Unity UI/GUI/UGUI/NGUI

deepwaterooo

April 12, 2018

Contents

1 Unity UI/GUI/UGUI/NGUI

1.1 Unity3D 快速实现 UI 架构设计一

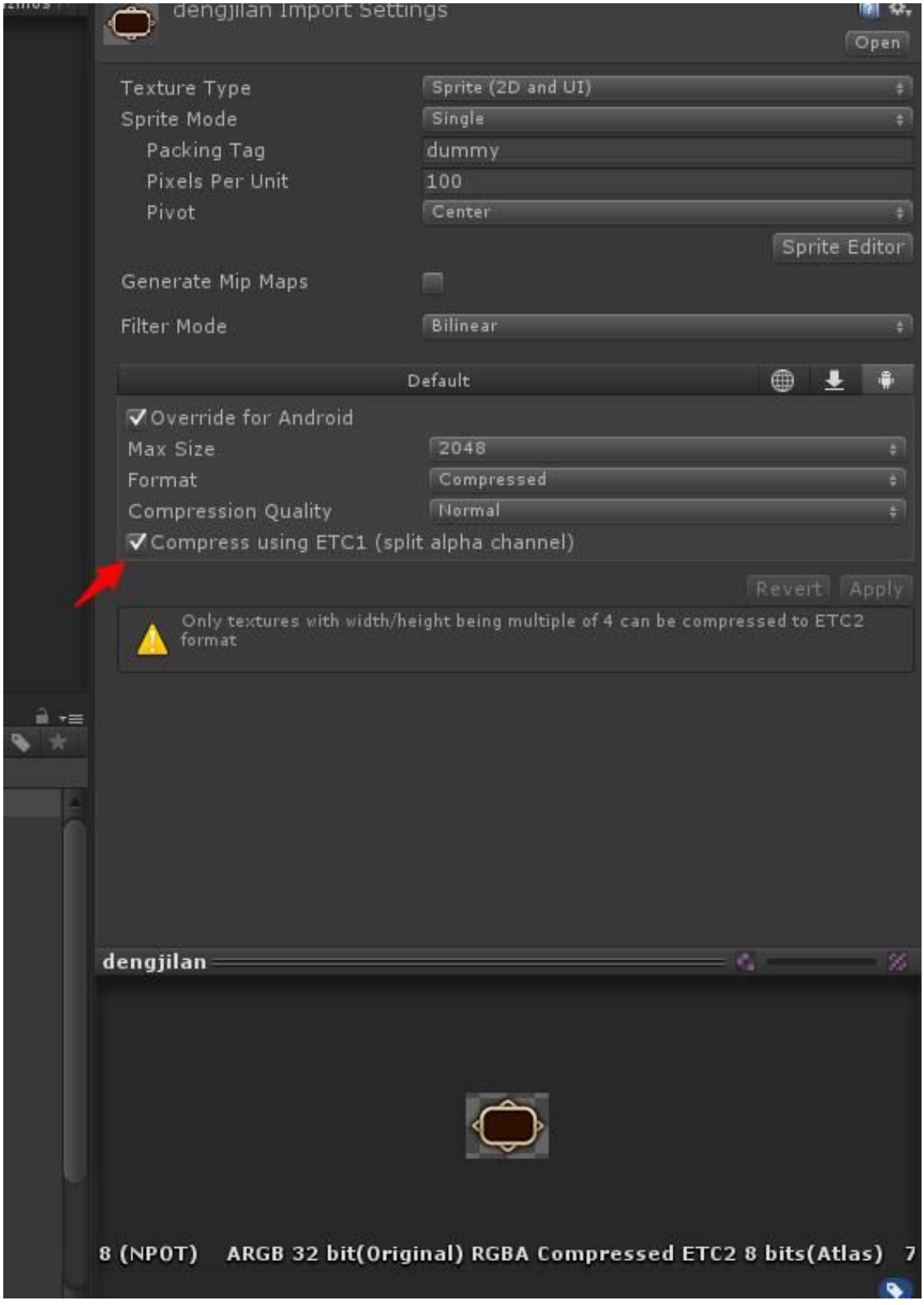
- https://blog.csdn.net/jxw167/article/details/72057771

1.2 关于 Unity 中的 UGUI 优化，你可能遇到这些问题

- https://blog.uwa4d.com/archives/QA_UGUI-1.html

1.2.1 界面制作

1. UGUI 里的这个选项，应该是 ETC2 拆分 Alpha 通道的意思，但是在使用中并没起作用？请问有没有什么拆分的标准和特别要求呢？



- 据我们所知，alpha split 的功能最初只对 Unity 2D 的 Sprite（SpriteRenderer）有完整的支持，而 UI 的支持是在 Unity 5.4 版本之后的。建议大家在 Unity 5.4 版本以后的 UGUI 中尝试该功能。

2. 在 UI 界面中，用 Canvas 还是用 RectTransform 做根节点更好？哪种方法效率更高？

- Canvas 划分是个很大的话题。简单来说，因为一个 Canvas 下的所有 UI 元素都是合在一个 Mesh 中的，过大的 Mesh 在更新时开销很大，所以一般建议每个较复杂的 UI 界面，都自成一个 Canvas(可以是子 Canvas)，在 UI 界面很复杂时，甚至要划分更多的子 Canvas。同时还要注意动态元素和静态元素

的分离，因为动态元素会导致 Canvas 的 mesh 的更新。最后，Canvas 又不能细分的太多，因为会导致 Draw Call 的上升。我们后续将对 UI 模块做具体的讲解，尽请期待。

3. UWA 性能检测报告中的 Shared UI Mesh 表示什么呢?

   - Shared UI Mesh 是在 Unity 5.2 版本后 UGUI 系统维护的 UI Mesh。在以前的版本中，UGUI 会为每一个 Canvas 维护一个 Mesh（名为 BatchedMesh，其中再按材质分为不同的 SubMesh）。而在 Unity 5.2 版本后，UGUI 底层引入了多线程机制，而其 Mesh 的维护也发生了改变，目前 Shared UI Mesh 作为静态全局变量，由底层直接维护，其大小与当前场景中所有激活的 UI 元素所生成的网格数相关。
   - 一般来说当界面上 UI 元素较多，或者文字较多时该值都会较高，在使用 UI/Effect/shadow 和 UI/Effect/Outline 时需要注意该值，因为这两个 Effect 会明显增加文字所带来的网格数。

4. 在使用 NGUI 时，我们通常会将很多小图打成一个大的图集，以优化内存和 Draw Call。而在 UGUI 时代，UI 所使用的 Image 必须是 Sprite；Unity 提供了 SpritePacker。它的工作流程和 UGUI Atlas Paker 有较大的差别。在 Unity Asset 中，我们压根看不到图集的存在。问题是:

   - 1. SpritePacker 大概的工作机制是什么样的?
   - 2. 如果 Sprite 没有打包成 AssetBundle，直接在 GameObject 上引用，那么在 Build 时 Unity 会将分散的 Sprite 拼接在一起么？如果没有拼接，那 SpritePacker 是不是只会优化 Draw Call，内存占用上和不用 SpritePacker 的分离图效果一样?
   - 3. 如果将 Sprite 打成 AssetBundle，AssetBundle 中的资源是分散的 Sprite 吗？如果不是，不同的 AssetBundle 中引用了两张 Sprite，这两张 Sprite 恰好用 SpritePacker 拼在了一起，是不是就会存在两份拼接的 Sprite 集?
   - 4. 如果想使用 NGUI Atlas Packer 的工作流程，该如何去实现?
     - 简单来说，UGUI 和 NGUI 类似，但是更加自动化。只需要通过设定 Packing Tag 即可指定哪些 Sprite 放在同一个 Atlas 下。
       * 可以通过 Edit -> Project Settings -> Editor -> Sprite Packer 的 Mode 来设置是否起效，何时起效（一种是进入 Play Mode 就生效，一种是 Build 时才生效）。所以只要不选 Disabled，Build 时就会把分散的 Sprite 拼起来。
       * 可以认为 Sprite 就是一个壳子，实际上本身不包含纹理资源，所以打包的时候会把 Atlas 打进去。如果不用依赖打包，那么分开打两个 Sprite 就意味各自的 AssetBundle 里都会有一个 Atlas。
       * 可以通过第三方工具（如 Texture Packer）制作 Atlas，导出 Sprite 信息（如，第 N 个 Sprite 的 Offset 和 Width，Height 等），然后在 Unity 中通过脚本将该 Atlas 转成一个 Multiple Mode 的 Sprite 纹理（即一张纹理上包含了多个 Sprite），同时禁用 Unity 的 Sprite Packer 即可。
     - 两种做法各有利弊，建议分析一下两种做法对于自身项目的合适程度来进行选择。

5. 在 Unity 5.x 版本下，我们在用 UGUI 的过程中发现它把图集都打进了包里，这样就不能自动更新了，请问图集怎么做自动更新呢?

   - 在 Unity 5.x 中 UGUI 使用的 Atlas 确实是不可见的，因此无法直接将其独立打包。但我们建议，可以把 Packing Tag 相同的源纹理文件，打到同一个 AssetBundle 中（设置一样的 AssetBundle Name），从而避免 Atlas 的冗余。同时这样打包可以让依赖它的 Canvas 的打包更加自由，即不需要把依赖它的 Canvas 都打在一个 AssetBundle 中，在更新时直接更新 Atlas 所在的 AssetBundle 即可。

6. ScrollRect 在滚动的时候，会产生 Canvas.SendwillRenderCanvases，有办法消除吗?

   - ScrollRect 在滚动时，会产生 OnTransformChanged 的开销，这是 UI 元素在移动时触发的，但通常这不会触发 Canvas.SendWillRenderCanvases。
   - 如果观察到 Canvas.SendWillRenderCanvases 耗时较高，可以检查下 ScrollRect 所在的 Canvas 是否开启了 Pixel Perfect 的选项，该选项的开启会导致 UI 元素在发生位移时，其长宽会被进行微调（为了对其像素），而 ScrollRect 中通常有较多的 UI 元素，从而产生较高的 Canvas.SendWillRenderCanvases 开销。因此可以尝试关闭 Pixel Perfect 看效果是否可以接受，或者尝试在滚动过程中暂时关闭 Pixel Perfect 等方式来消除其开销。

1.2.2 网格重建

1. 我在 UGUI 里更改了 Image 的 Color 属性，那么 Canvas 是否会重建？我只想借用它的 Color 做 Animation 里的变化量。

   - 如果修改的是 Image 组件上的 Color 属性，其原理是修改顶点色，因此是会引起网格的 Rebuild 的（即 Canvas.BuildBatch 操作，同时也会有 Canvas.SendWillRenderCanvases 的开销）。而通过修改顶点色来实现 UI 元素变色的好处在于，修改顶点色可以保证其材质不变，因此不会产生额外的 Draw Call。

2. Unity 自带的 UI Shader 处理颜色时，改 $_{Color}$？

   - 在 UI 的默认 Shader 中存在一个 Tint Color 的变量，正常情况下，该值为常数 (1,1,1)，且并不会被修改。如果是用脚本访问 Image 的 Material，并修改其上的 Tint Color 属性时，对 UI 元素产生的网格信息并没有影响，因此就不会引起网格的 Rebuild。但这样做因为修改了材质，所以会增加一个 Draw Call。

3. 能否就 UGUI Batch 提出一些建议呢？是否有一些 Batch 的规则？

   - 在 UGUI 中，Batch 是以 Canvas 为单位的，即在同一个 Canvas 下的 UI 元素最终都会被 Batch 到同一个 Mesh 中。而在 Batch 前，UGUI 会根据这些 UI 元素的材质（通常就是 Atlas）以及渲染顺序进行重排，在不改变渲染结果的前提下，尽可能将相同材质的 UI 元素合并在同一个 SubMesh 中，从而把 DrawCall 降到最低。而 Batch 的操作只会在 UI 元素发生变化时才进行，且合成的 Mesh 越大，操作的耗时也就越大。
   - 因此，我们建议尽可能把频繁变化（位置，颜色，长宽等）的 UI 元素从复杂的 Canvas 中分离出来，从而避免复杂的 Canvas 频繁重建。

4. 我用的是 UGUI Canvas，Unity 5.3.4 版本，请问如何查看每次 Rebuild Batch 影响的顶点数，Memory Profiler 是个办法但是不好定位。

   - 由于 Unity 引擎在 5.2 后开始使用 Shared UI Mesh 来存储 UI Mesh，所以确实很难查看每次 Rebuild 的 UI 顶点数。但是，研发团队可以尝试通过 Frame Debugger 工具对 UI 界面进行进一步的查看。

5. 动静分离或者多 Canvas 带来性能提升的理论基础是什么呢？如果静态部分不变动，整个 Canvas 就不刷新了？

   - 在 UGUI 中，网格的更新或重建（为了尽可能合并 UI 部分的 DrawCall）是以 Canvas 为单位的，且只在其中的 UI 元素发生变动（位置、颜色等）时才会进行。因此，将动态 UI 元素与静态 UI 元素分离后，可以将动态 UI 元素的变化所引起的网格更新或重建所涉及到的范围变小，从而降低一定的开销。而静态 UI 元素所在的 Canvas 则不会出现网格更新和重建的开销。

6. UWA 建议“尽可能将静态 UI 元素和频繁变化的动态 UI 元素分开，存放于不同的 Panel 下。同时，对于不同频率的动态元素也建议存放于不同的 Panel 中。”那么请问，如果把特效放在 Panel 里面，需要把特效拆到动态的里面吗？

   - 通常特效是指粒子系统，而粒子系统的渲染和 UI 是独立的，仅能通过 Render Order 来改变两者的渲染顺序，而粒子系统的变化并不会引起 UI 部分的重建，因此特效的放置并没有特殊的要求。

7. 多人同屏的时候，人物移动会使得头顶上的名字 Mesh 重组，从而导致较为严重的卡顿，请问一下是否有优化的办法？

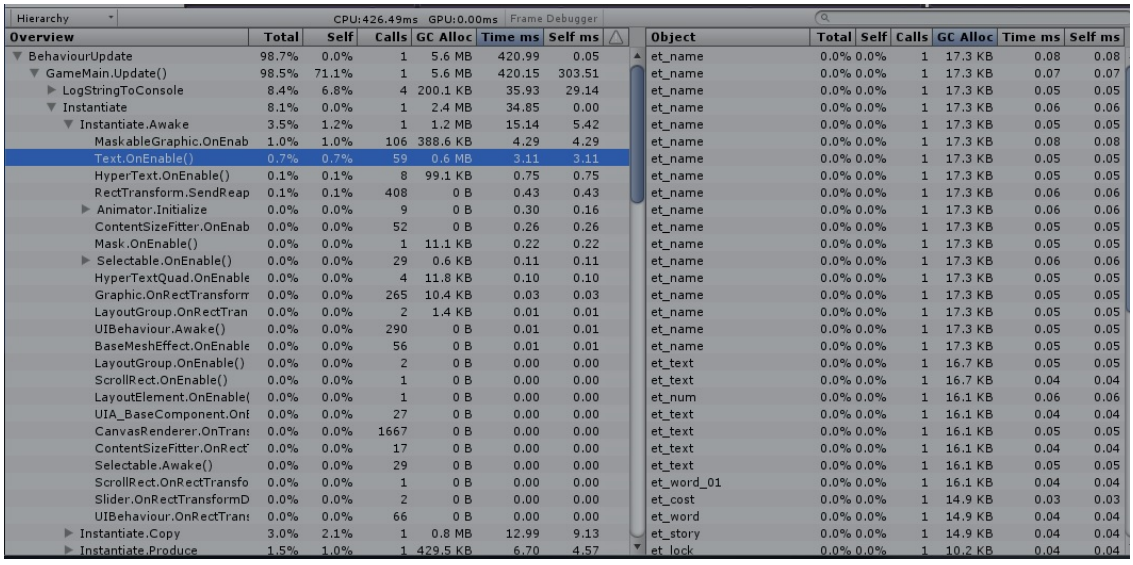
   - 如果是用 UGUI 开发的，当头顶文字数量较多时，确实很容易引起性能问题，可以考虑从以下几点入手进行优化：
     - 尽可能避免使用 UI/Effect，特别是 Outline，会使得文本的 Mesh 增加 4 倍，导致 UI 重建开销明显增大；
     - 拆分 Canvas，将屏幕中所有的头顶文字进行分组，放在不同的 Canvas 下，一方面可以降低更新的频率（如果分组中没有文字移动，该组就不会重建），另一方面可以减小重建时涉及到的 Mesh 大小（重建是以 Canvas 为单位进行的）；
     - 降低移动中的文字的更新频率，可以考虑在文字移动的距离超过一个阈值时才真正进行位移，从而可以从概率上降低 Canvas 更新的频率。

1.2.3 界面切换

1. 游戏中出现 UI 界面重叠，该怎么处理较好？比如当前有一个全屏显示的 UI 界面，点其中一个按钮会再起一个全屏界面，并把第一个 UI 界面盖住。我现在的做法是把被覆盖的界面 SetActive(False)，但发现后续 SetActive(True) 的时候会有 GC.Alloc 产生。这种情况下，希望既降低 Batches 又降低 GC Alloc 的话，有什么推荐的方案吗？

   - 可以尝试通过添加一个 Layer 如 OutUI，且在 Camera 的 Culling Mask 中将其取消勾选（即不渲染该 Layer）。从而在 UI 界面切换时，直接通过修改 Canvas 的 Layer 来实现“隐藏”。但需要注意事件的屏蔽，禁用动态的 UI 元素等等。
   - 这种做法的优点在于切换时基本没有开销，也不会产生多余的 Draw Call，但缺点在于“隐藏时”依然还会有一定的持续开销（通常不太大），而其对应的 Mesh 也会始终存在于内存中（通常也不太大）。
   - 以上的方式可供参考，而性能影响依旧是需要视具体情况而定。

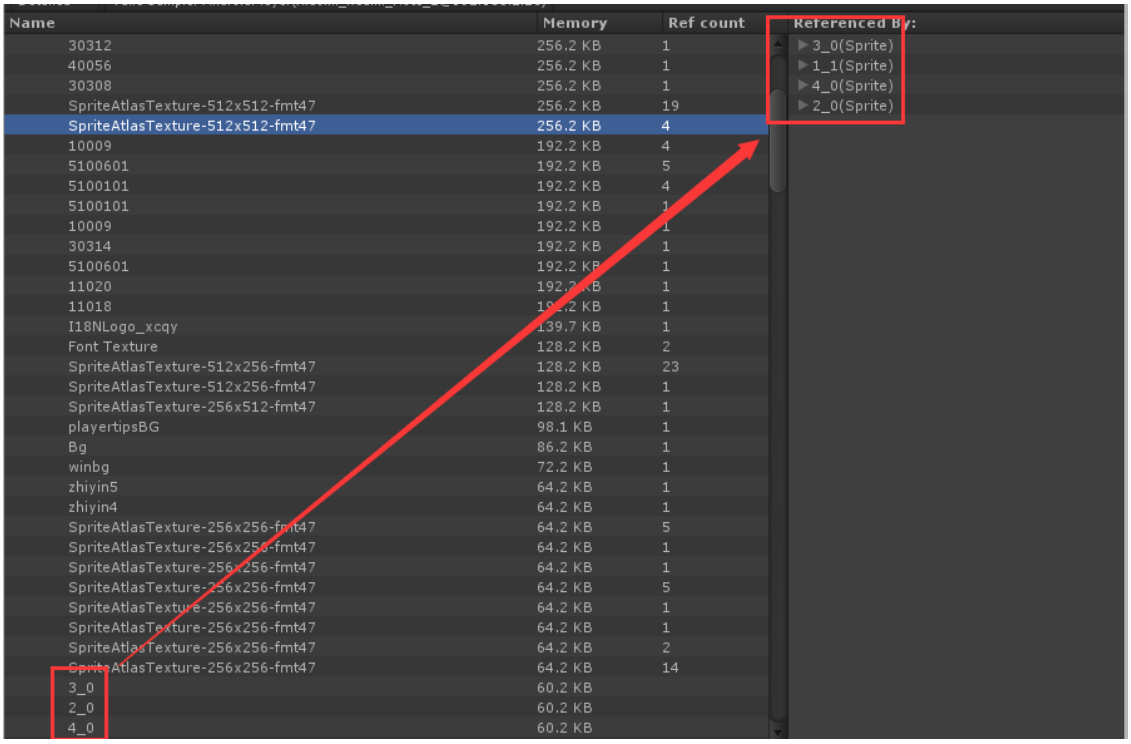
2. 如图，我们在 UI 打开或者移动到某处的时候经常会观测到 CPU 上的冲激，经过进一步观察发现是因为 Instantiate 产生了大量的 GC。想请问下 Instantiate 是否应该产生 GC 呢？我们能否通过资源制作上的调整来避免这样的 GC 呢？如下图，因为一次性产生若干 MB 的 GC 在直观感受上还是很可观的。



   - 准确的说这些 GC Alloc 并不是由 Instantiate 直接引起的，而是因为被实例化出来的组件会进行 OnEnable 操作，而在 OnEnable 操作中产生了 GC，比如以上图中的函数为例：
   - 上图中的 Text.OnEnable 是在实例化一个 UI 界面时，UI 中的文本（即 Text 组件）进行了 OnEnable 操作，其中主要是初始化文本网格的信息（每个文字所在的网格顶点，UV，顶点色等等属性），而这些信息都是储存在数组中（即堆内存中），所以文本越多，堆内存开销越大。但这是不可避免的，只能尽量减少出现次数。
   - 因此，我们不建议通过 Instantiate/Destroy 来处理切换频繁的 UI 界面，而是通过 SetActive(true/false)，甚至是直接移动 UI 的方式，以避免反复地造成堆内存开销。

1.2.4 加载相关

1. UGUI 的图集操作中我们有这么一个问题，加载完一张图集后，使用这个方式获取其中一张图的信息：assetBundle.Load (subFile, typeof (Sprite)) as Sprite; 这样会复制出一个新贴图（图集中的子图），不知道有什么办法可以不用复制新的子图，而是直接使用图集资源。



- 经过测试，这确实是 Unity 在 4.x 版本中的一个缺陷，理论上这张“新贴图（图集中的子图）”是不需要的，并不应该加载。因此，我们建议通过以下方法来绕过该问题：
- 在 assetBundle.Load (subFile, typeof (Sprite)) as Sprite; 之后，调用
- Texture2D t = assetBundle.Load (subFile, typeof (Texture2D)) as Texture2D;
- Resources.UnloadAsset(t);
- 从而卸载这部分多余的内存。

2. 加载 UI 预制的时候，如果把特效放到预制里，会导致加载非常耗时。怎么优化这个加载时间呢？

- UI 和特效（粒子系统）的加载开销在多数项目中都占据较高的 CPU 耗时。UI 界面的实例化和加载耗时主要由以下几个方面构成：
- 纹理资源加载耗时
  - UI 界面加载的主要耗时开销，因为在其资源加载过程中，时常伴有大量较大分辨率的 Atlas 纹理加载，我们在之前的 Unity 加载模块深度分析之纹理篇有详细讲解。对此，我们建议研发团队在美术质量允许的情况下，尽可能对 UI 纹理进行简化，从而加快 UI 界面的加载效率。

资源名称	生命周期(场景数)	内存占用	数量峰值	高度	宽度	格式	Mipmap数量
Lightmap-4_comp_light	2	4.0 MB	1	1024	1024	RGB24	11
Lightmap-1_comp_light	2	4.0 MB	1	1024	1024	RGB24	11
Atlas_Alpha1	2	2.7 MB	1	1024	1024	RGBA4444	11
Ground1_B	2	1.3 MB	1	1024	1024	ETC_RGB4	11
Ground6_B	2	1.3 MB	1	1024	1024	ETC_RGB4	11
ground3	2	1.3 MB	1	1024	1024	ETC_RGB4	11
Ground5	2	1.3 MB	1	1024	1024	ETC_RGB4	11
stamps	2	1.3 MB	1	1024	1024	ETC2_RGBA8	11
Atlas_Nature1	2	682.8 KB	1	1024	1024	ETC_RGB4	11
Atlas_Nature1_nmp	2	682.8 KB	1	1024	1024	ETC_RGB4	11

清除选中

« ‹ 1 2 3 4 5 … 21 › »

- UI 网格重建耗时
  - UI 界面在实例化或 Active 时，往往会造成 Canvas（UGUI）或 Panel（NGUI）中 UIDrawCall 的变化，进而触发网格重建操作。当 Canvas 或 Panel 中网格量较大时，其重建开销也会随之较大。

- UI 相关构造函数和初始化操作开销
  - 这部分是指 UI 底层类在实例化时的 ctor 开销，以及 OnEnable 和 OnDisable 的自身开销。
- 上述 2 和 3 主要为引擎或插件的自身逻辑开销，因此，我们应该尽可能避免或降低这两个操作的发生频率。我们的建议如下：
  - 在内存允许的情况下，对于 UI 界面进行缓存。尽可能减少 UI 界面相关资源的重复加载以及相关类的重复初始化；
  - 根据 UI 界面的使用频率，使用更为合适的切换方式。比如移进移出或使用 Culling Layer 来实现 UI 界面的切换效果等，从而降低 UI 界面的加载耗时，提升切换的流畅度。
  - 对于特效（特别是粒子特效）来说，我们暂时并没有发现将 UI 界面和特效耦合在一起，其加载耗时会大于二者分别加载的耗时总和。因此，我们仅从优化粒子系统加载效率的角度来回答这个问题。粒子系统的加载开销，就目前来看，主要和其本身组件的反序列化耗时和加载数量相关。对于反序列化耗时而言，这是 Unity 引擎负责粒子系统的自身加载开销，开发者可以控制的空间并不大。对于加载数量，则是开发者需要密切关注的，因为在我们目前看到的项目中，不少都存在大量的粒子系统加载，有些项目的数量甚至超过 1000 个，如下图所示。因此，建议研发团队密切关注自身项目中粒子系统的数量使用情况。一般来说，建议我们建议粒子系统使用数量的峰值控制在 400 以下。

3. 我有一个 UI 预设，它使用了一个图集，我在打包的时候把图集和 UI 一起打成了 AssetBundle。我在加载生成了 GameObject 后立刻卸载了 AssetBundle 对象，但是当我后面再销毁 GameObject 的时候发现图集依然存在，这是什么情况呢？

   - 这是很可能出现的。unload(false) 卸载 AssetBundle 并不会销毁其加载的资源，是必须对其调用 Resources.UnloadAsset，或者调用 Resources.UnloadUnusedAssets 才行。关于 AssetBundle 加载的详细解释可以参考我们之前的文章：你应该知道的 AssetBundle 管理机制。

1.2.5 字体相关

1. 我在用 Profiler 真机查看 iPhone App 时，发现第一次打开某些 UI 时，Font.CacheFontForText 占用时间超过 2s，这块主要是由什么影响的？若 iPhone5 在这个接口消耗 2s 多，是不是问题很大？这个消耗和已经生成的 RenderTexture 的大小有关吗？

   - Font.CacheFontForText 主要是指生成动态字体 Font Texture 的开销，一次性打开 UI 界面中的文字越多，其开销越大。如果该项占用时间超过 2s，那么确实是挺大的，这个消耗也与已经生成的 Font Texture 有关系。简单来说，它主要是看目前 Font Texture 中是否有地方可以容下接下来的文字，如果容不下才会进行一步扩大 Font Texture，从而造成了性能开销。

1.3 NGUI 与 UGUI 最详细对比

- http://www.u3dc.com/archives/412
- 1.ugui 的 ui 根目录为 canvas（画布），ngui 则是 uiroot。在命名上官方似乎更贴合想象力。
- 2. 在屏幕自适应方面，ugui 为 render mode。ngui 则为 scaling style。
- 3.anchor（锚点）的使用方式差不多，都是用来固定位置，在可视化方面，ugui 的花瓣锚点真不太好调。
- 4.ngui 灵活性不是一般的高，随意创建一个 sprite，加了 boxcollider，它就可以是按钮、滑动条……

- 5.ugui 的 sprite 的切图功能真心不错。ngui 使用图集不能直接拖拉（毕竟是三方插件）略不方便。
- 6.ngui 的 tween 动画功能很省心，无需额外定义代码，使用封装好的脚本就可以实现一些简单动画，叠加脚本甚至能实现相对复杂的动画效果。
- 最后，强大的网友分享了一张比较全面的对比图（点击图片放大）：

A	B	C	D
	ngui与ugui优缺点及性能比较		
	ngui	**ugui**	**对比结果**
同一界面性能对比	1.drawcall:7-9 2.Used Textures:16.3 MB 3.VRAM usage :11.4MB-25.4MB 4.发布后APK大小:19.8MB	1.drawcall:11-13 2.Used Textures:13.5 MB 3.VRAM usage :9.0MB-22.6MB 4.发布后APK大小:18.5MB	同样界面(其中图片资源约为1MB),ugui的DrawCall高2个.但是内存等其他 明显减小.安装包也小1.3MB
自适应	1.PixelPerfect 完美像素：直接显示设定好的像素。当屏幕高度低于minimum Height时按比例缩小，当屏幕高度大于maximum Height时按比例扩大。 2.FixedSize 按比例缩放：在设定好的基础上，直接按比例缩放。 3.FixedSizeOnMobiles 合体版，android和ios为FixedSize方式，其它按照PixelPerfect方式。	1.有三种renderMode： overlay：始终在最前面并充满屏幕，不可设置大小。可设置canvas层级显示的顺序 camara ：始终充满屏幕可设置摄像机，不可设置大小，不一定在最前可设置层级显示的顺序。 worldSpace：可设置大小，可设置层级。可用于人物血条等的制作	ugui的更好用
text文本	1.若需要的中文字体较少，可以减少最终安装包的大小。 2.对中文支持不友好，需要用fontMaker，BMFont等工具制作成图集，比较麻烦。 3.无法支持动态字体的图文混排。	1.可直接导入ttf字体，无需制作成图集。支持富文本格式，方面易用。 2.支持动态字体的图文混排。 3.若需要的中文字体比较少，则会加大最终安装包的大小 4.图文混编需要自己编写	ugui的性能更强大,对动态字体中文等支持更好,但是有些功能需要自己写组件
图集方面	1.使用图片前必须要打成图集. 2.可设置图集大小	1.可直接使用小图..发布时自动打成图集.或者你也可以自己设置图集 2.可设置图集大小	ugui的更友好,但是无法在一个图集中动态获取其中的一个小图
image		可以使用material	无太大区别.
深度排序	1、相同panel 相同atlas sprite受depth控制 2、相同panel 不同atlas sprite受z轴控制(同时受1的影响) 3、不同panel 相同atlas sprite受z轴控制(同时受1的影响) 4、不同panel 不同atlas sprite受z轴控制(同时受1的影响)	在Hieraychy视图中,越在下方显示越靠前,可用SetSiblingIndex 和GetSiblingIndex等来设置深度	ngui更复杂
事件系统			无太大区别.都可使用委托等
Tween缓动	有常用的一些缓动效果.	无	ngui自带缓动.ugui需用插件.如dotween.itween等