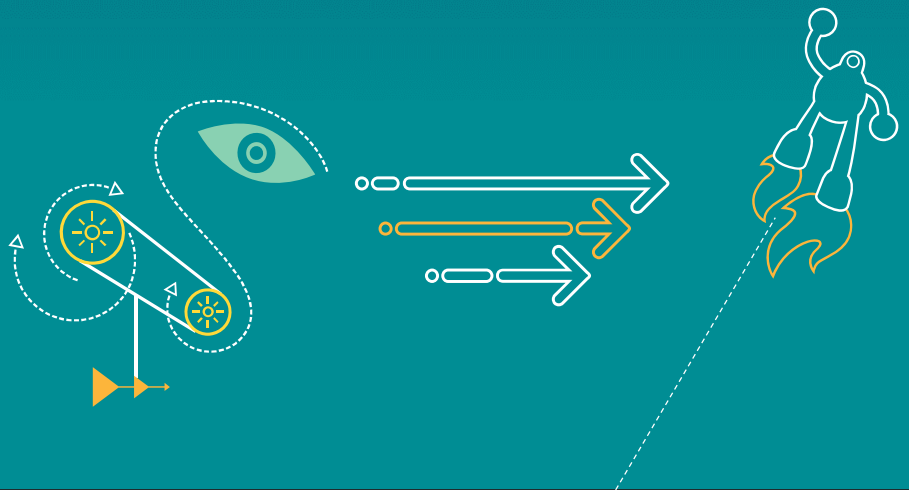
# 高通多媒体技术期刊　Graphics合辑 20150121

Qualcomm Technologies, Inc.





# Confidential and Proprietary – Qualcomm Technologies, Inc.

# Revision History

| Revision | Date | Description |
|---|---|---|
| A | Jan 2015 | Initial release |

**Note:** There is no Rev. I, O, Q, S, X, or Z per Mil. standards.

# Contents

# Android Graphics Overview

# Solution#:00029716 Graphics相关视频下载

- 2014年8月高通在分别在深圳和北京举行了大型的Graphics and display 深入培训，受到了广泛的好评。为了更广泛的帮助到更多的客户，我们陆续发布了相关的视频文件，Solution#:00029716 包含了现在已经发布的Graphics相关的视频，希望你尽早阅读
- VD80-NR299-1SC - Video: Adreno Debugging Overview  Training - Simplified Chinese https://downloads.cdmatech.com/qdc/drl/objectId/09010014829ffcd4：详细的解释了常见的Graphics相关的调试技巧和方法, 这是最重要的视频，希望客户多多阅读
- VD80-NP885-1SC - Video: Graphics Power Overview  Training - Simplified Chinese https://downloads.cdmatech.com/qdc/drl/objectId/09010014829f44e2：介绍了Graphics相关的功耗管理策略和调试技巧
- VD80-NP885-2SC - Video: Graphics Performance Overview  Training - Simplified Chinese https://downloads.cdmatech.com/qdc/drl/objectId/0901001482a0df9c：介绍了Graphics相关的性能分析和调试技巧
- VD80-NP884-1SC - Video: Android Graphics Overview Training - Simplified Chinese  https://downloads.cdmatech.com/qdc/drl/objectId/09010014829ffc70：介绍了Graphics相关的硬件模块，系统结构，软件结构
- 更多视频请阅读Solution#:00029716

# Graphics 软件框架和驱动结构介绍

- Graphics 软件框架包括基于Java 层的view类，hardwarerender类和基于native库的libui.so, libgui.so, libhwui.so, libskia.so。通常Android UI系统都是通过Java view类来实现的。如果APK使能了硬件加速，那么就会使用libhwui.so里面的OpenGL Render 去完成绘图。反之，则是通过libskia.so 的Software Render去完成绘图。
- 基于OpenGL ES的游戏可以直接加载OpenGL ES驱动，调用GLES/EGL API 进行游戏界面绘图。
- Graphics驱动结构包括：
  - 用户态驱动是高通闭源代码， 包括一些动态库文件

    libGLESv1_CM_adreno200.so : OpenGLES 1.1 Driver
    libGLESv2_adreno200.so : OpenGLES 2.0/3.0 Driver
    libEGL_adreno200.so : EGL 1.4 Driver
    eglsubAndroid.so : EGL Driver for Android Sub System
    libq3dtools_adreno200.so : Adreno Profiler support layer
    libgsl.so : GSL Library
    libC2D2.so : C2D Library
    libOpenCL.so : OpenCL Library
    libllvm-a3xx.so : LLVM Compiler Library for Adreno3xx GPU
    libllvm-arm.so : LLVM Compiler Library for CPU
    libsc-a3xx.so : Adreno 3xx Shader Compiler Library
  - 内核态驱动是开源代码，所有Kernel Graphics System Layer (KGSL)代码都在driver kernel/drivers/gpu/msm/ 目录下
- 请阅读http://source.android.com/devices/graphics/architecture.html 了解更多通用Android Graphics 软件框架

# Graphics 常见问题的分类

- **UI Corruption/Distortion/Mess**

  主要表现在UI界面的绘图不正确，有可能是局部区域内容花掉了，也可能字体出现毛刺，花掉等情况。通常需要从Hardware UI的角度进行调试，也有可能是Graphics 驱动的问题。

- **Native crash in graphics 用户态驱动**

  表现是应用进程异常退出，从logcat或者tombstone看到进程crash时的callstack/backtrace, 出错是在graphics 用户态驱动库里面。引起这类问题的原因可能是驱动代码的问题，也可能是应用不正确地调用OpenGL ES/EGL API造成, 还可能是memory corruption引起的。

- **GPU page fault issue**

  当GPU想要去访问一个非法的地址时，就会产生GPU page fault. 从kernel log里面能看到类似如下的信息：

  <2>[122867.712606] c1 81 kgsl kgsl-3d0: |kgsl_iommu_fault_handler| GPU PAGE FAULT: addr = 773E9B00 pid = 2770

  <2>[122867.712617] c1 81 kgsl kgsl-3d0: |kgsl_iommu_fault_handler| context = 0 FSR = 80000002 FSYNR0 = 582 FSYNR1 = 3C030008(read fault)

- **GPU hang issue**

  GPU hang 意味着GPU硬件已经停止正常工作，被挂起来了。如果你从kernel log里面看到如下信息，那么基本可以断定是一个GPU hang 发生了。

  [ 1144.957632 / 03-25 15:31:58.510] kgsl kgsl-3d0: ndroid.keyguard[1226]: gpu fault ctx 4 ts 9899 status E5678011 rb 0bca/0bca ib1 3fa8a900/03b5 ib2 3f66f000/0000

  你可以找到 “gpu fault” 关键字，后面紧跟着 “status”, “rb”, “ib1” and “ib2”等关键字。

  如果你看到 “time out” 关键字，那么这个就不是GPU hang，而是另外一种叫做Long IB的问题，表示GPU在执行某一个IB是花费的时间田长了，超过系统设定的门限。但是这时候GPU不一定是hang了，需要区别对待。GPU hang问题通常很难处理，我们需要客户提供完整的logcat log和kernel log，以及GPU snapshot，还需要提供Kgsl trace log.

# 通用的 Graphics 调试方法

针对/system/build.prop 属性的修改，可以采用以下3种方法:

1. 本地修改build.prop
    1. adb pull /system/build.prop  build.prop
    2. 本地修改
    3. adb root  && adb wait-for-device
    4. adb remount && adb wait-for-device
    5. adb push build.prop /system/build.prop
    6. adb shell chmod 644 /system/build.prop
    7. adb reboot

2. 通过命令行设置
    1. adb shell setprop property value
    2. adb shell stop
    3. adb shell start
    4. adb shell getprop property

3. 重新编译
    1. 直接修改build路径的system.prop,
    2. 重新build Images。

方法1和方法2在某些时候可能失效，这样只能用方法3。

# 通用的 Graphics 调试方法 – 续一

针对adreno_config.txt 属性的修改，可以采用如下2种方法:

1. 通过push新的adreno_config.txt 文件到调试手机上
   1. 创建一个空的文件，文件名为adreno_config.txt
   2. 在adreno_config.txt文件里面添加/编辑需要的属性
   3. adb push adreno_config.txt  /data/local/tmp/
   4. 然后重新启动，并需要确保/data/local/tmp/adreno_config.txt的访问属性，一般设置为 chmod 777 /data/local/tmp/adreno_config.txt

2. 也可以通过命令行，利用echo和重定向命令完成

```
adb root && adb wait-for-device
adb remount && adb wait-for-device
adb shell chmod  777 /data/local/tmp/
adb shell rm  /data/local/tmp/adreno_config.txt
adb shell "echo ‘disableTiledRendering=1’>/data/local/tmp/adreno_config.txt"
adb shell "echo ‘enableRotationShaderPatching=1’>>/data/local/tmp/adreno_config.txt"
adb reboot
```

# UI Corruption常用的调试三方法

此类问题通常跟HW UI 相关，因此第一步先从HW UI 层面调试。

1. Disable HWUI 的方法：

   如果有app源代码，那么在源代码路径下找到 AndroidManifest.xml file, 在标签 <application /> 修改如下属性的值为false:

   android:hardwareAccelerated="false"

   如果没有app源代码，那么只能把HWUI 从整个Framework层面disable.

   ```
   public static boolean isAvailable() {
      - return GLES20Canvas.isAvailable();
      +return false;
   }
   ```

   如果disable HW UI后问题不出现，则可以断定是HW UI 或者GLES/EGL driver的问题。否则可以证明是app软件的问题。

2. Disable HWUI 部分更新的方法：

   在/system/build.prop, 修改属性 hwui.render_dirty_regions = false， 此属性验证问题是否与HWUI 部分更新相关。

3. Disable TILE rendering 的方法:

   在/system/build.prop, 修改属性 debug.enabletr=false, 此属性验证问题是否与TILE rendering 相关。

   [note] 不过这个属性在最新的android KK 上面已经不用，TILE rendering默认是enabled，只能在GPU driver层面去diasble，具体方法在下一页介绍。

   还有一些跟HW UI相关的其他属性可以去验证：

   | | | |
   |---|---|---|
   | debug.hwui.disable_draw_defer | - | 是否与HWUI deferred draw 有关系 |
   | debug.hwui.disable_draw_reorder | - | 是否与HWUI reordered draw 有关系 |
   | ro.hwui.disable_scissor_opt | - | 是否与HWUI scissor 优化有关系 |

# UI Corruption常用的调试方法 – 续一

第二步需要在Adreno GPU driver 层面调试。

在adreno_config.txt里面定义很多调试开关，用来调试Adreno用户态的驱动程序，控制驱动一些特性的开关。

enableRotationShaderPatching - enabled 或者 disable PreRotation feature
disableTiledRendering - disable Tile Rendering feature
forceClearOfUndefinedBuffer - 强制把未定义的Buffer显示洋红色
forceClearOfUndrawnBuffer - 强制把没有绘制的Buffer显示成洋红色
disableTextureUpdatesIncremental - disable Texture Incremental updating method
disableVboUpdateIncremental - disable VBO Incremental updating method
useUncachedVBOs - enable 或者 disable Cached VBOs
vboCacheType - 设置VBO cache的类型
vboDataAlignment - 设置VBO data alignment
binningRenderingModes - 设置Binning render模式(HW Binning, SW Binning, direct rendering)

如下示例如何设置Direct rendering 模式:

```
adb shell "chmod 777 /data/local/tmp"
adb shell "rm /data/local/tmp/adreno_config.txt"
adb shell "binningRenderingModes= 0x04>/data/local/tmp/adreno_config.txt"
adb reboot
```

以上debug方法对我们解决UI corruption问题很有帮助，因此希望OEM 工程师能够熟练地掌握这些调试方法。

# Android HW UI issue Q & A

# HW UI 问题解答(Solution: 00028654)

- **1.** 怎么检查一个apk是否应用了HW UI?
  Google 提供了dump HW UI 信息的方法，运行命令 dumpsys gfxinfo 就会打印出最近50 批commands中调用那些HWUI 接口函数。
  如果HW UI 没有应用，则dumpsys gfxinfo 就没有信息打印出来。
  下面是一个例子：
  adb shell
  >dumpsys gfxinfo [PID of the app, or name of app]

```
** Graphics info for pid 1095 [com.android.systemui] **
Recent DisplayList operations
DrawDisplayList
                Save
                ClipRect
```

- **2.** 怎样得到Application 设置的EGL configurations?
  apk可以应用Java层面的Holder 或者Window 操作来设置。我们可以检查是否是EGL configures引起的问题。
  具体命令如下：
  adb shell setprop debug.hwui.print_config choice
  adb shell stop
  adb shell start
  adb logcat

```
D/HardwareRenderer( 8808): EGL configuration com.google.android.gles_jni.EGLConfigImpl@4202ad88:
D/HardwareRenderer( 8808):   RED_SIZE = 8
D/HardwareRenderer( 8808):   GREEN_SIZE = 8
D/HardwareRenderer( 8808):   BLUE_SIZE = 8
D/HardwareRenderer( 8808):   ALPHA_SIZE = 8
D/HardwareRenderer( 8808):   DEPTH_SIZE = 0
D/HardwareRenderer( 8808):   STENCIL_SIZE = 8
D/HardwareRenderer( 8808):   SAMPLE_BUFFERS = 0
D/HardwareRenderer( 8808):   SAMPLES = 0
D/HardwareRenderer( 8808):   SURFACE_TYPE = 0x5e5
D/HardwareRenderer( 8808):   CONFIG_CAVEAT = 0x3038
```

# HW UI 问题解答(Solution: 00028654) - 续一

- **3.** 怎么获得GL-API调用trace 信息?

  Canvas API 是基于HWUI之上工作的，通过调用OPENGLES API来实现。因此GL API call trace 是一个很重要的调适方法。用如下命令可以在logcat log中获得GL-API 调用trace信息

  adb shell setprop debug.egl.trace 1

  adb shell stop

  adb shell start

- **4.** 如果我怀疑一个issue可能是HWUI引起的，我怎么关掉HWUI?

  1.) 从系统层面彻底关掉HWUI，影响所有application。

  HWUI 是android framework的一部分，Canvas API 本来是基于2D SKIA软件图像库实现的，现在部分被替换到3D Graphics来实现，因为在高分辨率的情况下，3D的性能要比2D号很多。如果关掉HWUI, 那么所有的Canvas API都只工作在2D SKIA软件图形库上面。

  修改如下code, 重新编译libhwui.so库文件并更新

  frameworks/base/core/jni/android_view_GLES20Canvas.cpp
  **static** jboolean **android_view_GLES20Canvas_isAvailable**(JNIEnv* **env**, jobject **clazz**) {
  + return JNI_FALSE;

  -

  }

  通过关掉HWUI, 我们可以判断是否application自己的问题。

  2.) 从具体应用apk层面关掉HWUI, 只影响特定的application。

  在apk的源文件目录，找到AndroidManifest.xml, 定位<application /> tag后做如下修改:

  android:hardwareAccelerated="false“

  然后重新build apk push到手机测试。

# HW UI 问题解答(Solution: 00028654) - 续二

- **5.** 怎么设置让HWUI全屏更新

  默认情况下，HWUI在render EGL Surface时设置了preserved bit, 当只有部分区域render不正确，需要检查一下是否是partial update的问题

  adb shell setprop debug.hwui.render_dirty_regions false

  adb shell stop

  adb shell start

  设置之后，HWUI render会不再保留EGL Surface里面的内容，每次都会render全部的内容。

- **6.** 怎么获取更多Texture的信息?

  HWUI 自己处理Texture相关的信息，意味着即使application 删除了一个Texture, 实际的数据可能还存在于HWUI的Cache里面，下面的编译选项可以输出更多Texture相关的debugging log

  [PLATFORM]/frameworks/base/libs/hwui/Debug.h

  - #define DEBUG_TEXTURES 0

  + #define DEBUG_TEXTURES 0

  然后rebuild hwui然后更新libhwui.so 手机测试。

  adb push libhwui.so /system/lib

- **7.** 怎么找到有问题的Call trace?
- Google 提供了GLTracer 用来Trace GLES/EGL的相关操作。具体的信息，可以参看Google Android 官方网站 http://developer.android.com/tools/help/gltracer.html

# GPU Performance issue Q & A

# GPU performance问题解答(Solution: 00028664)

- **1.** 怎么监控设备FPS (刷新率) ?
  通过设置adreno_config.txt可以使能FPS log输出，参考如下command：
  adb shell rm /data/local/tmp/adreno_config.txt
  adb shell “echo log.fps=1 > /data/local/tmp/adreno_config.txt”
  adb reboot

  然后就可以在logcat log中搜索fps关键字
  adb logcat | grep i –fps
  监控在不同应用场景下fps的变化。

- **2.** 怎么检查设备上GPU, CPU 和DDR 支持的最大Clocks
  首先要mount debugfs
  adb shell mount -t debugfs none /sys/kernel/debug
  对GPU adb shell cat sys/class/kgsl/kgsl-3d0/max_gpuclk
  对CPU adb shell cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
  对DDR adb cat /sys/kernel/debug/clk/bimc_clk/rate

- **3.** 怎么实时监控GPU, CPU, DDR的Clocks
  首先要mount debugfs
  adb shell mount -t debugfs none /sys/kernel/debug
  对GPU， adb shell cat /sys/kernel/debug/clk/oxili_gfx3d_clk/measure
  对GPU， adb shell cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
  对DDR， adb shell cat /sys/kernel/debug/clk/bimc_clk/measure

# GPU performance问题解答(Solution: 00028664) - 续一

- **4.** 什么是GPU Performance mode 和GPU performance mode?

  GPU performance mode就是强制GPU一直运行在最高时钟频率上。如果设置GPU performance mode问题得到改进，那就需要继续从GPU/Kernel方面检查。

  CPU performance mode是所有的CPU cores运行在最高时钟频率，同时DDR也是最高时钟频率。如果设置CPU performance mode 问题解决了，那么需要System performance team帮助进一步找到root cause.

- **5.** 怎么设置GPU/CPU performance mode?
- GPU performance mode

  你可以检查device 是否支持devfreq? 检查在/sys/class/kgsl/kgsl-3d0 路径下面是否有devfreq目录，如果有就支持，没有就不支持。

  如果你的device 支持devfreq,

  adb shell "echo 1 > /sys/class/kgsl/kgsl-3d0/force_clk_on"

  adb shell "echo 10000000 > /sys/class/kgsl/kgsl-3d0/idle_timer"

  adb shell "echo performance > /sys/class/kgsl/kgsl-3d0/devfreq/governor"

  adb shell "echo <max GPU clock> > /sys/class/kgsl/kgsl-3d0/gpuclk"

  反之你可以用如下命令：

  adb shell "echo 0 > /sys/class/kgsl/kgsl-3d0/pwrnap"

  adb shell "echo 10000000 > /sys/class/kgsl/kgsl-3d0/idle_timer"

  adb shell "echo none > /sys/class/kgsl/kgsl-3d0/pwrscale/policy"

  adb shell "echo <max GPU clock> > /sys/class/kgsl/kgsl-3d0/gpuclk“

- CPU Performance mode

  可以运行如下命令：

  adb shell stop mpdecision

  adb shell stop thermal-engine

  sleep 1

  adb shell "echo 1 > /sys/devices/system/cpu/cpu1/online"

  adb shell "echo 1 > /sys/devices/system/cpu/cpu2/online"

  adb shell "echo 1 > /sys/devices/system/cpu/cpu3/online"

  sleep 1

  adb shell "echo performance > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor"

  adb shell "echo performance > /sys/devices/system/cpu/cpu1/cpufreq/scaling_governor"

  adb shell "echo performance > /sys/devices/system/cpu/cpu2/cpufreq/scaling_governor"

  adb shell "echo performance > /sys/devices/system/cpu/cpu3/cpufreq/scaling_governor“

# GPU performance问题解答(Solution: 00028664) - 续二

- 6. 什么工具用来调适Performance问题? 怎么获得?

  一般用Systrace 和 Snapdragon performance visualizer (原名QView)工具来调适Performance问题。Systrace是Android SDK里面提供的一个功能，可以从Google开发者网站上下载。Qualcomm Snapdragon performance Visualizer 是高通自己的工具，在你开始一个Qualcomm的正式项目，你可以通过联系CE或者TAM，我们会把这个工具release给你。

- 7. 怎么使用Systrace ?

  你可以按照如下步骤去抓取一个问题的Systrace：

  1. 下载Android SDK并安装(http://developer.android.com).
  2. 打开tools 路径下面的 monitor.bat
  3. 点击“Capture system wide trace using Android systrace” (http://developer.android.com/tools/help/systrace.html#gui )
  4. 在'Select tags to enable' 里面勾选所有的选项
  5. 现在开始重现出现问题的应用场景，可能会比较慢
  6. 点击 OK开始抓取 Systrace

  然后把抓到Systrace 上传到case attachment供QC case owner 去分析问题。

- 8. 怎么使用Snapdragon Performance Visualizer 工具?

  完整安装好这个工具以后，里面会有一个用户手册(User guide)，有很详细的使用方法说明。

# GPU hang issue Q & A

# GPU hang FAQ (Solution: 00028668)

- **Q1) 我碰到一个issue，怀疑是GPU hang的问题。但是我不熟悉GPU hang, 怎么来确认是GPU hang呢?**
- **A1)** 虽然GPU hang 问题不容易解决，但是知道是否是GPU hang 相对容易。

  如果你在kernel log里面看到如下类似的log，那么这就是一个GPU hang。

  [ 1144.957632 / 03-25 15:31:58.510] kgsl kgsl-3d0: ndroid.keyguard[1226]: gpu fault ctx 4 ts 9899 status E5678011 rb 0bca/0bca ib1 3fa8a900/03b5 ib2 3f66f000/0000

  你可以找到"gpu fault" 字符串，后面跟着"status", "rb", "ib1" and ib2"等字符串。

  NOTE: 这个错误log内容在以后可能会改变，以增强GPU hang 恢复或检测。

  NOTE2：如果你在kgsl log中看到"time out"字符串，那么这就不是一个GPU hang，而是Long IB detection。

  Long IB detection 不同于GPU hang, 会有单独的Question 表述。

  <3>[40879.596442] c0 2200 kgsl kgsl-3d0: ogle.android.gm[21367]: gpu timeout ctx 8 ts 1220

  <3>[40879.596479] c0 2200 kgsl kgsl-3d0: ogle.android.gm[21367]: gpu failed ctx 8 ts 1220

- **Q2) GPU fault log的具体含义是什么?**
- **A2)** 你需要着重关注"status"的值， 我们用这个值作为Signature，来区分不同类型的GPU hang，因为它反映了GPU内部在GPU hang发生时刻的状态。

  [ 1144.957632 / 03-25 15:31:58.510] kgsl kgsl-3d0: ndroid.keyguard[1226]: gpu fault ctx 4 ts 9899 status E5678011 rb 0bca/0bca ib1 3fa8a900/03b5 ib2 3f66f000/0000

  在"rb"字符串后面的值是Ringbuffer 读和写操作指针的位置。

  在"ib1"和"ib2"字符串后面的值是IB buffer 的基地址，以及还没有被GPU执行的commands的大小。

  NOTE: 这并不意味相同的status值就表示相同类型的GPU hang 根本原因。但是它还是很有用的以区分不同类型的GPU hang，用作Signature。

- **Q3) 我已经确认一个GPU hang，还需要提供哪些信息去缩小GPU hang的范围?**
  **A3)** 下面是用来缩小GPU hang 范围的基本信息

  GPU snapshot

  这个是很关键信息用来缩小GPU hang的范围。没有GPU snapshot 我们没法缩小一个GPU hang的范围。

  GPU snapshot 是一个二进制文件，包含GPU在hang时刻的所有resource, 包括

  * GPU registers
  * Command buffers

  Shader binaries

  以及其他

  这些可以帮助我们理解GPU在Hang时刻在做什么，为什么卡住了。

  如果你现在没有GPU snapshot，请一定提取出GPU snapshot来。再次重申，没有GPU snapshot我们没有办法找到GPU hang的根本原因。

# GPU hang FAQ (Solution: 00028668) – 续一

B. 你当前使用的QCOM Build

对GPU, 我们需要APSS build 信息。如果没有APSS build, 那么Meta build 也可以，我们可以从Meta build中查到对应的APSS build。但是很难从其他build, 比如MPSS 或者RPM 匹配对应的APSS build。

例如) LNX.LA.3.6-00720-8084.0-1 (APSS build, 很好)

F8084AAAAANLYD121121A (Meta build, 也可以)

MPSS.BO.1.0.r6-00011-M9635TAARANAZM-1(MPSS build, 这个不行，无法找到对应的APSS build)

正确的APSS build信息可以帮助我们去检查在你当前的build上是否缺少一些已知的GPU hang问题的修复。如果我们发现有任何缺少的change, 就可以立刻提供一个Test SBA。

-你的GPU hang 是否可以重现吗?

如果是的，我们可以很容易地接近GPU hang问题。最好能知道GPU hang发生的频率，是100%，还是15分钟一次，等等。

- **Q4) 我知道GPu snapshot是缩小GPU hang问题所必须的，那么怎么提取GPU snapshot呢？**
- **A4)** 可以用一下几种方法：

方法 1

========

GPU snapshot 被保存在sys文件系统默认的虚拟文件节点上，但是重启手机后就删除了。

如果在GPU hang发生时手机还是活着，可以用以下方法提取GPU snapshot.

- 确认GPU snapshot是否已经产生了？

>adb shell cat sys/class/kgsl/kgsl-3d0/snapshot/timestamp

如果不为0，表示GPU snapshot 已经产生了

从虚拟节点上面保存GPU snapshot到一个文件

>adb pull /sys/class/kgsl/kgsl-3d0/snapshot/dump GPUsnapshot.bin

如果保存成功了，提供GPUsnapshot.bin 文件给我们分析。

NOTE: 通过这种方法得到的GPU snapshot是第一次GPU snapshot.

如果你有多个GPU hang, 只有最后一次GPU hang影响用户使用，这里提取的GPU snapshot可能不包含最后一次GPU hang的信息。

NOTE2: 不是每一个GPU hang都会影响用户使用，因为有些GPU hang是可以恢复的。

# GPU hang FAQ (Solution: 00028668) – 续二

方法 2
========
在很多情况下，在GPU hang发生后可能不太容易device 还是活着状态。
有些GPu hang引起系统不稳定，最终导致手机crash。
如果你碰到这种情况，而且可以得到RAMDUMP, RAMDUMP将是提取GPU snapshot的第二条选择。
把kernel log和RAMDUMP一起提供给我们。
kernel log应该有以下的信息：

<3>[ 1406.544189 / 04-24 13:53:47.698] kgsl kgsl-3d0: |kgsl_device_snapshot| snapshot created at pa 0x25880000 size 447960
这里列出了GPU snapshot的物理地址和大小。有了RAMDUMP和这个信息，我们可以尝试从RAMDUMP中提取GPU snapshot。
然而请注意，这是第二条选择，因为可能提取GPU snapshot会失败，或者没有我们想要的全部的信息。

方法 3
========
Adreno 库支持把GPU snapshot保存到一个OEM指定的路径
如果你可以在你的device上重现问题，下面是最容易提取GPU snapshot的方法

怎么设置GPU snapshot 保存的路径
- 创建一个名为adreno_config.txt的文件文件， 文件名很重要，Adreno 库通过这个文件来加载配置选项。
- 在这个文件里添加下面一行
gpuSnapshotPath={一个你想要保存GPU snapshot的路径, 必须有写权限}
- 把这个adreno_config.txt 文件推送到设备的/data/local/tmp
重启设备，在Adreno 库初始化时，就会把这个路径设置为GPU snapshot保存路径。
下面是一个设置GPU snapshot保存路径的例子，在GPU hang发生时，GPU snapshot 文件名为dump_xxxxxxxx就会被保存在/data/local/tmp
>vi adreno_config.txt
gpuSnapshotPath=/data/local/tmp
>adb push adreno_config.txt /data/local/tmp
>adb shell sync

>adb reboot
>adb wait-for-device
>adb shell chmod 777 /data/local/tmp

# GPU hang FAQ (Solution: 00028668) – 续三

- **Q5) 你在前面的问题里提到GPU hang recovery ，那么什么是GPU hang recovery，它是怎么工作的呢？**
- **A5)** QCOM把它称作GPU hang recovery 或者 GPU fault tolerance。
  我们知道最好是避免GPU hang发生，但是如果不能避免，退而求其次在GPU hang发生后能够避免对用户产生影响也很好。
  GPU fault tolerance (GFT)的目的就是即使GPU hang发生了减少对用户的影响。
  终端用户不用关心GPU hang发生，即便是在很多GFT的情况下GPU hang都发生了。

  GFT工作的方式如下：
  - KGSL (kernel part of GPU driver) detects GPU hang rapidly.
  - Reset GPU HW
  - Issue last GPU command again.
  - Check GPU hang
  - If no GPU hang, move forward to next command.
  - If GPU hang detects again, skip the command and issue next command.
  - If no GPU hang, move forward to next command.

  If GPU hang detects again, mark context bad and this eventually kills the process from user mode.

- **Q6) 前面你提动long IB detection 不同于GPU hang。那么什么是long IB detection，需要提供什么信息?**
- **A6)** Long IB detection 是一种情况，GPU 处理一批GPU command花费了太长的时间。
  NOTE: IB (indirect buffer)里面包含的是GPU commands，Long IB 意思是GPU command buffer 太长了。
  虽然这不是GPU hang，GPU一直在运行中处理GPU command, KGSL driver等了太长时间去等待这项任务完成。
  如果GPU在很长的时间还不能完成一个任务，终端用户就会看到绘图卡住了。
  当前Long IB detection超时时间是2秒钟，对于大多数情况都是足够了。
  如果KGSL 没有定义超时时间，用户感觉到系统UI都卡住了，因为GPU可能被一个context的超长IB一直占用着。
  Long IB的典型原因：
  - 应用程序bug

  一个典型的会产生Long IB 的情况就是在shader中的无限或者很长的循环。如果应用程序产生了无效的shader code，就会引起Long IB.
  - 针对桌面电脑的Web GL应用网站

  一些WebGL 网站会产生Long IB因为这些网站是针对高性能的桌面系统GPU。一些Web GL网站甚至把桌面电脑弄得很慢，因为他们对GPU产生了太多繁重的操作。
  在这种情况下，浏览器进程可以被Long IB detection杀掉而退出。

  为了确认是否是Long IB detection引起的问题， 你可以从下面方法禁止Long IB detection。
  >adb shell "echo 0 > > /sys/class/kgsl/kgsl-3d0/ft_long_ib_detect“

# GPU pagefault issue Q & A

# GPU Page Fault FAQ (Solution: 00028669)

- **Q1) 什么是GPU page fault? 怎么确认是不是GPU page fault?**
- **A1)** GPU 有自己独有的MMU (GPU IOMMU)去访问CPU-GPU 共享的内存。当GPU试图去访问一个非法的地址时，GPU page fault就发生了。

  你可以在kernel log中很容易地找到GPU page fault，下面就是一个GPU page fault 的例子，包含有“GPU PAGE FAULT"字符串。

  <2>[122867.712606] c1 81 kgsl kgsl-3d0: |kgsl_iommu_fault_handler| GPU PAGE FAULT: addr = 773E9B00 pid = 2770
  <2>[122867.712617] c1 81 kgsl kgsl-3d0: |kgsl_iommu_fault_handler| context = 0 FSR = 80000002 FSYNR0 = 582 FSYNR1 = 3C030008(read fault)
  <3>[122867.712633] c1 81 kgsl kgsl-3d0: ---- premature free ----
  <3>[122867.712640] c1 81 kgsl kgsl-3d0: [773DB000-773FB000] (vertexarraybuffer) was already freed by pid 2770
  <3>[122867.712649] c1 81 kgsl kgsl-3d0: ---- nearby memory ----
  <3>[122867.712679] c1 81 kgsl kgsl-3d0: [773D6000 - 773DA000] (+guard) (pid = 2770) (texture)
  <3>[122867.712686] c1 81 kgsl kgsl-3d0: <- fault @ 773E9B00
  <3>[122867.712692] c1 81 kgsl kgsl-3d0: [773FC000 - 77400000] (+guard) (pid = 2770) (texture)

- **Q2) GPU page fault log的含义是什么?**
- **A2)** 你可以在log中找到产生GPU page fault 的进程PID 和GPU试图要去访问的地址。

  NOTE: 这个地址是GPU 地址，不同于普通的CPU 虚拟地址。
  <2>[122867.712606] c1 81 kgsl kgsl-3d0: |kgsl_iommu_fault_handler| GPU PAGE FAULT: addr = 773E9B00 pid = 2770
  第二行包括IOMMU 寄存器的信息和page fault 类型 - read fault还是write fault
  <2>[122867.712617] c1 81 kgsl kgsl-3d0: |kgsl_iommu_fault_handler| context = 0 FSR = 80000002 FSYNR0 = 582 FSYNR1 = 3C030008(read fault)
  同时在fault地址附近的内存信息。
  我们可以从这些信息中推测引起GPU page fault 的内存类型。
  然而这并不意味着我们可以确定引起GPU page fault 的内心类型，因为那块内心实际上并没有影射到GPU page table。
  为了确定内存类型，我们还需要更多的信息。这只是一个Hint而已。
  <3>[122867.712679] c1 81 kgsl kgsl-3d0: [773D6000 - 773DA000] (+guard) (pid = 2770) (texture)
  <3>[122867.712686] c1 81 kgsl kgsl-3d0: <- fault @ 773E9B00
  <3>[122867.712692] c1 81 kgsl kgsl-3d0: [773FC000 - 77400000] (+guard) (pid = 2770) (texture)

# GPUPage Fault FAQ (Solution: 00028669) - 续一

- **Q3) 对GPU page fault 问题，我需要提供什么信息给QCOM?**
- **A3)** 一般地，我们需要知道QCOM build ID, 这个可以帮助我们检查QCOM内部记录，找到任何在你的build上面缺少的修复。
  如果没有任何已知问题， 我们需要找到哪部分的代码引起这个问题。
  GPU驱动分为两个部分，KGSL (内核部分)和 UMD (用户态驱动)
  对KGSL 你可以直接查看code，这个在Linux kernel 目录下。然而QCOM并不共享UMD 驱动的源代码，而是作为共享库提供 (*.so 文件)。

  (1) KGSL的一个功能就是管理context和page table。
  如果你在kernel log里看到一个无效的PID 而且/或者一个无效的地址， 这很有可能是KGSL没有正确的处理page table 和context引起的问题。
  一个典型的无效PID 是0, 地址是FFFFxxxx
  kgsl kgsl-3d0: |kgsl_iommu_fault_handler| GPU PAGE FAULT: addr = FFFF0000 pid = 0

  如果你看到一个GPU pagefault有无效的PID 和地址，请先提供kernel log 和logcat log。

  (2) 如果你在GPU page fault log里面看到有效的地址和PID， 这可能是UMD 驱动的问题，有可能GPU 内存被提前释放了。
  UMD驱动的一个功能就是管理GPU使用的内存类型。GPU 内存有很多类型，比如texture, command, VBO, FBO, egl image 等。
  使用不同内存的代码路径完全不同，因此第一步先要缩小是哪种类型的内存引起GPU page fault。
  举例，如果你看到下面的log, GPU 在Process PID 2770里面试图去访问 773E9B00
  <2>[122867.712606] c1 81 kgsl kgsl-3d0: |kgsl_iommu_fault_handler| GPU PAGE FAULT: addr = 773E9B00 pid = 2770
  因此我们需要知道内存地址773E9B00 是什么类型。为了理解这个，我们需要用使能GPU Memory Logging的UMD 驱动库。
  NOTE: 这个使能GPU Memory Logging的UMD 驱动库需要QCOM 为你单独重新编译提供，正式发布的库不包含这个功能。

# GPU Page Fault FAQ (Solution: 00028669) - 续二

- 你可以用下面方法动态打开GPU memory Logging的功能。

  >vi adreno_config.txt

  log.vmem=1

  >adb push adreno_config.txt /data/local/tmp

  >adb shell sync

  >adb reboot

  >adb wait-for-device

  >adb shell chmod 777 /data/local/tmp

  然后你可以在设备的/data/local/tmp路径下找到这些 vmem_{pid}.txt，这些Logging允许用户了解哪种类型的内存分配在哪个地址上。

  如果你碰到这种类型的GPU page fault, 请提供这些vmem_{pid}.txt 和kernel log。

- (3) 如果是两种类型的GPU page fault (无效的PID/地址和有效的PID/地址) 都有， 最好能提供这个page fault 对应的GPU snapshot。

KGSL 有一个办法可以在GPU page fault的时候触发一个GPU snapshot，你可以用下面方法在GPU page fault 时dump GPU snapshot。

>adb shell "echo 0x3 > > /sys/class/kgsl/kgsl-3d0/ft_pagefault_policy"

请提供GPU snapshot和其他信息给QCOM检查。

# KGSL driver Q & A

# KGSL FAQ (Solution: 00028655)

- **Q1) KGSL 的源文件路径是哪里?**
- **A1)** KGSL所有的源文件路径在 */kernel/drivers/gpu/msm*，同时GPU DCVS governor, 在 */kernel/drivers/devfreq/governor_msm_adreno_tz.c 和*
- */kernel/include/linux/msm_adreno_devfreq.h*

- **Q2) GPU dtsi 文件路径在哪里?**
- **A2)** 目标dtsi 文件路径在 kernel/arch/arm/boot/dts/, 请注意针对不同芯片有独立，内容不同的dtsi文件。
- 例如, *msm8974-gpu.dtsi msm8974-v2.dtsi msm8974-v2.2.dtsi msm8974pro.dtsi msm8226-gpu.dtsi msm8610-gpu.dtsi apq8084-gpu.dtsi*

- **Q3) GPU dtsi 文档的入口在哪里?**
- **A3)** */kernel/Documentation/devicetree/bindings/gpu/adreno.txt*

- **Q4) 怎么察看Gpu clock 和 Bus voting?**
- **A4)** 具体步骤如下:
- //检查GPU 正在运行的clk的步骤
- >adb shell
- >cd / sys/kernel/debug /clk/ oxili_gfx3d_clk
- >cat measure
-
- //检查GPU bus voting的步骤
- >adb shell
- >cd /sys/kernel/debug /msm-bus-dbg/client-data
- >cat grp3d*
- 最后一个数值就是GPU ib votes

- **Q5) 怎么获得包含KGSL event的Ftrace trace log?**
- **A5)** 用下面的方法来获得KGSL trace events log:
- 挂载 debugfs:
- >adb shell mount -t debugfs none /sys/kernel/debug

# KGSL FAQ (Solution: 00028655) – 续一

- 查看可用的events:
- >adb shell cat /sys/kernel/debug/tracing/available_events
- >adb shell cat /sys/kernel/debug/tracing/available_events | grep kgsl

- 增大trace buffer size
- > adb  shell “echo 16384 > /sys/kernel/debug/tracing/buffer_size_kb”
- 
- 创建一个文本文件，包含你想要的events, 例如这些：
- kgsl:kgsl_a2xx_irq_status
- kgsl:kgsl_a3xx_irq_status
- kgsl:kgsl_clk
- kgsl:kgsl_irq
- kgsl:kgsl_rail
- kgsl:kgsl_bus
- kgsl:kgsl_pwrlevel
- kgsl:kgsl_buslevel
- kgsl:kgsl_pwrstats
- kgsl:kgsl_pwr_set_state
- kgsl:kgsl_pwr_request_state
- kgsl:kgsl_active_count

- 把这个event 列表推送到设备上:
- >adb push events.txt /sys/kernel/debug/tracing/set_event
- 进行相关操作:  运行你的测试，重现问题

- 抓出log:
- >adb pull /sys/kernel/debug/tracing/trace

- **Q6) 怎么强制让GPU clocks/bus/power rail 一直使能?**
- **A6)** 在adb 运行在root 用户下，进入adb shell 运行一下命令：
- >adb shell
- >cd /sys/class/kgsl/kgsl-3d0

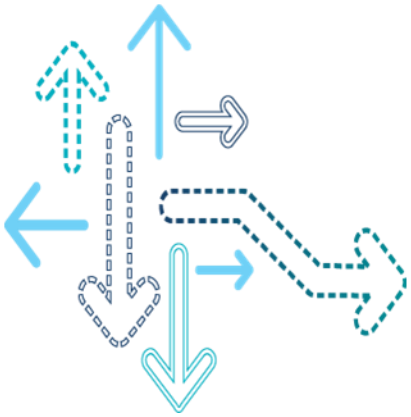
# KGSL FAQ (Solution: 00028655) – 续二

- >echo 1 > force_clk_on /* Clocks are not SW gated during standard rendering */
- >echo 0 > force_clk_on /* Resume SW gating of clocks */
- >echo 1 > force_rail_on /* power rail never turned off */

- **Q7) 怎么禁止idle timer 或者禁止GPU 进入睡眠模式?**
- **A7)** 在adb 运行在root 用户下，进入adb shell 运行一下命令：

- >adb shell
- >cd /sys/class/kgsl/kgsl-3d0
- >echo 1000000 > idle_timer

- 参考solution：00028655



# Questions?

**https://support.cdmatech.com**