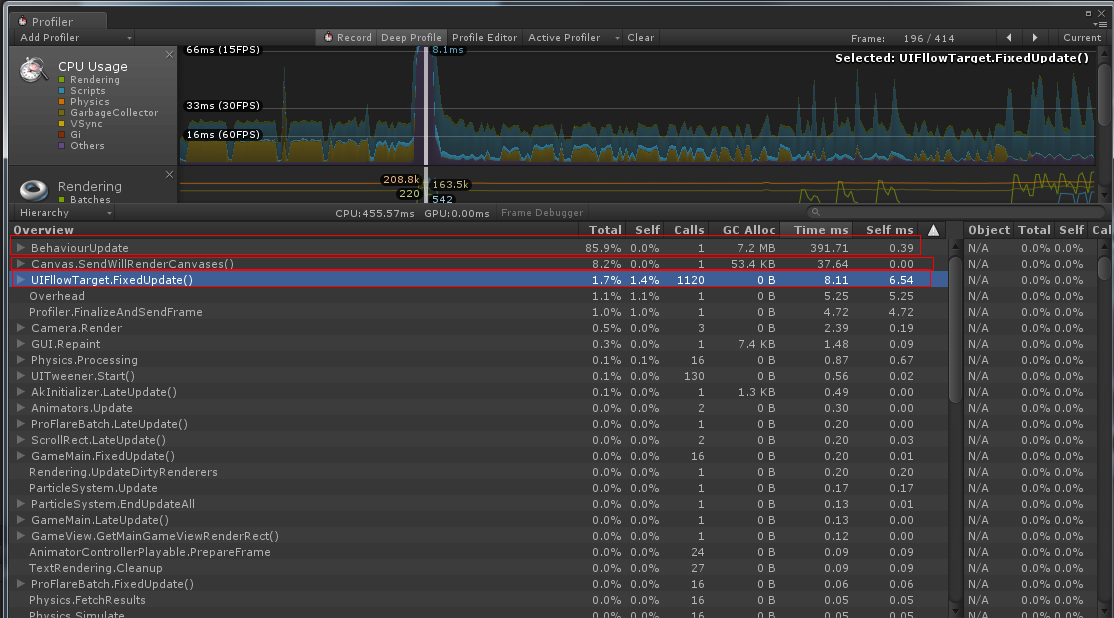
我在界王神界招了20个Npc攻击时，能引发极其明显的卡顿：

卡顿峰值：



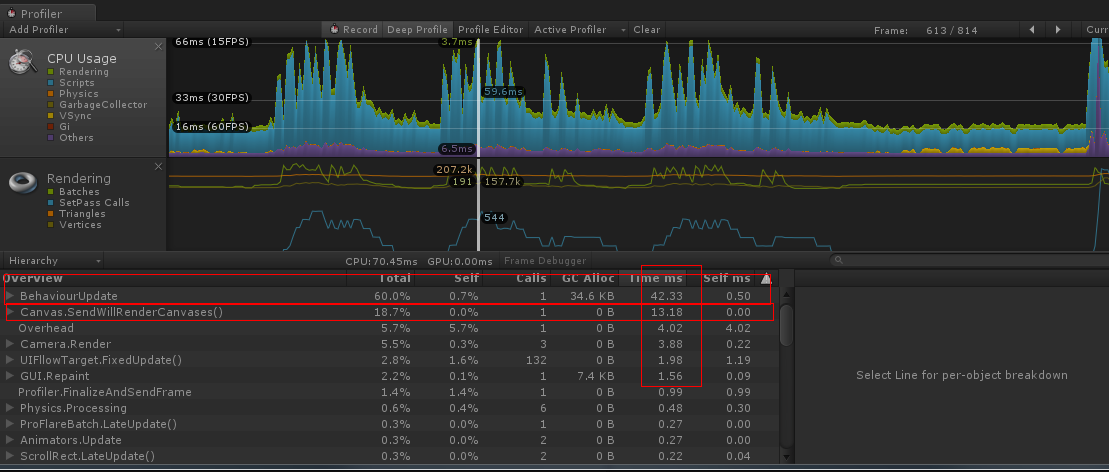
造成卡顿的元凶前三名分别是：

BehaviourUpDate耗时：391.71ms，

Canvas.SendWillRenderCanvas()耗时：37.64ms，

UIFllowTarget.FixUpDate()耗时：8.11ms

第一次卡顿之后，不停的用技能打出伤害，稳定峰值：

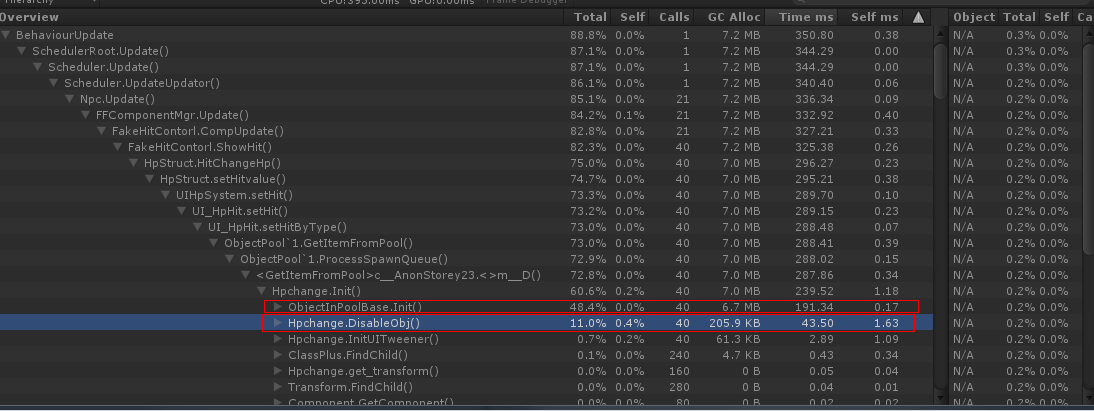


BehaviourUpDate耗时：42.33ms，

Canvas.SendWillRenderCanvas()耗时：13.18ms，

UIFllowTarget.FixUpDate()耗时：1.98ms

1. BehaviourUpDate：



如上图所示，主要消耗在HpChange.Init()方法。Init放法中主要消耗在于ObjectInPoolBase.Init (Gameobject的Instantiate（）)和Hpchange.Disable(SetActive(false)）。

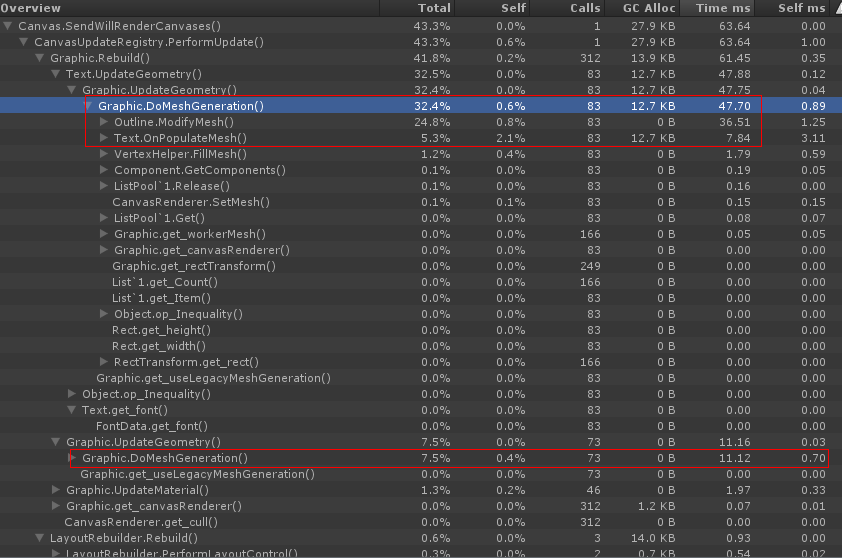
应对方案：

1. 在UI上默认添加一定数量的伤害数字Panel，每次显示伤害数字时不用再Instantitate，或者调用少量的Instantitate。
2. 在伤害数字消失时，关闭伤害数字跟随角色脚本，把伤害数字移出屏幕而不是使用DisActive。

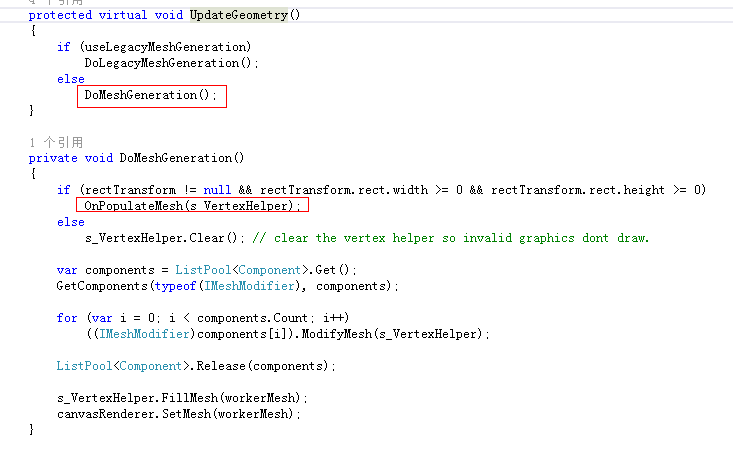
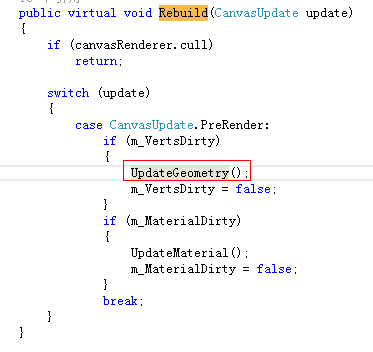


1. Canvas.SendWillRenderCanvas()

Unity的官方文档解释是Event that is called just before Canvas rendering happes。



如图我们可以看到主要消耗在Graphic.DoMeshGeneration()方法(UGUI源码)：Graphic类



所以该方法的消耗主要是，UI控件位置移动导致

Graphic.Rebuild->Graphic.UpdateGeometry->Graphic.DoMeshGeneration.这个只要有UI移动就难以避免。但是主要消耗在OutLine，也就是字体描边。

解决办法：

1. 与策划商讨能否不加描边，或者自作字体的图片自带描边，不使用OutLine控件。
2. 减少伤害数字在Canvas下面的层级深度。

# PC端优化前

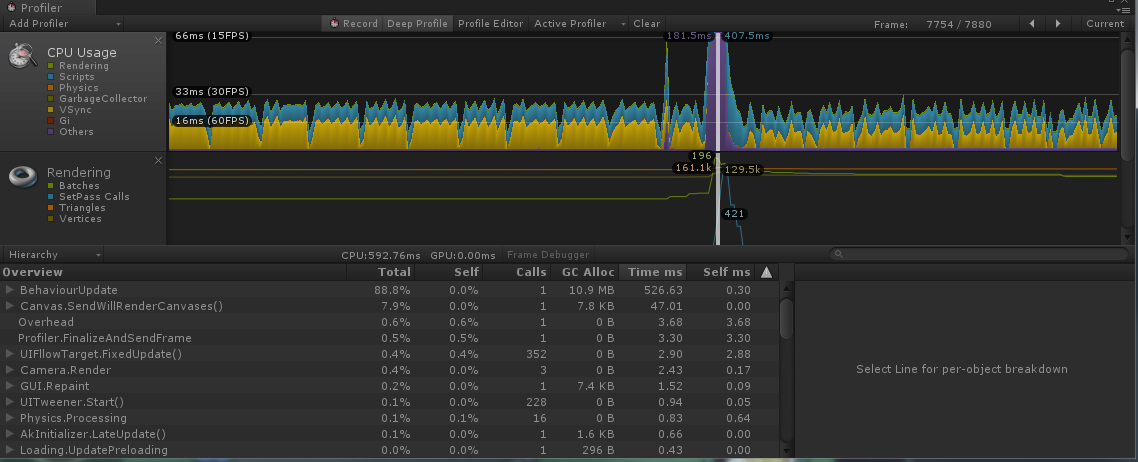
**测试条件，20个高级TF，用小悟空的1技能。群攻第一次爆伤害数字。**

1. 伤害数字卡顿耗时数据截图：

（1）CPU Usage项

BehaviorUpdate: GC: 10.9M Time:526.63ms

Cancas.SendWillRenderCanvases(): GC:7.8k Time:47.01ms



# PC端优化后

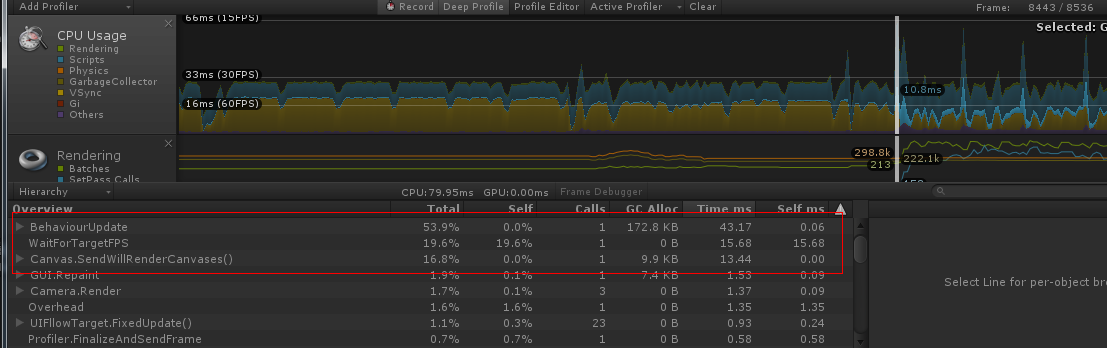
**测试条件，20个高级TF，用小悟空的1技能。群攻第一次爆伤害数字。**

1. 伤害数字卡顿耗时数据截图：

（1）CPU Usage项

BehaviorUpdate: GC: 172.8k Time:43.17ms

Cancas.SendWillRenderCanvases(): GC:9.9k Time:13.44ms

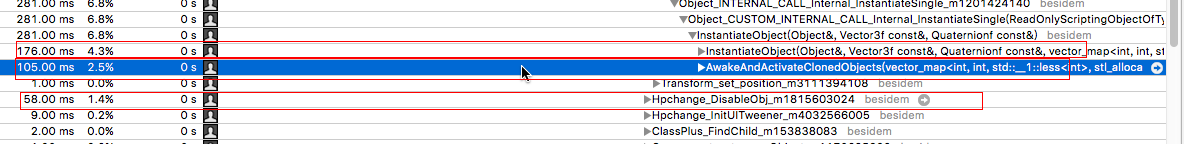


# IOS 真机：

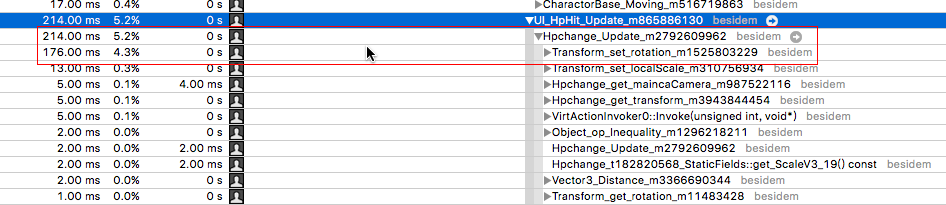
**测试条件，治安巡逻第一波怪，8个小怪，小悟空使用一技能，群攻第一次爆伤害数字**

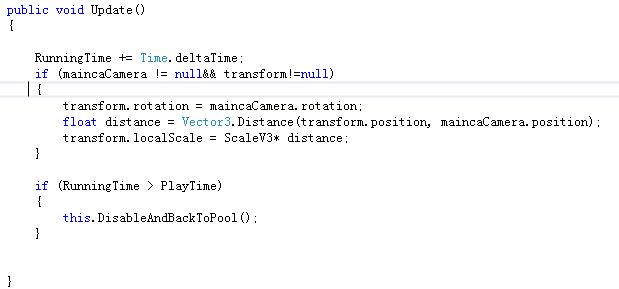
在Iphone6上真机调试统一截取第一次打出伤害数字的4秒数据

1、下图可以看到：281ms 的消耗主要来自于HpHit的Instantitate操作，即实例化伤害数字，以及active和disactive操作。

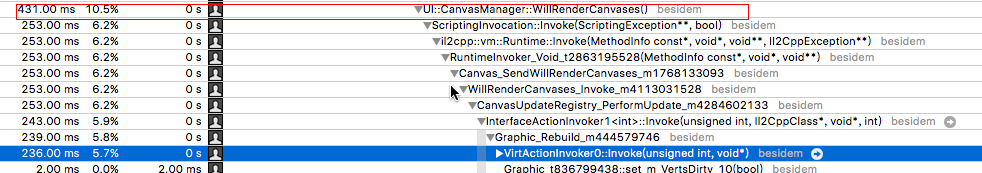


2、下图则是HpChange的Update的消耗为214ms。因为当先的伤害数字是在3d世界中，所以Hpchange的Update中会让伤害数字每一帧都朝向相机，并且根据与相机的距离调整大小。如下图所示，setrotation即让伤害数字旋转朝向相机占了很大一部分消耗。



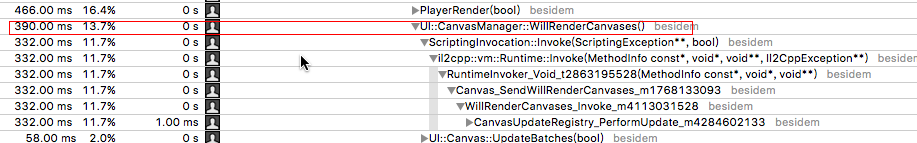


3、下面的这些消耗是在UGUI的控件动态变换时，UGUI会重新刷新Canvas下面的控件即SendWillRenderCanvas操作。这里消耗大是因为Canvas下面的层级深并且数量多导致，优化方向是，简化伤害数字层级，限制伤害数字数量，并且给伤害数字单独建立一个Canvas，以避免伤害数字变化时去刷新其他不必要的控件

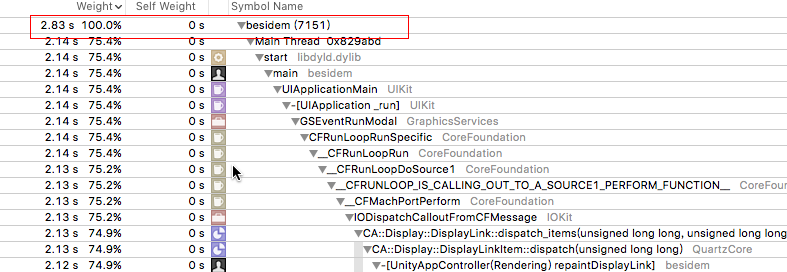


# IOS真机优化后

UI：CanvasManager.WillRenderCanvases(): 390ms 13.7%

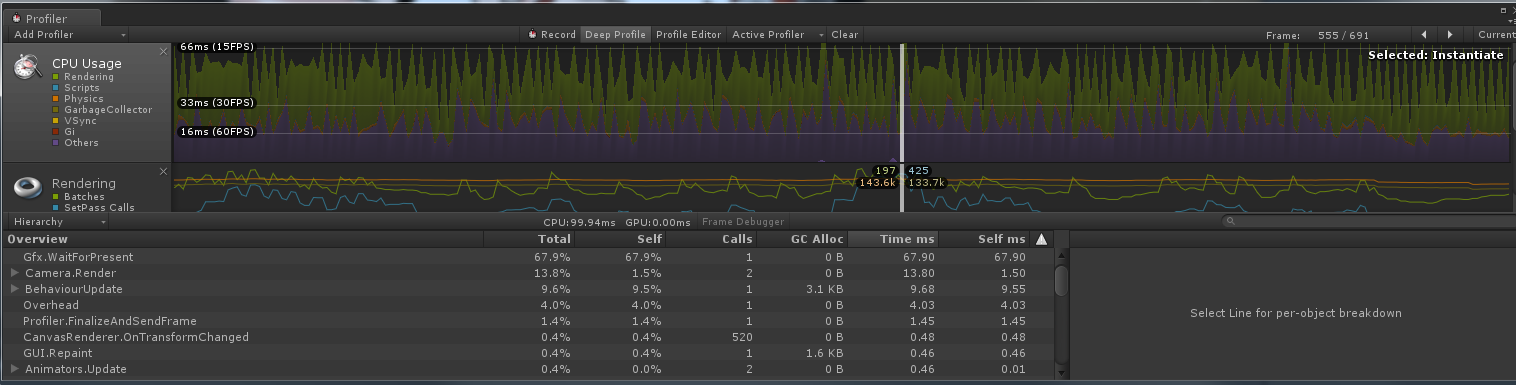


截取4s的总消耗为2.83s优化前总耗时是4.1s



# Android优化后数据

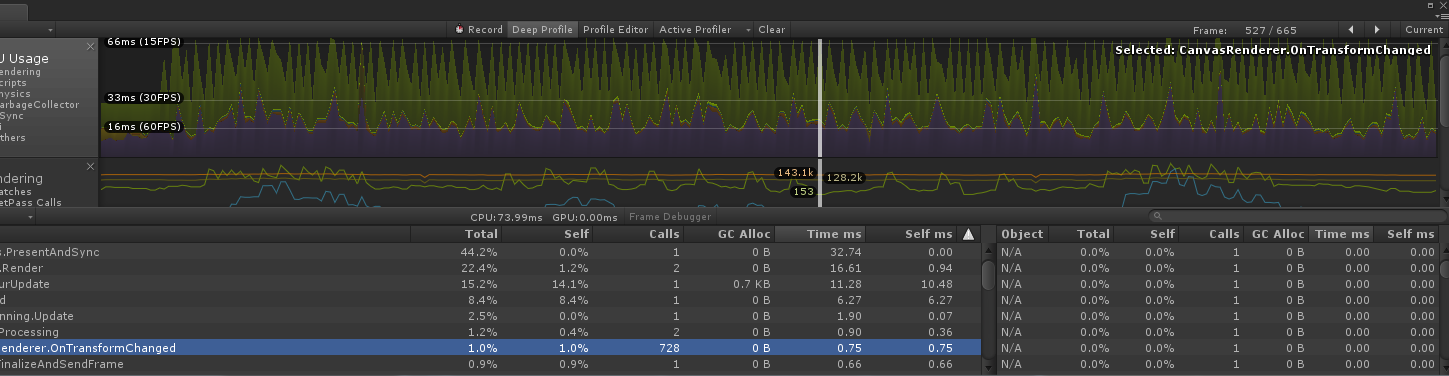
中配模式下，勾选Multithreaded Rendering



峰值在67ms左右，fps：21

均值在35—48左右，fps：16

中配模式下，不勾选Multithreaded Rendering



峰值在54ms左右，fps：25

均值在28—35左右，fps：17

# 优化前后数据对比

## PC端

|  |  |  |
| --- | --- | --- |
| 伤害数字卡顿 | 优化前 | 优化后 |
| BehaviorUpdate | GC: 10.9M ,Time:526.63ms | GC:172.8k ,Time:43.17ms |
| Cancas.SendWillRenderCanvases() | GC:7.8k , Time:47.01ms | GC:9.9k, Time:13.44ms |

## IOS真机

|  |  |  |
| --- | --- | --- |
| 伤害数字卡顿IOS | 优化前 | 优化后 |
| 总耗时 | 4.1s, 100% | 2.83s, 100% |
| InstantitateObject | 281ms, 6.8% | 伤害数字超过一定数量时才会有实例化否则没有目前40个 |
| Hpchange.Update | 214ms, 5.2% | 0,优化之后的伤害数字，不用update方法来确保数字面向相机 |
| Canvas.SendWillRenderCanvas | 431ms, 10.5% | 390ms , 13.7% |

优化方案总结：

1. 在UI上默认添加一定数量的伤害数字Panel，每次显示伤害数字时不用再Instantitate，或者调用少量的Instantitate。
2. 把伤害数字放在2DUI界面，就不用每帧根据相机距离来计算缩放了。

3、在伤害数字消失时，关闭伤害数字跟随角色脚本，把伤害数字移出屏幕而不是使用DisActive。

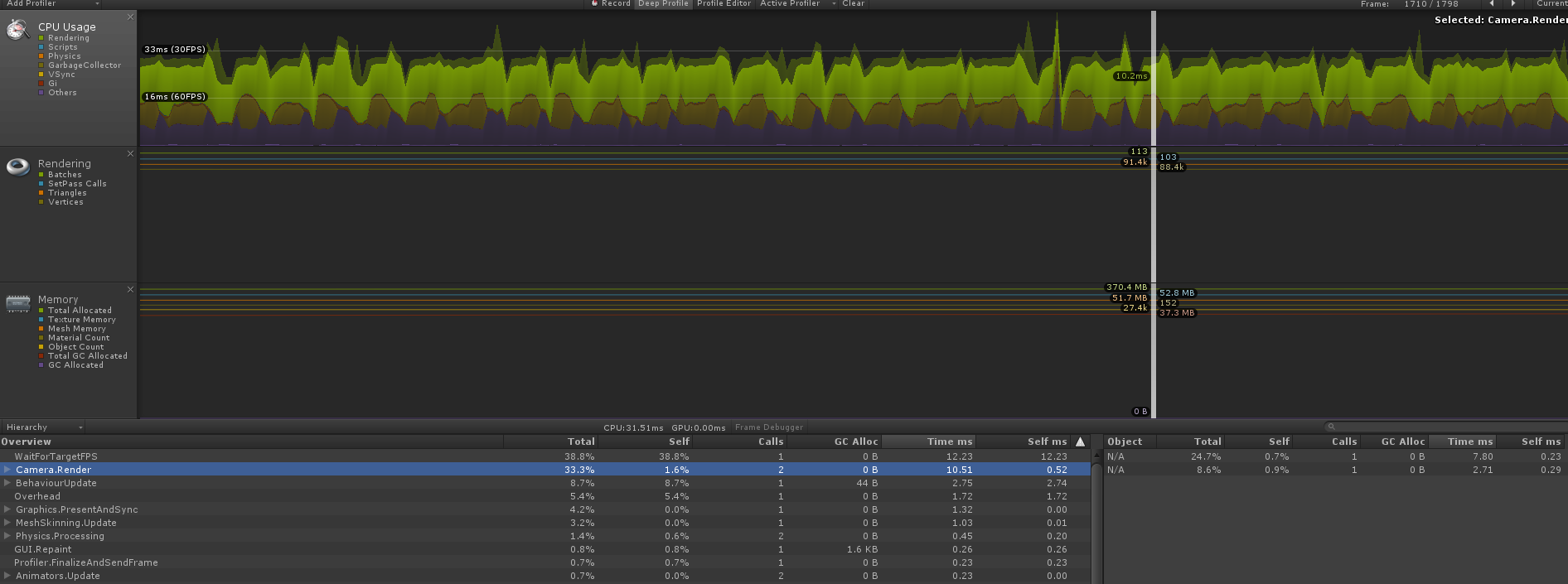
4、伤害数字暂时去掉OutLine控件，看一下效果。

5、减少伤害数字在Canvas下面层级的深度。

# Android多线程渲染选项测试

固定角度不勾Multithreaded Rendering选项

平均耗时11ms左右



固定角度与上面角度相同勾Multithreaded Rendering选项

平均耗时6.6ms左右

