HUD。开发团队可以就该方法来进行尝试。

## ****二、半透明物体渲染相关****

**Q1：这个批渲染是什么？好像开销很高 。**  

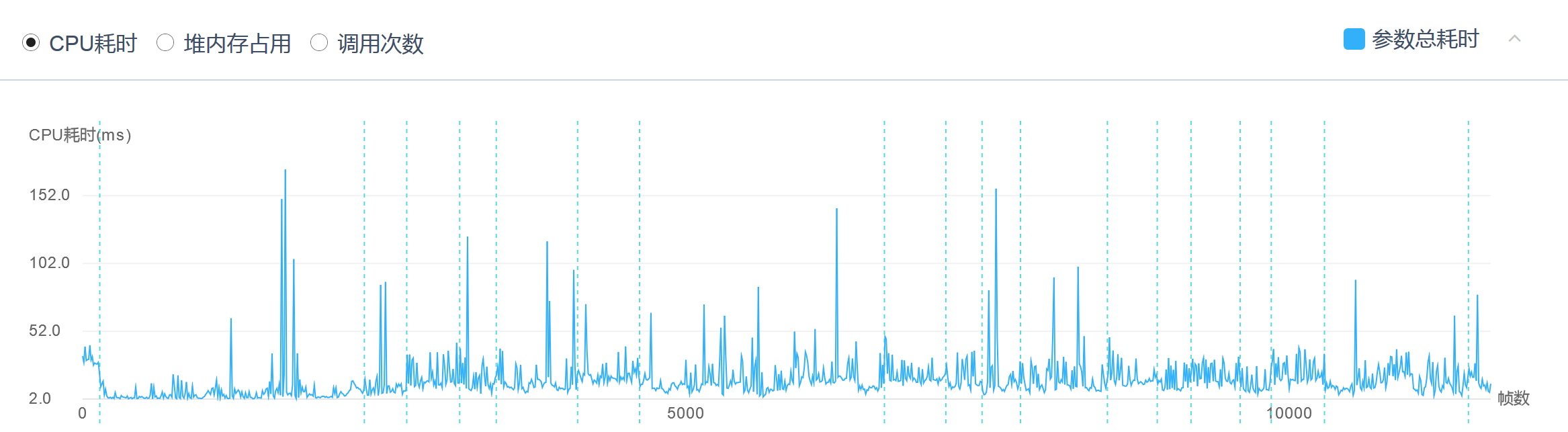

从图中看出，这是场景中不透明物体的渲染开销。建议研发团队对当时场景中的不透明物体（地形、建筑等）进行进一步检测，主要查看其三角面片数是否过高、Shader是否过于复杂等。

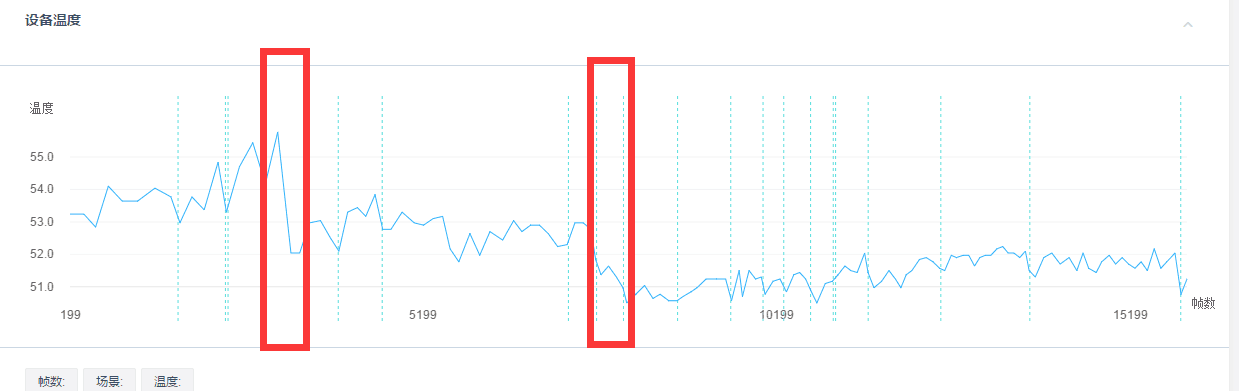
## ****三、多层纹理渲染相关****

**Q1：我们游戏用的是T4M，4层Tilling贴图+1层融合贴图，发现手机发热现象严重，影响性能的表现，请问有什么标准或者参考数据吗？**

对于中低端机器来说，我们建议地形纹理所刷的层数要尽可能小于3层。在中低端设备中，纹理采样次数越多，则GPU的压力越大，发热效果也就越明显。

在UWA性能测评报告中，我们加入了针对Graphics.PresentAndSync的统计，从而让大家来看到项目运行过程中，GPU的压力情况。同时，在设备的温度显示中，建议大家关注温度的走势图，看看是否存在大幅向下回落的情况，如果存在，则很可能是设备因为过热而主动降频。

**Graphics.PresentAndSync耗时统计**  


**设备温度走势**  


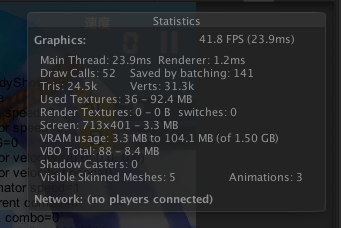
## ****Graphics.PresentAndSync相关****

**Q1：我们在编译安卓版本时，在某些设备上（Sony L36h,小米4）调试时，发现有一个时间消耗项叫Graphics.PresentAndSync，该函数对性能的消耗会特别夸张（渲染20毫秒，这个能达到50毫秒）。查了相关文档，发现该函数好像和安卓设备的垂直同步有关，但是大部分安卓设备的垂直同步是不可以关闭的。请问有什么好的办法解决吗？**

概括来说，该值很高表示 GPU 负担很重，可以从降面，或者简化shader入手。  
Graphics.PresentAndSync 是指主线程进行Present时的等待时间和等待垂直同步的时间。该参数在Profiler中CPU占用通常较高，且仅在发布版本中可以看到。究其原因，其实是CPU和GPU之间的垂直同步（VSync）导致的，主要是与项目是否开启多线程渲染有关。当项目开启多线程渲染时，你看到的则是Gfx.WaitForPresent；当项目未开启多线程渲染时，看到的则是Graphics.PresentAndSync。

其中的原理，可以参照我们之前对其函数的详细解释：[扒一扒Profiler中这几个“占坑鬼”](http://blog.uwa4d.com/archives/presentandsync.html)。

## ****VBO相关****

**Q1：我们现在有一个场景，Draw Call和面数都在正常范围内，Camera的距离也和其他场景一样，但是VBO却非常高。下图是该场景的数据情况，该场景下有很多复用的模型，如果不勾选Static那VBO会下降，但是Draw Call会上升。那我能否通过合并模型的方式把VBO降下来呢?**  


当你勾选Static时，Unity 会将其进行 Static Batching，进而将生成一个较大的VBO来进行渲染，同时降低Draw Call。而如果不勾选时，则引擎无法对其进行Batch，所以VBO会降低，而Draw Call升高。  
对此，我们的建议如下：  
1、一般来讲，降低Draw Call的意义大于降低VBO；  
2、在Draw Call较低的情况下，比如当前的50+，可以考虑适当增加一些Draw Call来降低VBO的占用，一方面可以降低带宽的压力，另一方面也可以适当降低一些内存。

## ****相机后处理特效相关****

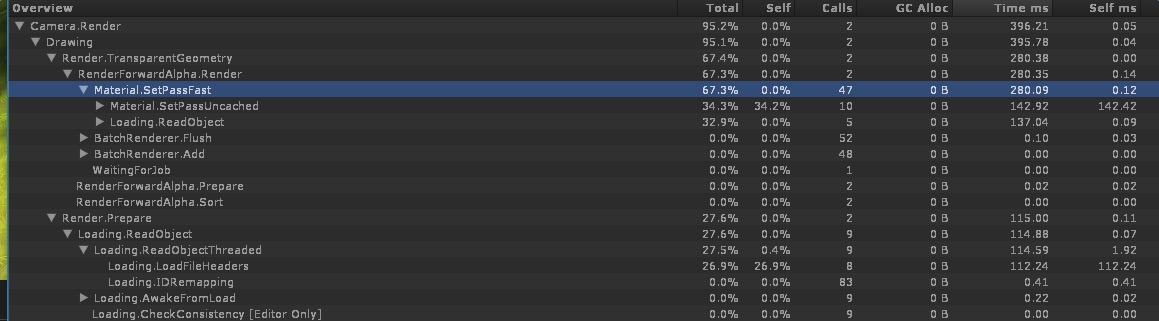
**Q1：关于抗锯齿和BLOOM，有什么好的优化方案或者优秀插件推荐？**

通常在中低端的设备上，抗锯齿并没有比较高效的方案；而对于中高端的设备，可尝试直接使用 Unity 内置的 MSAA 功能，但也只推荐使用 2x。  
关于Bloom效果，以下是适用于移动端，且评价较好的一款插件：BloomPro  
  

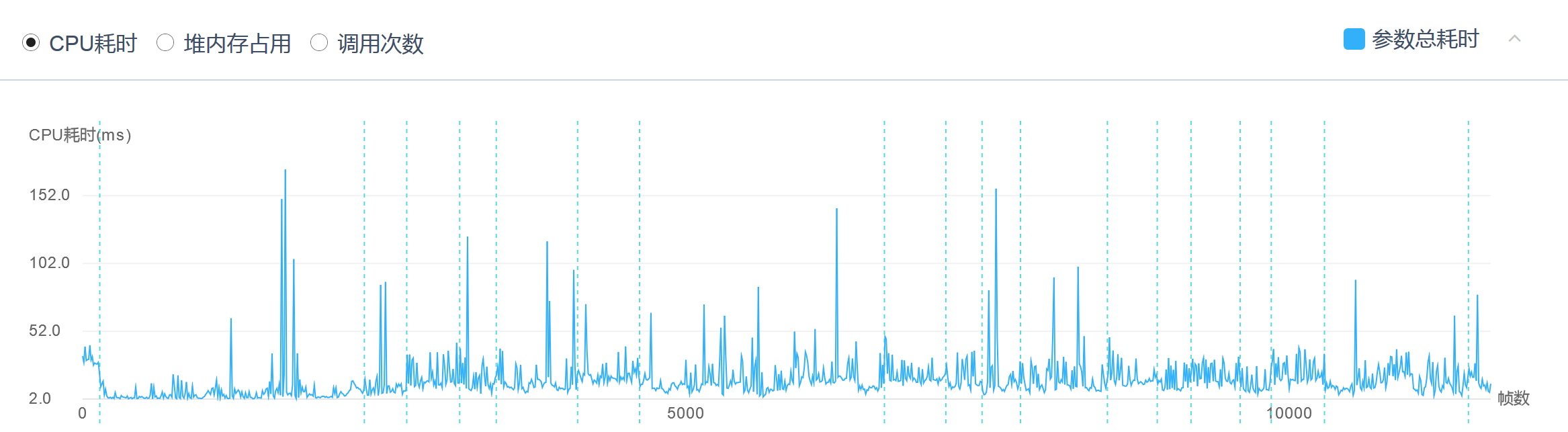

**Q2：如何在移动设备上，对Bloom和全屏抗锯齿进行优化？Unity标准资源里面自带的效率比较低（已经尝试过Bloom(Optimized)）。**

建议使用Asset Store上适合移动端的Bloom Shader插件，比如FxPro: Bloom&DOF和BloomPro等。  
对于AA，目前在移动设备上并没有特别优化的方法，仅能建议在低端设备上关闭AA功能，而在高端设备上可尝试开启较低倍数（2x）的MSAA。

## ****Shader解析相关****

**Q1：图中的Material.SetPassFast占用很高，这是我在第一次实例化一个特效，但是第二次实例化就不会出现高值了，请问能怎么优化吗？**  


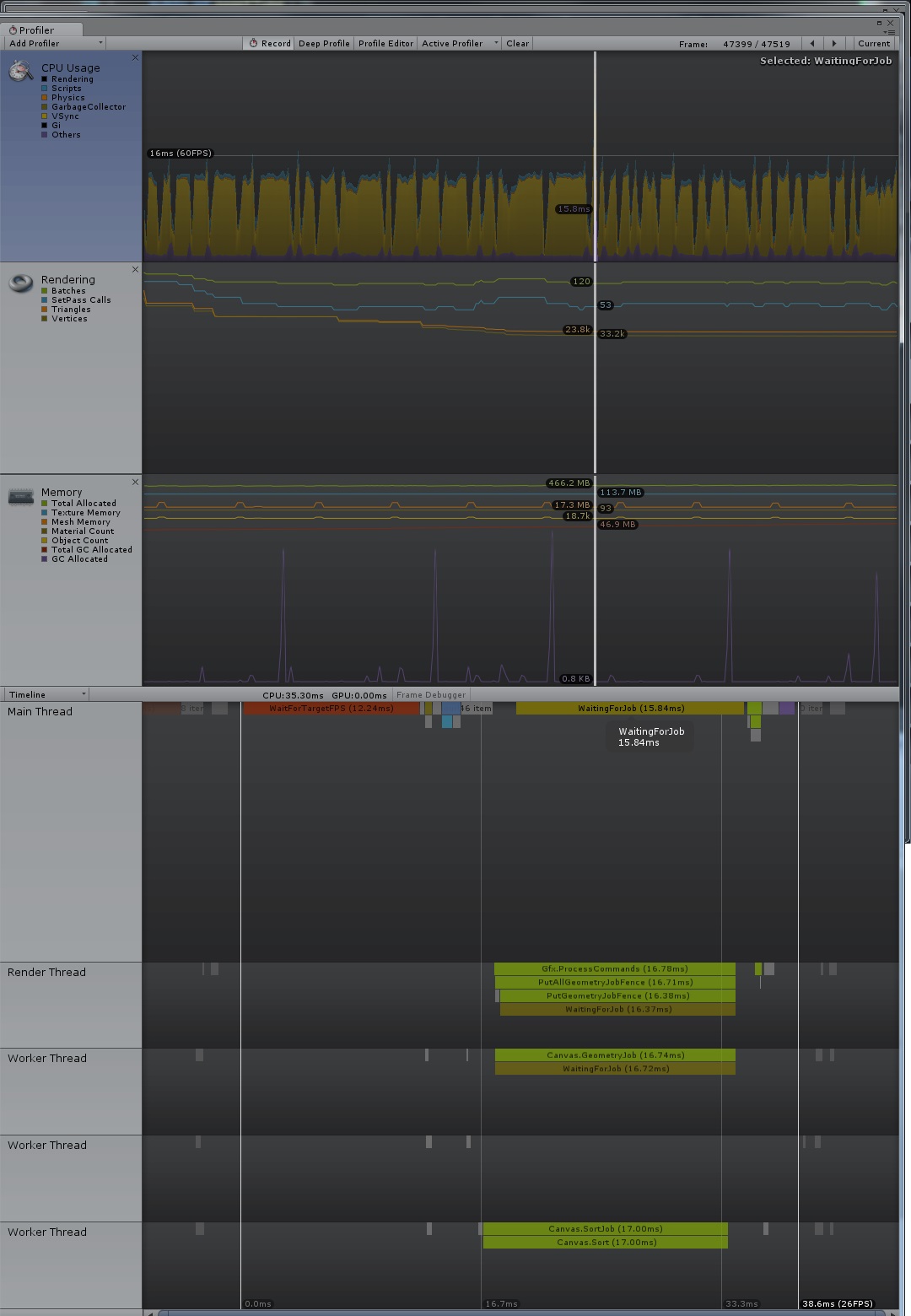
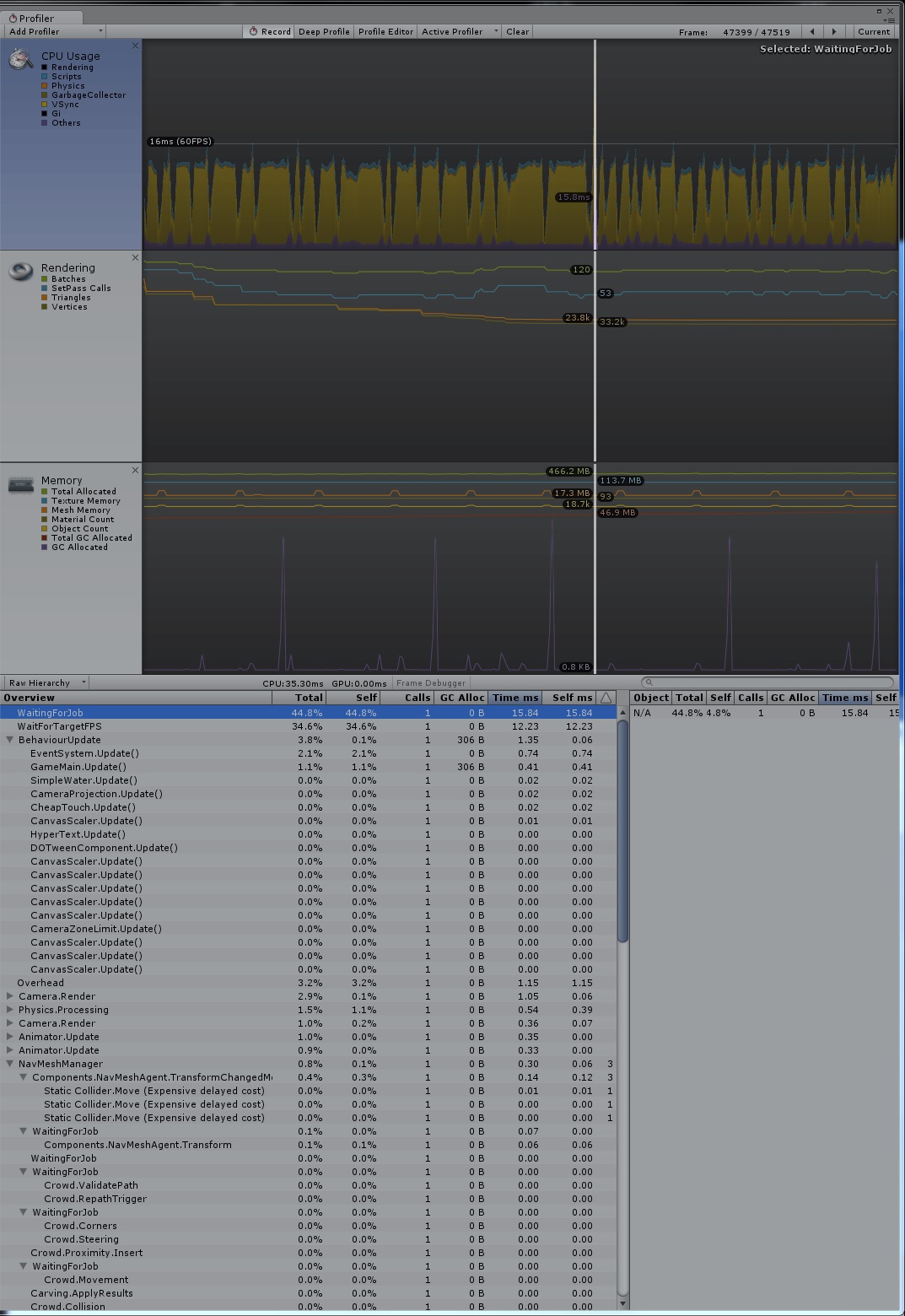
该过程是在处理Shader，Unity 5.3以后在第一次显示时才会将Shader进行Warmup，所以就会造成一次峰值卡顿。研发团队可以参考我们之前的分享：[Unity加载模块深度解析之Shader篇](http://blog.uwa4d.com/archives/LoadingPerformance_Shader.html)以加深理解。

**Q2：Shader.Parse 和 Shader.CreateGpuProgram 到底是做什么的？它们什么时候执行？**  


Shader.Parse体现的是Shader的加载和解析， Shader.CreateGpuProgram 是将Shader传入GPU的一次提交，GPU驱动会对其进行编译，以适应于特定的设备或平台。在Unity 5.x版本中，Shader.Parse在Shader资源加载时进行执行，而 Shader.CreateGpuProgram在所在GameObject第一渲染时进行执行。

## ****其他****

**Q1：如下图，我们发现WaitingForJob这个函数消耗过高导致了卡顿，请问该卡顿是否由于渲染压力过大导致？**

从图中看，该线程最后是在等待 Canvas.sortjob，而这是 UI 排序造成的开销（自Unity5.2版本开始，UGUI的部分计算已经移出了主线程）。  
详情参考：<http://blogs.unity3d.com/2015/09/07/making-the-ui-backend-faster/>  
因此理论上，这是 UI 的 canvas.sortjob 在指定的时间上没有完成，从而使得渲染线程等待，且最终导致主线程进行等待而造成的开销。