ENGINEERING AND R&D SERVICES

HCL

Android Graphics Power Consumption Optimization

WHITE PAPER

e) 2 Engineering experiences

Acceleration | Value | Technology

Table of Contents

Introduction

Smart phones and mobile devices are becoming more efficient, smart and powerful. It would be impressive if other consumer electronic devices like set top boxes (STBs) and televisions become as smart.

Android has matured a lot and plays a key role in making our mobile devices smarter. Mobile devices like cell phones with android show great performance with very little power consumption. But other consumer electronic devices like the set top box, with Android, have power and performance considerations to meet. Power consumption in such devices is an important issue, considering the advances in the processing capability of such devices. Network I/O intensive applications consume a lot of power due to the transmitting and receiving of signals, and graphic-intensive applications such as games also consume lot of power due to large GPU and CPU utilization.

Power optimization varies from system to system and is dependent on a number of factors like the OS version, HW components, CPU, GPU, Memory, and more. Apart from these limiting factors, there are some provisions available in the system, which lets you improve graphics power consumption of production-ready android consumer electronic systems, especially mobile STB devices. Much of the power consumption improvements suggested in this paper have been tested while running customized Android with Intel's mobile STB platform, with a focus on future products. Nonetheless, they should be applicable on products running Android on any system/device.

I also propose methods that can help reduce power consumption and improve performance for graphic intensive applications on consumer electronic devices, such as video player browsers and games.

Android Overview

Android is an OS based on Linux, designed primarily for mobile devices, but is also making its presence felt in various other consumer products. Android was unveiled in 2007 with the founding of the OHA and HTC Dream as the first publicly available smartphone.

The UI of Android is based on direct manipulation, using touch inputs that loosely correspond to real-world actions like swiping, tapping, pinching and reverse pinching to manipulate on-screen objects. Internal hardware such as accelerometers, gyroscopes and proximity sensors are used by some applications to respond to additional user actions. Android allows users to customize their home screens with shortcuts to applications and widgets.

Android's source code is released by Google under the Apache License which allows the software to be freely modified and distributed. Most Android devices ship with a combination of open source and proprietary software. Android is popular with technology companies who require a ready-made, low-cost and customizable operating system for high-tech devices. Despite being primarily designed for phones and tablets, it is also being used in TVs, game consoles, digicams and other consumer electronic products. Its open nature has encouraged the use of source code as a foundation for community-driven projects, which adds new features for advanced users to bring Android to devices which were officially released, running on other operating systems.

Market Trends: Android TVs and STB

Today, Android is one of the most popular operating systems for mobile and smart device applications and is well established to fulfill the nessesities and requirements in these areas. Recently, with the latest version of Android, Google has introduced support for TVs and other consumer electronic devices in Android Lollipop.

Consumer electronics devices have been using proprietary software from consumer electronic device manufacturers or a small number of middleware providers. There has been a big change in the use of Over-The-Top content (OTT) video services and other applications in the past few years. OTT service allows audio and video content to be delivered without the involvement of internet service providers, creating a hybrid environment of coventional and broadband content delivery. The world is accepting this environment, which is a combination of conventional content delivery mechanisms (Cable, Terrestrial or Satellite), and physical media and OTT, with open arms.

The already established middleware is well positioned to handle the conventional content delivery mechanism but doesn't have support for new standards like OTT. Device manufacturers take Android as a building block of this new technology because Android is built for mobile devices where the speed of innovation in the applications as well as content delivery mechanisms, is much faster than consumer electronic devices.

Other additional reasons for Android building momentum in the consumer electronics space are:

- Android provides device manufacturers with the ability to build an open or closed platform simultaneously. The virtue of openness of the Android SW developer platform invites developers to build applications, while at the same time allowing device manufacturers to define the hardware platform. This significantly reduces time to market.
- It is a very well established development platform with a big developer community.

Graphics Power Consumption Bottlenecks

SurfaceFlinger

SurfaceFlinger is one of the most important components of the Android Graphics system. SurfaceFlinger is used to accept buffers of data from multiple sources, compose and send them to the display. When an application comes to the foreground, the WindowManager service asks SurfaceFlinger for a drawing surface. SurfaceFlinger creates a layer and acts as the consumer for a/v buffers.

Many a times, it is seen that SurfaceFlinger takes more CPU than required. The optimal CPU usage by SurfaceFlinger is less than 5%. This is one of the factors for consuming extra power.

Hardware Composer
Hardware Composer (HWC) is used to determine the most efficient way to composite buffers with the available hardware. As a HAL, its implementation is device-specific and usually implemented by the display hardware manufacturer.

The HWC performs the following functions:

- SurfaceFlinger provides the HWC with a full list of layers and asks, "How do you want to handle this?"
- The HWC responds by marking each layer as "overlay" or "GLES composition"
- SurfaceFlinger takes care of any GLES composition, passing the output buffer to HWC, and lets HWC handle the rest

Many a times, it's seen that the HWC is not used and the composition is handled using OGLES. This will be more expensive and consume a lot of extra power.

Number of Displays and Primary Display configuration
There is more than one display used and configured in the system. SurfaceFlinger supports a "primary" display, i.e., what is built into a phone or tablet, and an "external" display, such as a television connected through HDMI. It also supports a number of "virtual" displays, which make composited output available within the system. Virtual displays can be used to record the screen or send it over a network.

In case of STBs, the main display would be HDMI, while touch screen primary display is not used. In the case of STBs, it is observed that even if touch screen displays are not utilized, it is configured as primary and HDMI would work as the external or secondary display. With this configuration, the power dissipation is more in comparison with a system without touch screen display configured or HDMI configured as primary. Sometimes, it is also seen that as the HDMI display is the secondry display, the hardware composition is not applied on the secondry display and has been enabled only for primary or touch screen display.

Power modes, Governor and GPU clock frequency
The system supports various GPU power modes and governors. The choice of the correct mode and governor is very important for graphics power and performance parameters. Graphics power and performance can be optimized with the proper mode selection of the graphics power island. In our board we have support for simple ondemand governor, powersaver governor and performance governor.

OnDemand Governor
This governor has a hair trigger for boosting clock speed to the maximum speed set by the user. OnDemand has excellent interface fluidity because of its high-frequency bias but it can also have a

relatively negative effect on battery life versus other governors. OnDemand is commonly chosen by smartphone manufacturers because it is well-tested, reliable and virtually guarantees the smoothest possible performance for the phone. This is because users are vastly more likely to desire performance than the few hours of extra battery life that another governor could have granted them.

Performance Governor

This locks the phone's CPU/GPU at maximum frequency. While this may sound like an ugly idea, there is growing evidence to suggest that running a phone at its maximum frequency at all times will allow a faster race-to-idle. Race-to-idle is the process by which a phone completes a given task, such as syncing email and returns the CPU/GPU to the extremely efficient low-power state.

Powersave Governor

The opposite of the performance governor, the powersave governor locks the CPU/GPU frequency at the lowest frequency set by the user. Proper selection of graphics clock frequency can also reduce the power consumption and significantly improve performance.

Screen Resolution and Sprite

Height and width, in case of smart phone devices, is differnt from that of TVs. Therefore, when porting Android for STB or TV systems, check the resolution, height and width properly. Similarily, check if the sprite (a graphic image that can move within a larger graphic) setting for HDMI is according to resolution. Sometimes, due to wrong height and width values, the driver doesn't get proper resolution value, which affects system performance and power consumption.

Hardware scaler

It is more power efficient to use a hardware scaler in an application with high resolution screen buffer. It is advisable to render to a fixed-size graphics buffer, rather than use system-provided default buffers, which are sized to the device's full screen to take advantage of the hardware scaler. When rendered to a fixed-size buffer, the device's hardware does the work of scaling scene up or down to match the device's screen resolution, including making any adjustments to the aspect ratio. Typically, we would create a fixed-size buffer that's smaller than the device's full screen resolution, which lets us render more efficiently, especially on high-resolution screens.

Using the hardware scaler is more efficient for several reasons. Firstly, hardware scalers are extremely fast and can produce great visual results through multi-tap and other algorithms that reduce artifacts. Secondly, because the app is rendering to a smaller buffer, the computation load on the GPU is reduced, and performance improves. Thirdly, with less computation work to do, the GPU runs cooler and uses less power. And finally, it is possible to choose the size rendering buffer, which can be the same on all devices, regardless of the actual screen resolution.

Error Gets

It might be possible that glGetError gets called every time we draw. Hence, it decreases performance drastically and affects power consumption.

Graphics Power Consumption Optimization

One way to minimize power consumption is to minimize the utilization of hardware components. Turning off devices and sensors that are not being used at the user level is one method to achieve this. Hardware manufacturers in the embedded systems domain are coming up with the latest processors which consume less power and can also operate on low voltages. Each and every device/component is analyzed/optimized to reduce the power consumption during development.

Another approach to optimize power consumption is software based. Programmers can optimize the power consumed by using software techniques, and efficient design and algorithms.

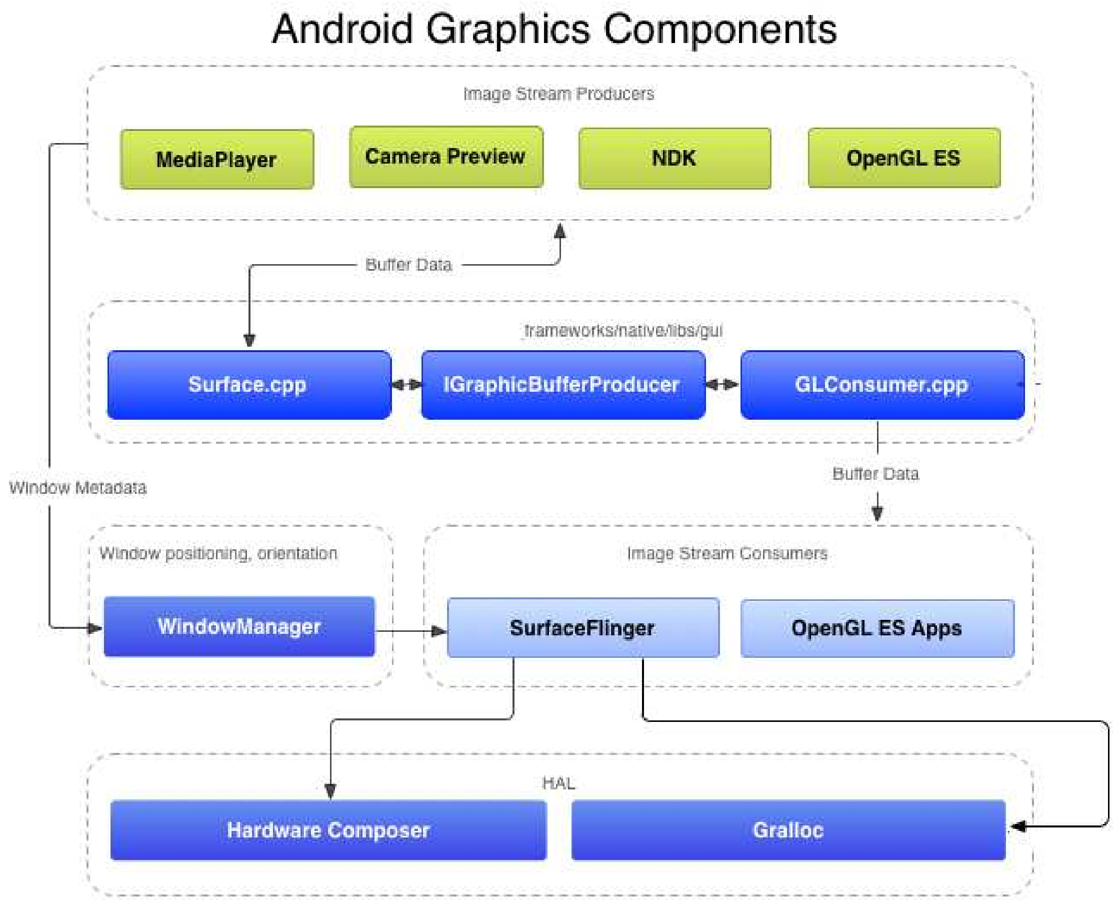


Figure 1: Android graphics components

Reference: https://source.android.com/devices/graphics/index.html

We have made changes in the software algorithm to optimize graphics power consumption and performance.

The different tools used for analysis include:

- Dumpsys SurfaceFlinger
- Monsoon Power Monitor
- PVRTune
- GLTrace
- DDMS and ADB Logcat

Suggestion I: SurfaceFlinger and Hardware Composer Optimization
SurfaceFlinger and HardwareComposer are the most important components of the Android Graphics system. Optimization of these components can save a lot of graphical power. To optimize these components, the programmer can use tools like dumpsys and PVRTune. With PVRTune the user can check the CPU utilization via SurfaceFlinger. Dumpsys provides important information on graphics layers and how these layers have been composed. If GLES is used for composition or hardware composer, it is utilized for composition. It will also provide information on different graphical windows. Taking initial readings from these tools, the programmer can take these readings as reference and then make the changes in the hardware composer and SurfaceFlinger modules, and then take the readings again. These optimizations are very important for saving power.

Apart from the method above, the programmer can also check the kernel configuration and turn some of the unused modules off, and see changes in power performance.

In our case, we have disabled the frame buffer support on primary display (which is not used in case of TVs). If frame buffer support on primary display is enabled, then framebuffer will automatically select the primary display device. Otherwise, the framebuffer console will always select the first framebuffer driver that is loaded in the device. The other change is related to vga arbitration and vga txt support. We have disabled both as we don't require these. After making these changes, we have seen improvement in performance and power readings.

Suggestion II: Hardware composer with GPU 2D
HwComposer is used to combine the specific surface layers supported by specific vendor devices. Devices use GPU 2D to combine most surface layers and the system power can be reduced with GPU 2D instead of GPU 3D. A typical power saving case is video playback. HwComposer with GPU 2D can offload a GPU 3D task when running games and benchmarks; it has been proven to improve the overall system performance by about 20%. The changes have been done in the HWC module.

Suggestion III: Swapping the Primary and External Display
Changes can be made in the display driver and hwc driver related to swapping of displays so that our HDMI display works as the primary display. Changes have been made in the modules below for swapping the primary display with the external display.

```
vendor/Intel/hardware/native/libmultidisplay
vendor/Intel/hardware/native/mfld_cdk
```

With the above two changes, we have verified that on both displays, primary as well as external HWC overlay is used instead of GLES for external display, where the CPU utilization from SurfaceFlinger reduced to around 5-7% which was around 12-13% previously. Power results are consistently around 600mW lesser than the reading before changes.

Suggestion IV: Changing the power governor and frequency
As per the data sheet, the minimum frequency at which the GPU can clock is at 200MHz, and the maximum is at 533MHz. The base frequency for better performance is 457 MHz. We have performed the analysis and tried to reduce the frequency with optimal performance and found the best possible results to be at 320MHz, and changed the governor from the simple_ondemand governor to the powersaver governor.
The changes have been done in pm_cmd_freq_set() function defined in the files below:
linux/kernel/drivers/external_drivers/intel_media/graphics/dfrgx/df_rgx.c
linux/kernel/drivers/external_drivers/intel_media/display/tng/drv/ospm/gfx_ospm.c
With the above changes, we see that the GPU's current frequency went down to 320 MHz from 457 MHz providing a power saving of around ~100mW.

Suggestion V: Modifications in OpenGL calls
OpenGL never signals errors but simply records them; it is a must to determine whether an error occurred. During the debugging phase, the program should call glGetError() to look for errors frequently (for example, once per redraw) until glGetError() returns GL_NO_ERROR. We found that glGetError gets called every time we draw. It is recommended calling glGetError after every gl call, but with a macro that is only defined when debugging.
http://www-f9.ijs.si/~matevz/docs/007-2392-003/sgi_html/ch15.html
During development, however, it's quite common to call glGetError. When the application is ready to go into production, make sure to remove glGetError calls and any other state getting and checking functions. For better usage we have taken the glGetError with OpenGL traces. If it's required to see the error logs, then anybody can enable the logs from user settings by going to:
Settings -> {} Developer Options -> Enable OpenGL Traces

Suggestion VI: Use Force GPU rendering
Force GPU usage for all drawing, composition and rendering. Graphics are handled in one of two ways, "software", which means the primary CPU does the heavy lifting, and "hardware" which means the GPU does the lifting. Hardware rendering is better, because it frees up CPU clock cycles for other activity, so the devices runs faster and smoother. This feature supposedly forces programs to use the GPU to paint 2D objects on the screen.
Settings -> {} Developer Options -> Force GPU Rendering

Suggestion VII: Change in height and width for Screen Resolution and Sprite
In case of TVs, the height and width are reversed from those of smartphones. Many times, when porting the Android to TV or STB platforms, the change in height and width value is not taken care of. We have made changes in height and width so that the driver selects the proper resolution.

```
// Intel Platform
   int hbW = 720;
   int hbH = 1280;

//Changed to
   int hbW = 1280;
   int hbH = 720;
```

Without changes, the code checks the width and sets the resolution based on it. With the width as 720, it cannot set the resolution properly.

References

1. Android graphics Architecture (Android developers Forum: https://source.android.com/index.html)
2. Whitepaper "HCL's capability in Android performance Optimization"

Appendix

Analysis Tools

dumpsys SurfaceFlinger

SurfaceFlinger and HWC analysis can be done with dumpsys tool. This tells what layers are on screen, whether they're being handled with overlays ("HWC") or OpenGL ES composition ("GLES"), and gives a bunch of other facts.

$adb shell dumpsys SurfaceFlinger

```
h/w composer present and enabled
Hardware Composer state (version  1030000):
mDebugForceFakeVSync=0
Display[0] : 720x1280, xdpi=254.000000, ydpi=254.000000, refresh=16666666
numHwLayers=3, flags=00000000
type    |  handle  |   hints  |   flags  | tr | blend |  format  |          source crop            |           frame
name
------------+----------+----------+----------+----+-------+----------+---------------------------------+-----------------------------
---
HWC | f858a9d0 | 00000002 | 00000000 | 04 | 00100 | 7fa00e00 | [    0.0,    0.0, 1280.0,  720.0] | [    0,
0,  720, 1280] SurfaceView
HWC | f8572320 | 00000002 | 00000000 | 00 | 00105 | 00000001 | [    0.0,    0.0,  720.0, 1280.0] | [    0,
0,  720, 1280] SurfaceView
FB TARGET | f8544510 | 00000000 | 00000000 | 00 | 00105 | 00000001 | [    0.0,    0.0,  720.0, 1280.0] |
[    0,    0,  720, 1280] HWC_FRAMEBUFFER_TARGET
Display[1] : 1920x1080, xdpi=304.000000, ydpi=304.000000, refresh=16666666   numHwLayers=3,
flags=00000000
```

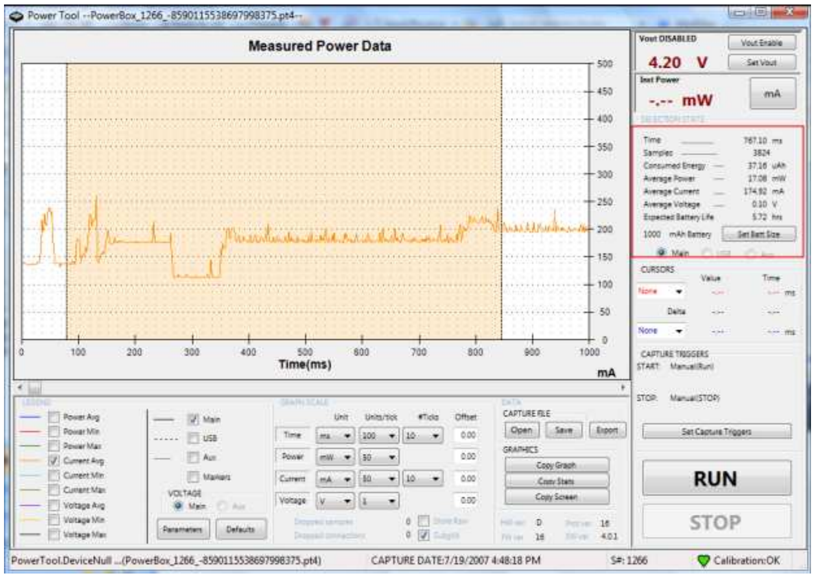
```
type    |  handle  |    hints  |    flags   | tr | blend |  format  |          source crop            |           frame
name
------------+----------+----------+----------+----+-------+----------+---------------------------------+-----------------------------
---
HWC | f858a9d0 | 00000002 | 00000000 | 00 | 00100 | 7fa00e00 | [    0.0,    0.0, 1280.0,  720.0] | [    0,
0, 1920, 1080] SurfaceView
GLES | f8572320 | 00000000 | 00000000 | 07 | 00105 | 00000001 | [    0.0,    0.0,  720.0, 1280.0] | [    0,
0, 1920, 1080] SurfaceView
FB TARGET | f854bd20 | 00000000 | 00000000 | 00 | 00105 | 0[    0.0,    0.0, 1920.0, 1080.0] | [    0,    0,
1920, 1080] HWC_FRAMEBUFFER_TARGET
```

After changes

```
h/w composer state:
h/w composer present and enabled
Hardware Composer state (version  1030000):
mDebugForceFakeVSync=0
Display[0] : 720x1280, xdpi=254.000000, ydpi=254.000000, refresh=16666666
numHwLayers=3, flags=00000000
type    |  handle  |    hints  |    flags   | tr | blend |  format  |          source crop            |           frame
name
------------+----------+----------+----------+----+-------+----------+---------------------------------+-----------------------------
---
HWC | f8918280 | 00000002 | 00000000 | 04 | 00100 | 7fa00f00 | [    0.0,    0.0, 1920.0, 1080.0] | [    0,
0,  720, 1280] SurfaceView
HWC | f893cda0 | 00000002 | 00000000 | 00 | 00105 | 00000001 | [    0.0,    0.0, 1080.0, 1920.0] | [    0,
0,  720, 1280] SurfaceView
FB TARGET | f88eee40 | 00000000 | 00000000 | 00 | 00105 | 00000001 | [    0.0,    0.0,  720.0, 1280.0] |
[    0,    0,  720, 1280] HWC_FRAMEBUFFER_TARGET
Display[1] : 1920x1080, xdpi=304.000000, ydpi=304.000000, refresh=16666666
numHwLayers=3, flags=00000000
type    |  handle  |    hints  |    flags   | tr | blend |  format  |          source crop            |           frame
name
------------+----------+----------+----------+----+-------+----------+-------------------------------+----------------------------
---
HWC | f8918280 | 00000002 | 00000000 | 00 | 00100 | 7fa00f00 | [    0.0,    0.0, 1920.0, 1080.0] | [    0,
0, 1920, 1080] SurfaceView
HWC | f893cda0 | 00000000 | 00000000 | 07 | 00105 | 00000001 | [    0.0,    0.0, 1080.0, 1920.0] | [    0,
0, 1920, 1080] SurfaceView
FB TARGET | f8946fa0 | 00000000 | 00000000 | 00 | 00105 | 00000001 | [    0.0,    0.0, 1920.0, 1080.0]
| [    0,    0, 1920, 1080] HWC_FRAMEBUFFER_TARGET
```

Monsoon Power Measurement Tool

The PowerTool software and the Power Monitor hardware provide a robust power measurement solution for mobile devices. They can analyze the power on any mobile device. After connecting the device, power up the Power Monitor and run the PowerTool software. The software should connect to the Power Monitor, and display the user interface (UI).


Power Tool --PowerBox_1266_-8590115538697998375.pt4--

Vout DISABLED
4.20 V
Inst Power
-.-- mW

Time	767.10 ms
Samples	3824
Consumed Energy	37.16 uAh
Average Power	17.08 mW
Average Current	174.92 mA
Average Voltage	0.10 V
Expected Battery Life	5.72 hrs
1000 mAh Battery	

CAPTURE TRIGGERS
START: Manual(Run)
STOP: Manual(STOP)

	Unit	Units/tick	#Ticks	Offset
Time	ms	100	10	0.00
Power	mW	50		0.00
Current	mA	50	10	0.00
Voltage	V	1		0.00

PowerTool.DeviceNull ...(PowerBox_1266_-8590115538697998375.pt4) CAPTURE DATE:7/19/2007 4:48:18 PM S#: 1266 Calibration:OK

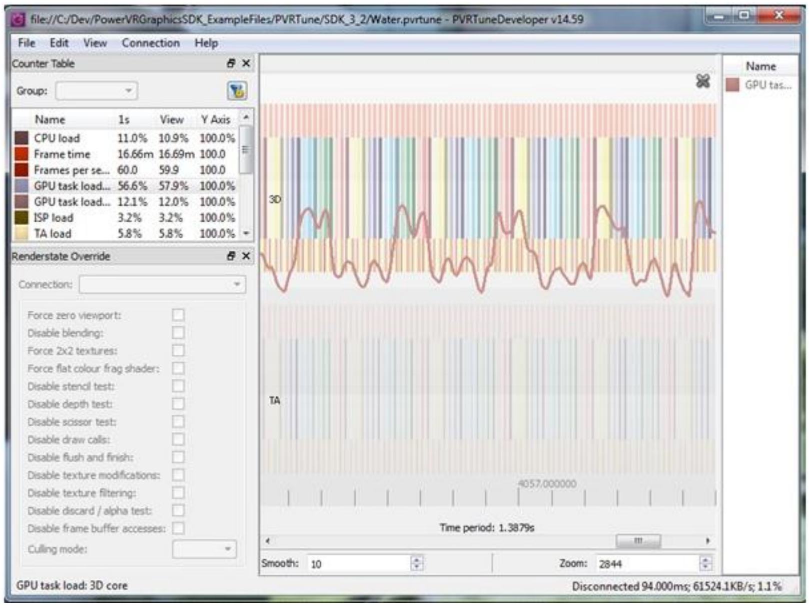
PVRTune

PVRTune is the PowerVR hardware performance analysis tool included in the PowerVR Graphics SDK. It consists of two parts: PVRTune and PVRPerfServer.

Prerequisites

A. PowerVR SDK v3.3 (added support for android kitkat) setup installed on the host machine.

B. Device running android with PowerVR GPU. Hummingbird requires using “input tap x y” for menu item selection.



file://C:/Dev/PowerVRGraphicsSDK_ExampleFiles/PVRTune/SDK_3_2/Water.pvrtune - PVRTuneDeveloper v14.59

Name	1s	View	Y Axis
CPU load	11.0%	10.9%	100.0%
Frame time	16.66m	16.69m	100.0
Frames per se...	60.0	59.9	100.0
GPU task load...	56.6%	57.9%	100.0%
GPU task load...	12.1%	12.0%	100.0%
ISP load	3.2%	3.2%	100.0%
TA load	5.8%	5.8%	100.0%

Force zero viewport: ☐
Disable blending: ☐
Force 2x2 textures: ☐
Force flat colour frag shader: ☐
Disable stencil test: ☐
Disable depth test: ☐
Disable scissor test: ☐
Disable draw calls: ☐
Disable flush and finish: ☐
Disable texture modifications: ☐
Disable texture filtering: ☐
Disable discard / alpha test: ☐
Disable frame buffer accesses: ☐
Culling mode:

Time period: 1.3879s

Smooth: 10
Zoom: 2844

GPU task load: 3D core
Disconnected 94.000ms; 61524.1KB/s; 1.1%

These are the steps to run PVRTune on a device for GPU profiling on a device running PowerVR GPU.

Steps:

A. **Install the PVRHub.apk** on device from
(/path/to/PowerVR_SDK/PVRHub/Android_armeabi_armeabi-v7a_x86_mips/PVRHub.apk)

```
$ adb shell install PVRHub.apk
```

This application is used to install PVRTrace recording libraries on the device.

B. **Flow of data from device to host PC**

- Connect the device to the same WiFi Network to which the host machine is connected.
- Run the above mentioned application on the device.
  ```
  # am start –n com.powervr.PVRHub/.MainActivity
  ```
- Clear warning message: "Warning – unable to get root access – Is Superuser.apk installed?"
  ```
  # input keyevent 4
  ```
  This warning message can be ignored.
- Start the application PVRPerfServer (this is for a screen size of 720x1080). Items grayed out are unavailable.
  ```
  # input tap 50 800
  ```
  For 1080p
  ```
  #input tap 50 700
  ```
- Now run PVRTuneDeveloper GUI application on Host.
- Connect the GUI via IP address. (IP automatically comes when PVRPerfServer starts running with the device connected to WiFi N/W)
- Now run any application that generates frames (ex: Water demo, GL-debug, Quadrant Benchmark) or get the OnCue application running.
- Analyze on PVRTune GUI.

- For further debugging you may collect the Logcat and Kernel traces.
- If you need to stop, and restart the application:
  # am force-stop com.powervr.PVRHub -> to stop

Author Info

Nitin Kumar Garg is a Senior Technical Lead at HCL Technologies Limited.