


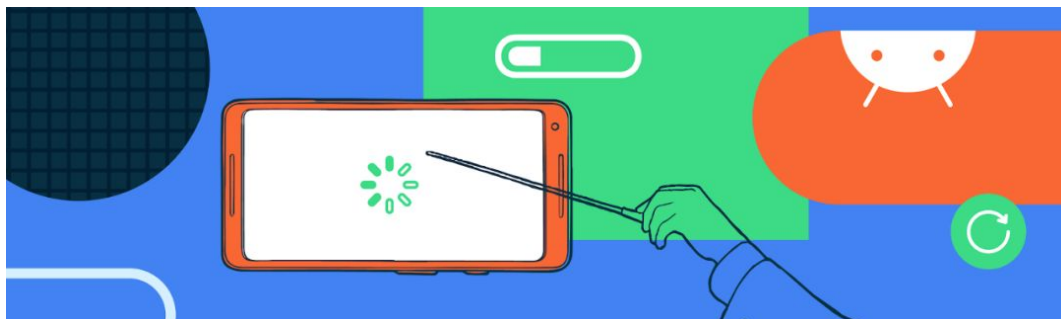
在 Android 上进行高刷新率渲染



谷歌开发者 
已认证的官方帐号

已关注

77 人赞同了该文章



作者 / Ady Abraham, Software Engineer

长久以来，手机屏幕刷新率都是 60Hz。应用和游戏开发者也习惯了假定刷新率为 60Hz，也就是每 16.6ms 生成一帧，而且这样开发出来的应用和游戏都会正常进行。但现在的情况已经不同了。最新的旗舰级设备往往会搭载刷新率更高的屏幕，可以带来更流畅的动画效果、更低的延迟，从而获得更好的整体用户体验。还有一些设备支持可变刷新率，比如 Pixel 4，它支持 60Hz 和 90Hz 两种刷新率。

60Hz 的屏幕每 16.6ms 刷新一次显示内容。这意味着图像显示的时间是 16.6ms 的倍数 (16.6ms、33.3ms、50ms 等)。支持多种刷新率的屏幕则带来了更多的选择，这些屏幕能以不同的速度进行渲染，并且不会出现抖动。例如，一个无法维持 60fps 渲染的游戏，在 60Hz 的屏幕上必须一路降到 30fps 才能确保流畅无抖动 (因为显示器只能以 16.6ms 的倍数周期呈现图像，所以 60Hz 的下一档可用帧速是每 33.3ms 显示一帧，即 30fps)。而在 90Hz 设备上，同样的游戏只需要下降到 45fps (每帧 22.2ms) 即可，这就为用户带来了更流畅的体验。而同时支持 90Hz 和 120Hz 的设备，则可以用每秒 120、90、60 (120/2)、45 (90/2)、40 (120/3)、30 (90/3)、24 (120/5) 等帧率流畅地呈现内容。

高频率渲染

渲染频率越高，就越难维持帧率，因为只有更少的时间完成相同的工作量。要在 90Hz 下进行渲染，应用需要在 11.1ms 内生成一帧，与此相比，在 60Hz 时则有 16.6ms 来生成一帧。

为了详细说明这一点，我们来看看 Android UI 的渲染流水线。我们可以将帧渲染大致分为五个流水线阶段：

1. 应用的 UI 线程处理输入事件，调用应用的回调，并更新视图 (View) 层次结构中记录的绘图命令列表；
2. 应用的 RenderThread 将记录的命令发送到 GPU；
3. GPU 绘制这一帧；
4. SurfaceFlinger 是负责在屏幕上显示不同应用窗口的系统服务，它会组合出屏幕应该最终显示出的内容，并将画面提交给屏幕的硬件抽象层 (HAL)；
5. 屏幕最终呈现该帧的内容。

整个流水线由 Android Choreographer 控制。Choreographer 基于显示垂直同步 (vsync) 事件，它表示屏幕开始扫描出图像并更新显示像素的时间点。虽然 Choreographer 基于 vsync 事件，但对应用和 SurfaceFlinger 来说，其唤醒偏移量不同。下图展示了在 Pixel 4 设备上运行的流水线，应用在 vsync 事件后 2ms 被唤醒，SurfaceFlinger 则在 vsync 事件后 6ms 被唤醒。这样一来，应用产生一帧画面的时间为 20ms，SurfaceFlinger 组合画面内容的时间则为 10ms。

当以 90Hz 频率运行时，应用依然在 vsync 事件后 2ms 被唤醒。然而，SurfaceFlinger 在 vsync 事件后 1ms 被唤醒，同样有 10ms 的时间来合成屏幕内容。但这样一来应用只有 10ms 来渲染一帧画面，这时间就非常窘迫了：

为了缓解这种情况，Android 的 UI 子系统采用了预先渲染 (render ahead，指维持一帧的启动时间不变，但推迟其呈现时间) 来深化流水线，并将帧的呈现时间推迟一个 vsync。这样一来，应用可以有 21ms 的时间来生成一帧，同时确保维持 90Hz 的吞吐量。

一些应用，包括大多数游戏，都有自己自定义的渲染流水线。这些流水线可能会有更多或更少的阶段，具体取决于它们要完成的任务。一般来说，流水线越深，可以并行执行的阶段就越多，整体的吞吐量也会相应增加。但另一方面，这样可能会增加单帧的延迟 (延迟量为 $\text{number_of_pipeline_stages} \times \text{longest_pipeline_stage}$)。这中间如何取舍需要开发者审慎考虑。

利用可变刷新率

如上所述，可变刷新率允许我们使用更多样的渲染频率。对于可以控制渲染速度的游戏，以及需要以特定速率呈现内容的视频播放器来说，这一点尤其有用。例如，要在 60Hz 的显示器上播放 24fps 的视频，我们需要使用 3:2 pulldown 算法，这就会产生抖动。但是，如果设备的屏幕可以原生显示 24fps 的内容 (24/48/72/120Hz)，就无需使用 pulldown 算法，自然也就不会出现抖动了。

设备运行时的刷新率是由 Android 平台控制的。应用和游戏可以通过多种方法影响刷新率 (下面会有解释)，但最终结果由平台决定。尤其是当屏幕上同时有多个应用时，这一点至关重要：平台需要满足所有应用的刷新率需求。24fps 视频播放器就是一个很好的例子。24Hz 对于视频播放来说可能很好，但对于响应式 UI 来说就很糟糕了。如果一个推送通知的动画只有 24Hz，感觉就会很扎眼。在这种情况下，平台会选择让屏幕上的内容都显示良好的刷新率。

为此，应用可能需要知道当前设备的刷新率。可以通过以下方法来实现：

- SDK
 - 通过 DisplayManager.DisplayListener 注册一个显示监听器，并通过 Display.getRefreshRate 查询刷新率。
- NDK
 - 使用 android.hardware.display.RefreshRate 注册回调 (API 级别 30)



应用可以通过在其 Window 或 Surface 上设置帧率来影响设备刷新率。这是 Android 11 中引入的一个新功能，允许平台了解应用的渲染需求。应用可以调用以下方法之一：

- SDK
 - `Surface.setFrameRate`
[developer.android.google.cn...](https://developer.android.google.cn/...)
 - `SurfaceControl.Transaction.setFrameRate`
[developer.android.google.cn...](https://developer.android.google.cn/...)
- NDK
 - `ANativeWindow_setFrameRate`
[developer.android.google.cn...](https://developer.android.google.cn/...)
 - `ASurfaceTransaction_setFrameRate`

[Native Activity](#) | [Android NDK](#) | [Android Developers](#)

关于如何使用这些 API，请参考[帧率指南文档](#)。

系统会根据 Window 或 Surface 上设置的帧率选择最合适的刷新率。

在较旧的 Android 版本 (Android 11 之前) 中并不存在 `setFrameRate` API，这时应用仍然可以通过直接将 `WindowManager.LayoutParams.preferredDisplayModeId` 设置为 [Display.getSupportedModes](#) 中的可用模式之一来影响刷新率。从 Android 11 开始，我们不建议大家采用这种方法，因为平台会不知道应用的渲染意图。例如，如果一个设备支持 48Hz、60Hz 和 120Hz，屏幕上有两个应用分别调用 `setFrameRate(60, ...)` 和 `setFrameRate(24, ...)`，那么平台可以选择 120Hz 来同时满足这两个应用。而如果这些应用使用了 `preferredDisplayModeId`，它们很可能把模式设置为 60Hz 和 48Hz，那这时平台就无法使用 120Hz 了。这时平台只能从 60Hz 或 48Hz 中选择一个，从而影响到另一个应用的显示效果。

总结

刷新率不一定是 60Hz——不要想当然地认为它一定会是 60Hz，也不要基于历史经验作出硬性假设。

刷新率并不总是恒定的——如果您想了解实际的刷新率，就需要注册一个回调来知晓刷新率的变动，并相应地更新您应用内部的数据。

如果您没有使用 Android UI 工具包，而使用自定义的渲染器，请考虑根据当前的刷新率来改变您的渲染流水线。通过使用 OpenGL 上的 `eglPresentationTimeANDROID` 或 Vulkan 上的 `VkPresentationTimesInfoGOOGLE` 设置一个呈现时间戳，即可深化流水线。设置呈现时间戳可以向 `SurfaceFlinger` 指示何时呈现图像。如果设置为未来的几帧，它就会按照设置的帧数加深流水线。前文例子中的 Android UI 将呈现时间设置成了 `frameTimeNanos + 2 * vsyncPeriod`。

注: `frameTimeNanos` 从 `Choreographer` 获取; `vsyncPeriod` 从 `Display.getRefreshRate()` 获取。

使用 `setFrameRate` API 告诉平台您的渲染意图，平台会选择合适的刷新率来匹配不同的需求。

您应该只在必要时才使用 `preferredDisplayModeId`: 当 `setFrameRate` API 不可用时，或是当您需要使用非常特定的模式时。

最后，请您深入了解一下[Android 的帧同步库](#)。这个库可以为您的游戏妥善处理帧同步，并使用前文中的方法来处理多种刷新率。

[点击这里](#)即刻前往 Android 11 开发者网站



发布于 05-20

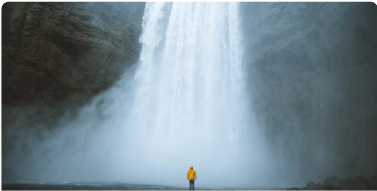
[Android](#) [Android 开发](#)

推荐阅读



新的流畅体验，90Hz 漫谈

高爷 发表于Andro...



Android 基于 Choreographer 的渲染机制详解

高爷 发表于Andro...

探索Android渲染

保持好奇心，保持激情，不被环境所同化这是大学毕业的时候，一位挚友、学长送我的一句话。转眼即将毕业两年，很庆幸自己能够一直践行着这句话，伴随着的是满满的收获与成长。更让我庆幸的是...

范宏伟 发表于Andro...



Android帧率、卡顿详解及使用

犀利哥 发表于软件测试杂...

13 条评论

切换为时间排序

写下你的评论...

- 

北极熊

05-20

净整那没用的，安卓垃圾的底层，到现在都无法高效实现动态高斯模糊，刷新率再高有个屁用

24
- 

Tabjy 回复 北极熊

05-20

牛逼啊😂

2
- 

peerless2012 回复 北极熊

05-20

来自开发者的吐槽[大笑][大笑][大笑]

2
- 查看全部 10 条回复



在三星Note8上试了下WindowManager.LayoutParams.preferredDisplayModeId这个api，赋值了个720p的显示id，结果整个系统都成720p了还只占用了屏幕的一部分，应用退出了也不行，最后只能重启了



👍 2



Macro

06-18

请打开语音交流

👍 赞

1 条评论被折叠 ([为什么?](#))