android vsync机制

Last edited by **caoquanli** 1 month ago

屏幕是由许多的像素点组成的，每个像素点通过显示不同的颜色最终屏幕呈现各种各样的图像。手机系统的类型和手机硬件的不同导致UI的流畅性体验不一致

屏幕展示的颜色数据

- 在GPU中有一块缓冲区叫做 Frame Buffer ,这个帧缓冲区可以认为是存储像素值的二位数组。数组中的每一个值就对应了手机屏幕的像素点需要显示的颜色。
- 由于这个帧缓冲区的数值是在不断变化的,所以只要完成对屏幕的刷新就可以显示不同的图像了.。 至于刷新工作手记的逻辑电路会定期的刷新 Frame Buffer的 目前主流的刷新频率为60次/秒 折算出来就是16ms刷新一次

GPU的Frame Buffer中的数据

- GPU 除了帧缓冲区用以交给手机屏幕进行绘制外. 还有一个缓冲区 Back Buffer 这个用以交给应用的,让CPU往里面填充数据。
- GPU会定期交换 Back Buffer 和 Frame Buffer ， 也就是对Back Buffer中的数据进行栅格化后将其转到 Frame Buffer 然后交给屏幕进行显示绘制，同时让原先的Frame Buffer 变成 Back Buffer 让程序处理

Android的16m

Android中我们一般都会提到16ms绘制一次，那么到底是那里控制这16ms的呢?

在Choreographer类中我们有一个方法获取屏幕刷新速率:

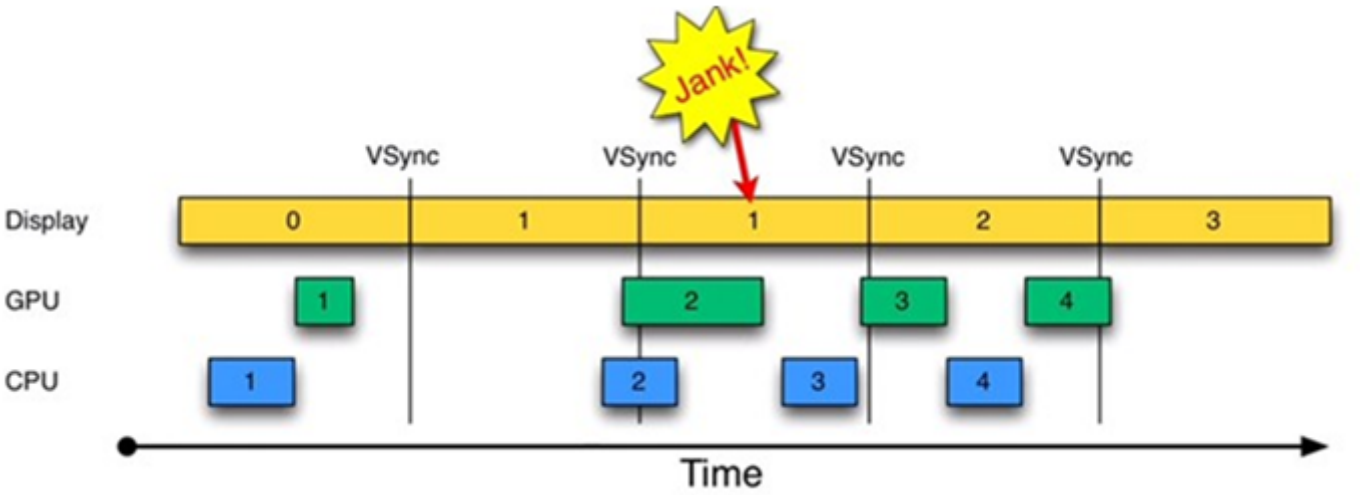
```
public final class Choreographer {
        private static float getRefreshRate() {
        DisplayInfo di = DisplayManagerGlobal.getInstance().getDisplayInfo(
                Display.DEFAULT_DISPLAY);
        return di.refreshRate;
    }
}
```

翻译来自android development中的原理解析

经典三缓冲机制

Project Butter对Android Display系统进行了重构，引入了三个核心元素，即VSYNC、Triple Buffer和Choreographer。其中， VSYNC是理解Project Buffer的核心。VSYNC是Vertical Synchronization（垂直同步）的缩写，是一种在PC上已经很早就广泛使用的技术.

首先是没有VSYNC的情况:



- 时间从0开始，进入第一个16ms：Display显示第0帧，CPU处理完第一帧后，GPU紧接其后处理继续第一帧。三者互不干扰，一切正常。
- 时间进入第二个16ms：因为早在上一个16ms时间内，第1帧已经由CPU，GPU处理完毕。故Display可以直接显示第1帧。显示没有问题。但在本16ms期间，CPU和GPU 却并未及时去绘制第2帧数据（注意前面的空白区），而是在本周期快结束时，CPU/GPU才去处理第2帧数据。
- 时间进入第3个16ms，此时Display应该显示第2帧数据，但由于CPU和GPU还没有处理完第2帧数据，故Display只能继续显示第一帧的数据，结果使得第1 帧多画了一次（对应时间段上标注了一个Jank）

- 通过上述分析可知，此处发生Jank的关键问题在于，为何第1个16ms段内，CPU/GPU没有及时处理第2帧数据？原因很简单，CPU可能是在忙别的事情（比如某个应用通过sleep 固定时间来实现动画的逐帧显示），不知道该到处理UI绘制的时间了。可CPU一旦想起来要去处理第2帧数据，时间又错过了！

为解决这个问题，Project Buffer引入了VSYNC，这类似于时钟中断。结果如图所示：

由图可知，每收到VSYNC中断，CPU就开始处理各帧数据。整个过程非常完美。 不过，仔细琢磨图2却会发现一个新问题：图中，CPU和GPU处理数据的速度似乎都能在16ms内完成，而且还有时间空余，也就是说，CPU/GPU的FPS（帧率，Frames Per Second）要高于Display的FPS。确实如此。由于CPU/GPU只在收到VSYNC时才开始数据处理，故它们的FPS被拉低到与Display的FPS相同。但这种处理并没有什么问题，因为Android设备的Display FPS一般是60，其对应的显示效果非常平滑。 如果CPU/GPU的FPS小于Display的FPS，会是什么情况呢？请看下图：

- 在第二个16ms时间段，Display本应显示B帧，但却因为GPU还在处理B帧，导致A帧被重复显示。
- 同理，在第二个16ms时间段内，CPU无所事事，因为A Buffer被Display在使用。B Buffer被GPU在使用。注意，一旦过了VSYNC时间点， CPU就不能被触发以处理绘制工作了。

为什么CPU不能在第二个16ms处开始绘制工作呢？原因就是只有两个Buffer。如果有第三个Buffer的存在，CPU就能直接使用它， 而不至于空闲。出于这一思路就引出了Triple Buffer。结果如图所示：

由图可知： 第二个16ms时间段，CPU使用C Buffer绘图。虽然还是会多显示A帧一次，但后续显示就比较顺畅了。 是不是Buffer越多越好呢？回答是否定的。由图4可知，在第二个时间段内，CPU绘制的第C帧数据要到第四个16ms才能显示， 这比双Buffer情况多了16ms延迟。所以，Buffer最好还是两个，三个足矣。

以上对VSYNC进行了理论分析，其实也引出了Project Buffer的三个关键点： 核心关键：需要VSYNC定时中断。 Triple Buffer：当双Buffer不够使用时，该系统可分配第三块Buffer。 另外，还有一个非常隐秘的关键点：即将绘制工作都统一到VSYNC时间点上。这就是Choreographer的作用。在它的统一指挥下，应用的绘制工作都将变得井井有条。