Application Note

调试 TDA4x 和 AM6x 器件上的 GPU 驱动程序问题

Erick Narvaez, Krunal Bhargav *EPD Processors*

摘要

德州仪器 (TI) 的处理器可能包含一个图形处理单元 (GPU)，用于加快图形相关任务的计算。图形处理堆栈中偶尔会出现错误，导致应用程序出现问题。鉴于处理堆栈的复杂性（即跨越多个软件层并在内部运行于长硬件流水线上），这类问题可能很难调试。此外，各层中不同的组件可能由不同的开发人员创建，将这些组件协同工作也颇具挑战。本指南介绍了图形应用程序中可能出现的不同问题，以及系统性调试 GPU 相关问题的步骤。

内容

商标

1 引言

本应用手册概述了 GPU 相关问题的调试过程。要成功进行调试，必须能够识别与图形相关的问题，还必须了解报告问题和提供必要信息的过程。

第一步是确定图形应用程序的问题并将这些问题与常见问题的症状关联起来。由于这些错误的副作用会遍布整个系统，因此无法提供完整的错误列表。此处提供了一个大致的列表，用于帮助识别常见的问题。第二步，需要了解识别和解决问题所需的信息。为了确定故障的严重程度并准确定位原因，需要多种调试日志和故障详细信息。最后，如果初始信息和基本日志记录不足以解决问题，则需要使用一些先进的调试日志记录技术。本指南中的重要知识包括围绕图形问题所使用的术语以及 GPU 专用的工具。

2 图形应用程序的常见问题

本节介绍在处理与图形相关的问题时遇到的常见问题。这些问题通常在显示器上显示，或者表现为应用中的数据流中断。

2.1 系统或应用程序冻结

冻结是应用程序停止处理数据的状态。冻结可能发生在应用程序或系统级别。在应用程序级别，冻结隔离到运行图形工作负载的进程。在系统级别，操作系统中可能存在其他症状，例如内核错误，这也会导致其他应用程序出现故障。这种区分很重要，因为冻结可以指示故障的严重程度以及位置。冻结通常伴随进程堆栈的日志转储，在这种情况下异常会使执行停止或根本没有任何故障指示。节 2.1.1 是发生并导致系统冻结的内核错误的示例。

2.1.1 典型内核紧急日志

```
root@tda4vm-sk:~# [ 5894.898990] Unable to handle kernel NULL pointer dereferen0
[ 5894.907770] Mem abort info:
[ 5894.907940] Unable to handle kernel paging request at virtual address ffb8000
[ 5894.910552] ESR = 0x96000046
[ 5894.918446] Adjusting arch_sys_counter more than 11% (651145911 vs 93113548)
[ 5894.921484] EC = 0x25: DABT (current EL), IL = 32 bits
[ 5894.928527] Unable to handle kernel paging request at virtual address ffd2004
[ 5894.933800] SET = 0, FnV = 0
[ 5894.941691] Mem abort info:
...
[ 5894.984251] Hardware name: Texas Instruments J721E SK (DT)
[ 5894.984254] pstate: 80000085 (Nzcv daIf -PAN -UAO -TCO BTYPE=--)
[ 5894.984265] pc : _raw_write_lock_irqsave+0x168/0x318
[ 5894.984270] lr : try_to_wake_up+0x5c/0x4e0
[ 5894.984271] sp : ffff8000113afdd0
[ 5894.984273] x29: ffff8000113afdd0 x28: ffff8000100d8ad0
[ 5894.984277] x27: ffff00087fa88300 x26: 0000000000000006
...
[ 5894.984319] x1 : 0000000000000000 x0 : 0000000000010003
[ 5894.984323] Call trace:
...
[ 5894.984397] el1_sync_handler+0xac/0xc8
[ 5894.984399] el1_sync+0x88/0x10
[ 5894.984401] 08ff00800v10f826b8
[ 5894.984406] Code: 451806018d5334611 526b0a90 f8800871 f8850fc60)
[ 5894.984413] ---[ end trace 2f5eabcaa9b203ad ]---
[ 5894.984416] Kernel panic - not syncing: Oops: Fatal exception in interrupt
[ 5894.984419] SMP: stopping secondary CPUs
[ 5896.056422] SMP: failed to stop secondary CPUs 0-1
[ 5896.056429] Kernel Offset: disabled
[ 5896.056431] CPU features: 0x0040022,20006008
[ 5896.056433] Memory Limit: none
[ 5896.481800] ---[ end Kernel panic - not syncing: Oops: Fatal exception in in-
```

2.2 屏幕撕裂

屏幕撕裂现象在图形应用程序中非常普遍，在视频游戏中最为常见。GPU 以特定的帧速率运行，显示器可能有单独的帧速率。当出现屏幕撕裂或屏幕闪烁时，GPU 和显示器之间的同步可能会出现问题。描述撕裂的样子非常重要。水平线表示 GPU 正在使用下一帧数据更新前一帧，但显示器过早使用更新。观察撕裂区域周围的像素数据以及屏幕的撕裂程度，因为这可以表明失去同步的严重程度。图 2-1 是由于显示器和图形应用程序之间失去同步而造成屏幕撕裂的一个示例。

请注意，在略高于中间标记的位置，与上一帧的三角形存在不一致。这是*屏幕撕裂*问题的典型示例。

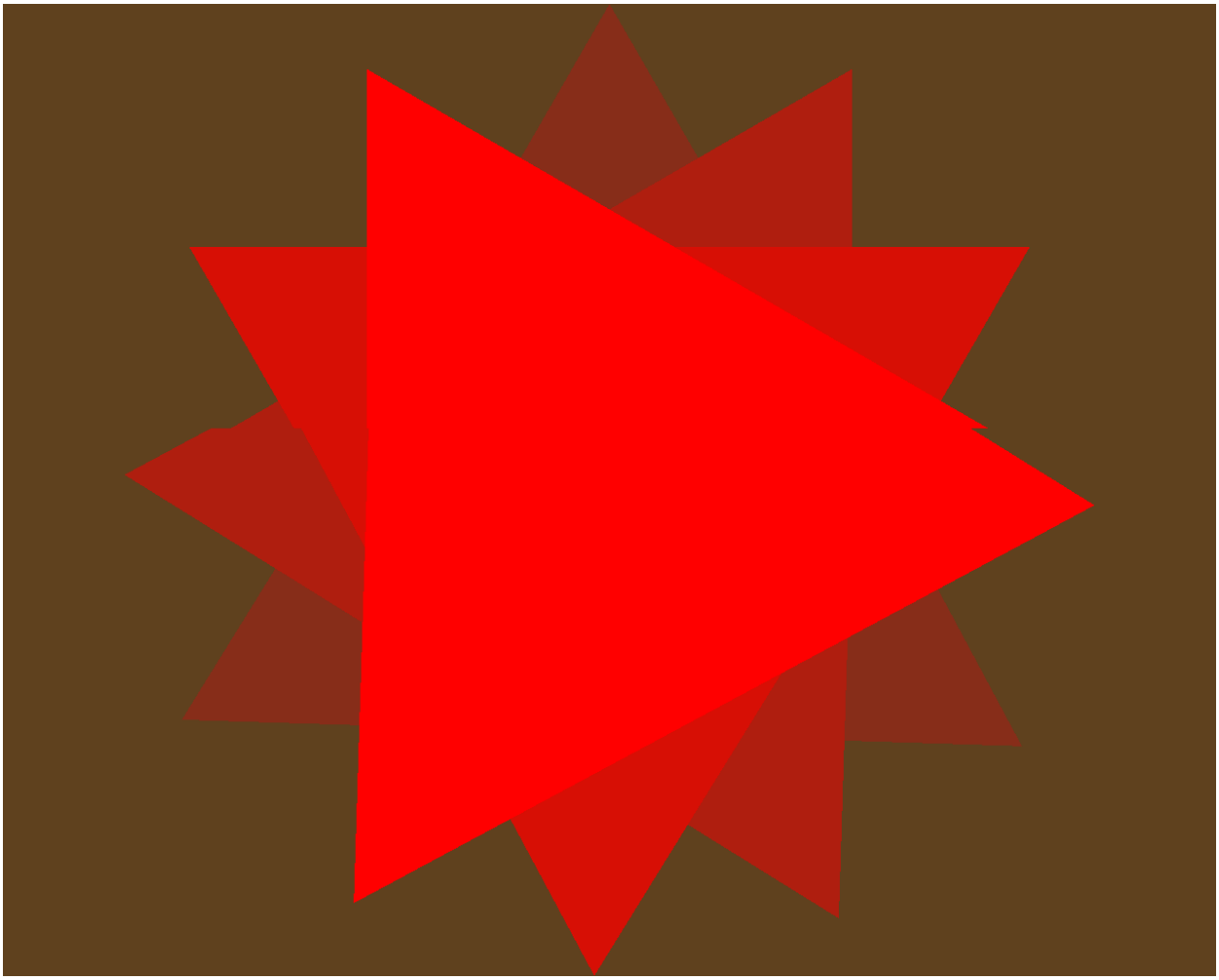

图 2-1. 屏幕撕裂示例

2.3 屏幕上出现伪影或损坏

屏幕问题中的伪影或损坏问题更难跟踪，因为这些问题可能由众多因素导致。从驱动程序 API 中的错误到硬件中的问题，可能原因的范围非常之大。但是，通过本文档后面介绍的一些工具，可以快速确定问题的根本原因。主要需要关注的是，损坏是否隔离到 GPU 输出上的某个区域、损坏持续时长以及损坏的覆盖范围。

图 2-2 展示了不同类型的伪影。一般情况下，伪影是随机数据，分布在一个大块、重复的小块或长块中。无论损坏的表现形式如何，通常都会由于渲染流水线的任何级别的损坏而导致帧缓冲区中引入过时数据或随机数据。

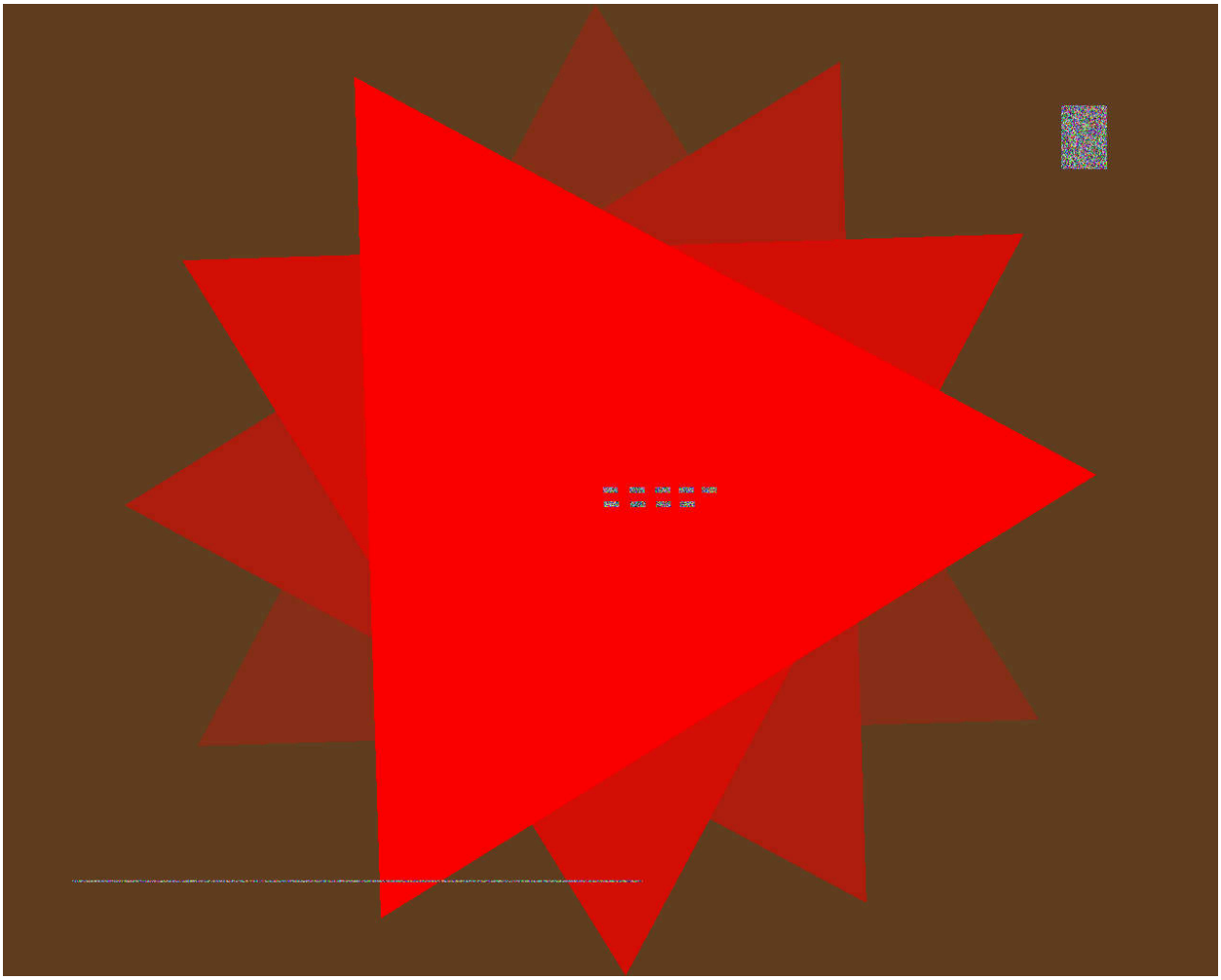

图 2-2. 伪影示例

2.4 空白屏幕

如果程序中的问题和代码停止执行导致显卡驱动程序而没有渲染，则可能出现没有渲染导致出现空白屏幕的情况。此问题通常与图形 API 静默发生故障或图形软件栈中的某项内容未正确配置有关。

2.5 低帧速率

如果系统回退到软件渲染（软件光栅化），应用程序有时仍然可以渲染。GPU 驱动程序仍然存在，但由于配置错误或引导序列期间的时序问题，窗口化系统可能无法识别 GPU，并回退到使用中央处理单元 (CPU) 进行渲染。在这些情况下，确保正在使用 GPU 的最简单方法是检查它的负载。也可能会存在部分利用率的情况，因此请检查应用程序的所有部分是否都正在使用 GPU 渲染。

2.6 GPU 驱动程序日志和硬件恢复

GPU 配备了一个框架来识别图形处理中的问题，并发出硬件重置或恢复以缓解症状。驱动程序中有计数器，用于跟踪已发生的恢复次数。需要分析硬件恢复 (HWR) 的原因，因为这些错误表示在 GPU 处理过程中发生了故障。GPU HWR 日志具有标准外观，识别它们的最简单方法是在日志中查找标题 *PVR*。控制台中的日志概述了一些一般信息，例如驱动程序版本和内部版本选项。日志还包括与崩溃有关的更具体信息。该数据由固件以标准化格式转储，可以与 TI 共享以进行分析。这些日志中并不总是提供识别问题所需的相关数据，因此，指派的工程师可能会请求再次收集日志，其中包含更多详细信息和修改后的驱动程序，以获取必要的信息。

在某些情况下，GPU 不会发出 HWR，但会显示与 GPU 驱动程序相关并且 GPU 处理受到影响的其他消息。与前面提到的情况类似，这些日志需要与 TI 共享以进行分析，并且可以指示 GPU 驱动程序中的其他类型的故障。节 2.6.1 展示了一个典型的 HWR 示例。

搜索这些内容的简单方法是在 dmesg 日志中搜索 *PVR*。 *PVR 日志转储收集* 一节介绍了如何收集这些日志以与 TI 共享。

2.6.1 典型 GPU HWR 日志

```
[  275.343261] PVR_K:  228: ------------[ PVR DBG: START (High) ]------------
[  275.350212] PVR_K:  228: OS kernel info: Linux 6.1.46-g5892b80d6b #1 SMP PREEMPT Wed Apr  3
19:34:28 UTC 2024 aarch64
[  275.360827] PVR_K:  228: DDK info: Rogue_DDK_Linux_WS rogueddk 23.2@6460340 (release)
j721s2_linux
[  275.369798] PVR_K:  228: Time now: 275369792us
[  275.374260] PVR_K:  228: Services State: OK
[  275.378483] PVR_K:  228: Server Errors: 0
[  275.382800] PVR_K:  228: Connections Device ID:0(128) P1128-V1-T1153-VayaDriveConsol
[  275.390613] PVR_K:  228: ------[ Driver Info ]------
[  275.395699] PVR_K:  228: Comparison of UM/KM components: MATCHING
[  275.405345] PVR_K:  228: KM Arch: 64 Bit
[  275.410821] PVR_K:  228: Driver Mode: Native
[  275.415527] PVR_K:  228: UM Connected Clients: 64 Bit
[  275.420814] PVR_K:  228: UM info: 23.2 @  6460340 (release) build options: 0x80000810
[  275.428899] PVR_K:  228: KM info: 23.2 @  6460340 (release) build options: 0x00000810
[  275.437396] PVR_K:  228: Window system: wayland
[  275.442353] PVR_K:  228: ------[ Server Thread Summary ]------
...
[  275.481396] PVR_K:  228: ------[ RGX Device ID:0 Start ]------
[  275.487504] PVR_K:  228: ------[ RGX Info ]------
...
[  275.789271] PVR_K:  228: ------[ RGX registers ]------
...
[  276.648689] PVR_K:  228: ---- [ RISC-V internal state ] ----
...
[  276.759651] PVR_K:  228: ------[ RGX FW Trace Info ]------
...
[  276.777139] PVR_K:  228: ------[ Full CCB Status ]------
...
[  276.868040] PVR_K:  228: ------[ AppHint Settings ]------
...
[  276.940907] PVR_K:  228: ------[ Active Sync Checkpoints ]------
...
[  277.035347] PVR_K:  228: ------------[ PVR DBG: END ]------------
[  277.043511] ------------[ cut here ]------------
...
[  277.303180] ---[ end trace 0000000000000000 ]---
```

3 图形问题支持流程

要快速解决图形问题，请在图形调试场景中执行以下步骤。对问题进行快速响应和细致观察可以加速这一过程。

备注

能否在所有器件（50% 或更多器件）上快速（一个小时以内）重现问题是解决问题的限制因素。否则，通过收集日志来解决问题会比较困难。

3.1 提交初步描述

现在已经确定了与图形相关的问题，接下来的调试步骤需要收集以下信息。此信息越完整，调试体验就越顺畅。

1. 症状：在系统级别出现了哪些症状？是出现在控制台输出吗？节 2 详细说明了许多常见的症状，有些症状可能会同时发生。
2. 设置：如何使用 GPU？显示器是本地 Linux® 显示器还是远程显示器？这是 RTOS + Linux 设置中的视觉处理流水线吗？使用的 GPU 驱动程序和软件开发套件 (SDK) 版本是什么？
3. 场景：此问题是在什么情况下发生的？是否有其他处理器正在运行？是否正在处理较大的视觉处理流水线或类似内容的一部分？该问题是在启动应用程序后立即发生还是在一段时间后发生？
4. 是否可以重现：重现此问题的难易程度如何？问题是仅出现在一个电路板上还是出现在所有电路板上？重现此问题需要多长时间？
5. 恢复：问题是否可以在没有任何干预的情况下恢复，或者系统是否处于崩溃状态？
6. 日志记录：请共享控制台和内核 (`dmesg`) 中的所有日志，或注明在失败发生时没有日志。

在整个调试过程中需要这些初步信息，这些信息与日志一起用于确定问题。

3.2 确定问题是否可在 TI EVM 上重现

在理想情况下，问题可在有最新 SDK 版本可用的 TI EVM 上重现。这样一来，TI 工程师就可以在本地重现问题并取得进展，而无需与客户来回沟通。遗憾的是，用户可能有定制的专有软硬件，无法共享。此外，问题有时仅在系统负载较高且需要比可以重新创建的设置复杂得多的生产系统中重现。这些问题会导致远程重现非常麻烦。

但是，如果将应用程序移植到 TI EVM 并在其上重现问题的开销很低，这将大大加快调查速度，并且值得根据紧急程度进行探索。否则，调查将继续在定制硬件上进行。

3.3 提供跟进测试和日志

大多数问题都可以通过升级到最新的驱动程序来解决。如果问题未解决，用户可以与 TI 工程师合作，通过提供日志并将提供的补丁程序应用于 GPU 驱动程序来解决问题。了解如何更新系统中的 GPU 驱动程序以加速调试过程是一项重要的知识。节 5 提供了有关在 Linux 系统上构建和安装显卡驱动程序的详细说明。此外，节 4 还可以作为在调试过程中可能用到的不同工具的参考。

4 用于调试 GPU 驱动程序的工具

在对问题描述和日志进行初始分析后，有时需要引入先进的调试工具和技术来精简调查。显卡驱动程序随附一些驱动程序工具，它们可以提供固件和 API 跟踪。有时需要对显卡驱动程序进行详细的日志记录和修改。以下各节将介绍这些步骤，以说明清楚并提供参考。

4.1 Linux® DebugFS 中的驱动程序状态

GPU 驱动程序会在 Linux 内核 debugfs (/sys/kernel/debug/pvr) 中添加一个用于显示相关统计信息的条目。这些统计信息可用于识别 GPU 驱动程序版本、负载和配置。首先要浏览的是 *status* 条目：

```
root@j721s2-evm:/sys/kernel/debug/pvr# cat status
Driver Status: OK

Device ID: 0:128
Firmware Status: OK
Server Errors: 0
HWR Event Count: 0
CRR Event Count: 0
SLR Event Count: 0
WGP Error Count: 0
TRP Error Count: 0
FWF Event Count: 0
APM Event Count: 15
GPU Utilisation: 0%

VM0
2D Utilisation: 0%
GEOM Utilisation: 0%
3D Utilisation: 0%
CDM Utilisation: 0%
RAY Utilisation: 0%
GEOM2 Utilisation: 0%
```

重要部分如下：

- 固件状态：在 GPU 内部运行的固件的当前状态。确保固件状态正常，否则 GPU 状态会恶化，并且处理会受到影响。
- HWR 事件计数：已发生的 HWR 数量。正常情况下，此数字保持为 0。如果此数字大于 0，请提交工单以分析问题的严重程度。
- GPU 利用率：此百分比粗略概括了 GPU 负载。可使用 PVRTune (Imagination Technologies® 的一款工具) 来进行更精确的测量。请在后续的 TI 应用手册中了解 PVRTune 指令。

4.2 驱动程序 AppHints

debugfs 还包含有关 GPU 驱动程序使用的 *AppHints* 的信息。AppHints 是用于修改驱动程序行为的运行时配置。但是，这些 AppHints 的详细信息不会被公开，因为这些详细信息对驱动程序的正常运行来说用处不大。不过，这些提示有助于调整驱动程序的一些特性，并且在需要高级调试的情况下非常有用。可通过 /etc/powervr.ini 文件为所有图形程序 (默认部分) 或特定程序设置这些 AppHints：

```
[default]
ParamBufferSize=2097152
[gles2test1]
ParamBufferSize=16777216
```

语法是用方括号括住可执行文件的名称，后跟要启用的 AppHints。默认部分适用于所有可执行文件，任何其他部分可针对特定应用程序指定。在本例中，单元测试用例 gles2test1 使用小于其他图形程序的 ParamBufferSize 运行。

4.3 PVR 日志转储收集

如节 2.6 中所述，HWR 日志包含 GPU 驱动程序和固件的日志。要从 GPU 收集日志，可使用名为 pvrlogdump 的实用程序。pvrlogdump 是随 GPU 驱动程序安装附带提供的。默认情况下，该实用程序会将日志保存在 /tmp

目录中，并输出文件的名称以方便提取文件。除了 pvrlogdump 生成文件外，它对于共享应用执行的控制台也很有用，因为这显示了 HWR 发生的时间。以下是 pvrlogdump 命令输出的示例：

```
root@j721s2-evm:~# pvrlogdump
Checking driver state ............... initialised
Checking for debugfs ................ found
Checking for lockdep ................ not found
Checking for ftrace ................. not found
Checking for firmware log groups .... not found
There are no AppHints enabled in /etc/powervr.ini or in debugfs for any of the firmware log groups.
Unless 'pvrdebug' tool was used for that purpose there will be no information in firmware log.
Please consider enabling some of the firmware log groups before the problem occurs.
Dumping data .......................
[ 163.767302] PVR_K: 1005: User requested PVR debug info
[ 163.772711] PVR_K: 1005: ------------[ PVR DBG: START (High) ]------------
…
[ 164.363015] PVR_K: 1005: ------------[ PVR DBG: END ]------------
done
Archiving data ...................... done
File /tmp/pvrlogdump_2402262237.txt.gz was created.
```

4.4 将日志组添加到固件跟踪

GPU 驱动程序可以通过 pvrdebug 命令在固件日志中启用更多跟踪。可以在固件中启用各种日志组，并在调试期间就所需的组进行沟通。要启用更多（或更少）日志组，可以按如下方式使用该命令：

```
pvrdebug -loggroups main,mts,hwr
```

4.5 在硬件恢复后禁用驱动程序

由于固件的日志记录是在小缓冲区中完成的，因此 GPU 会在崩溃发生后继续覆盖固件日志。解决此局限性的一种方法是在 HWR 发生后立即冻结固件。以下命令会停止固件和日志记录以保存最后执行的命令：

```
echo "Y">/sys/kernel/debug/pvr/apphint/0/AssertOnHWRTrigger
```

4.6 禁用自动加载 GPU 驱动程序

有时，GPU 驱动程序在启动时需要额外的参数。这要求不要自动加载 GPU 驱动程序，以便能够手动启动该驱动程序。要将 GPU 驱动程序列入黑名单，请通过将以下行添加到 /etc/modprobe.d/blacklist.conf 来禁用 modprobe：

```
blacklist pvrsrvkm
```

在 SDK 8.6 及之前的版本（GPU 驱动程序 1.15 及更早版本）中，GPU 驱动程序会安装一个初始化脚本(/etc/init.d/rc.pvr)。可以通过将该脚本移出目录外来禁用此脚本，例如将该脚本移动到主目录。

要在列入黑名单后手动启动 GPU，只需使用 insmod 或 modprobe 插入 GPU 模块即可。

5 集成打补丁后的 GPU 驱动程序

TI 提供了打补丁后的显卡驱动程序来识别问题。为了了解显卡驱动程序的性质，本节概述了 GPU 驱动程序以及通常集成补丁程序的步骤。

GPU 驱动程序由两部分组成：用户模式库和驱动程序 (UM) 以及内核模式 (KM) 驱动程序。UM 为闭源代码，这意味着 TI 不会公开发布源代码。KM 为开放源代码，这意味着 TI 会将源代码存储在 SDK 中或公共存储库中。要在系统上安装每个组件，需要执行不同的步骤。

5.1 UM 库安装

UM 库是 TI 在面向公众的 `umlibs` 公共存储库 (https://git.ti.com/cgit/graphics/ti-img-rogue-umlibs/) 中打包的一组预先编译的文件。

这些库是为每个发行版预先编译的，编译系统会将这些库添加到生成的文件系统中。用户还可以通过手动复制这些库来更新文件系统，并且这些库已经分布在正确的文件结构中。以下命令仅供参考，并假设文件系统位于一张在 `/media/user/rootfs` 位置挂载了 `rootfs` 分区的 SD 卡中：

```
cd ti-img-rogue-umlibs
sudo cp -r targetfs/j721e_linux/wayland/release/* /media/user/rootfs
```

更新已完成。GPU 驱动程序与初始化脚本挂钩，并在系统引导时加载。

在调试期间，会以相同的方式提供包含错误修复的更新后的库。通过替换文件系统中的现有文件来安装更新。

5.2 KM 库安装

KM 驱动程序在 TI SDK 中的以下目录下提供：`ti-processor-sdk-linux-adas-j721s2-evm-09_02_00_05/board-support/extra-drivers/ti-img-rogue-driver-23.3.6512818`

该驱动程序还发布在 TI 的公共资源库中：https://git.ti.com/cgit/graphics/ti-img-rogue-driver/。

可以从 TI 的 SDK 编译基础架构中提取编译指令，但为了清晰起见，这里添加了编译说明。可以在 Shell 中设置*或者* 在 make 命令上传递的环境变量如下所示：

表 5-1. 编译 KM 驱动程序的环境变量

环境变量名称	值	说明
ARCH	arm64	目标 CPU 的架构
CROSS_COMPILE	aarch64-none-linux-gnu-	正在使用的交叉编译器
KERNELDIR	<系统上 Linux 内核的绝对路径>	Linux 内核源目录的路径
RGX_BVNC	36.53.104.796 — TDA4VL、TDA4VH、TDA4AEN、AM62P 22.104.208.318 — TDA4VM 33.15.11.3 — AM62x	要使用的 GPU 版本，特定的硬件版本
BUILD	释放或调试	要使用的编译配置文件
PVR_BUILD_DIR	<soc>_linux	要在驱动程序中使用的编译目录，每个 SOC 各一个目录
WINDOW_SYSTEM	lws-generic	要使用的 Windows 系统，在 SDK 9.0 之后仅支持 lws-generic。

设置好这些变量后，make 命令应该会起作用。将会在 SDK 中的 `ti-processor-sdk-linux-adas-j721s2-evm-09_02_00_05/board-support/extra-drivers/ti-img-rogue-driver-23.3.6512818/build/linux/j721s2_linux` 或 `ti-img-rogue-driver/build/linux/j721s2_linux`（如果已手动克隆）中进行编译。

要在目标文件系统上安装，SDK 会从 create binary 目录中运行以下 make 命令：
`binary_j721s2_linux_wayland_release/target_aarch64/kbuild`

```
make -C ${LINUXKERNEL_INSTALL_DIR} INSTALL_MOD_PATH=${DESTDIR} INSTALL_MOD_STRIP=${INSTALL_MOD_STRIP} M=`pwd` modules_install
```

如果是从 build 目录安装，则需要下列一个或多个环境变量： `DISCIMAGE=<path-to-rootfs-root-directory>`

示例：

```
export DISCIMAGE=/media/user/rootfs
sudo -E env PATH=$PATH make install
```

安装 KM 库的另一种方法是使用 output 目录中的 `install.sh` 脚本：

示例：

```
cd binary_j721s2_linux_lws-generic_release
sudo ./install.sh --root <path-to-rootfs-directory>
```

备注

如果您重新编译 Linux 内核，版本可能会发生变化，并且必须重新安装 GPU 内核驱动程序。

在重新编译 Linux 内核时，如果 Linux 内核版本发生变化，请务必检查是否重新编译并重新安装 GPU 内核驱动程序。最值得注意的是，当安装 Linux 内核模块时，会在文件系统中创建一个文件夹，例如 `/lib/modules/6.1.80-ti-g2e423244f8c0`。修改内核通常会引入后缀 `-dirty`，这会导致创建新的 modules 目录。因此，GPU 内核驱动程序也必须重新安装到新的内核模块目录中。

示例：为 AM62 手动编译 GPU 内核驱动程序

```
make \ ARCH=arm64 CROSS_COMPILE=/home/toolchains/arm-gnu-toolchain-11.3.rel1-x86_64-aarch64-none-linux-gnu/bin/aarch64-none-linux-gnu- \ KERNELDIR=/home//am625_sdk/ti-processor-sdk-linux-am62xx-evm-09.01.00.08/board-support/ti-linux-kernel-6.1.46+gitAUTOINC+247b2535b2-g247b2535b2 \
BUILD=release PVR_BUILD_DIR=am62_linux
```

示例：手动安装 GPU 内核驱动程序

```
cd binary_j721s2_linux_lws-generic_release
sudo install.sh -root <path to rootfs>
```

5.3 安装后步骤

现在，UM 和 KM 库均已更新完毕，系统准备好进行测试了。当 Linux 提示符出现时，通过运行 `dmesg | grep -i PVR` 来确定库是否已自动加载，并按如下方式查看输出：

```
root@j721s2-evm:~# dmesg | grep -i pvr
[ 5.085836] pvrsrvkm: loading out-of-tree module taints kernel.
[ 5.222004] PVR_K: 189: Read BVNC 36.53.104.796 from HW device registers
[ 5.415963] PVR_K: 189: RGX Device registered BVNC 36.53.104.796 with 1 core in the system
[ 5.672381] [drm] Initialized pvr 1.15.6133109 20170530 for 4e20000000.gpu on minor 0
[ 9.168474] PVR_K: 274: RGX Firmware image 'rgx.fw.36.53.104.796' loaded
[ 9.186869] PVR_K: 274: Shader binary image 'rgx.sh.36.53.104.796' loaded
```

注意 *RGX Firmware image * loaded* 这一行。这意味着 GPU 已加载固件。

如果您想要彻底检查 GPU 是否正在运行，请运行 GPU 驱动程序中包含的下面这个单元测试应用程序：

```
root@j784s4-evm:~# rgx_compute_test
------------------ RGX compute test -----------------
---------------------- Start ----------------------
Call PVRSRVConnectionCreateDevice with a valid
argument:

Connecting to first (0) default pvr
device
```

```
 OK
Create dev var context:
 OK
Looking up General heap handle
 OK
Getting event object
 OK
Creating robustness buffer
 OK
Mapping robustness buffer
 OK
Creating Compute Context
 OK
Creating Buffer
Creating DWord for CDM Event Object
...
Destroy Compute Context
 OK

Total time: 0ms
Disconnect from services:
 OK
------------------------ End ------------------------
```

这些结果表明 GPU 驱动程序正在正常运行。应用程序尝试访问显示器可能会导致更多问题，但这不在本文档的讨论范围内。此时，用户就知道了 GPU 驱动程序已启动并正在运行。

6 总结

导致出现图形问题的原因众多，从应用软件到硬件设计，不一而足。如果充分了解问题发生的原因并采取结构化的方法，则可以高效地隔离问题并找到根本原因。

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265