Architecture Challenges in the Android 3D Graphics Stack

Pau Baiget
Google UK
London, United Kingdom
pbaiget@google.com

ABSTRACT

The increasing traction of high-fidelity games on mobile devices is highlighting the challenges game developers have to face in order to optimize their content within the Android ecosystem.

In this talk, we'll explain our understanding of these challenges through the lens of how Android's graphics stack works today. If you've ever wondered:

- Are Android graphics drivers as buggy as I've heard?
- Why is there so much difference from device to device?
- Why aren't there great profilers like on console?
- Why can't I just measure draw call timings like on desktop?
- Why aren't graphics drivers updatable?

… then this talk is for you! We'll cover the way the hardware ecosystem for Android works, including the quirks of SOC vs. OEM vs. IP makers and how that translates to unique challenges. Then we will cover how software flows through this ecosystem and out through carriers, and the challenges that brings. We will talk about how the unique architecture features on mobile translate to new types of tooling challenges. Finally, we will talk about parallels between these combined challenges and other more traditional driver models from Windows or Mac, and discuss some of the implications thereof.

CCS CONCEPTS

• **Computing methodologies** → **Graphics processors**; • **Software and its engineering** → **Operating systems**; • **Human-centered computing** → **Mobile devices**.

KEYWORDS

android, graphics, gpu, profiling, game, 3d

ACM Reference Format:
Pau Baiget. 2019. Architecture Challenges in the Android 3D Graphics Stack. In *Proceedings of SIGGRAPH '19 Talks.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3306307.3328164

1 INTRODUCTION

The increasing capabilities or modern mobile GPUs has allowed game developers to create 3D games on phones that would be only imagined for console or PC just a few years ago. In fact, the High-Fidelity Game (HFG) tier is steadily gaining presence in the top game lists on all the mobile platforms (iOS and Android). These games make an intensive use of GPUs to produce high-quality 3D content and therefore developers need to fully optimize their rendering pipelines, specially to sustain a long and pleasant gaming experience. In the case of Android, its unique caracteristics makes this optimization phase a challenging journey. In the following sections we summarize some of the main challenges for HFG development on Android.

2 HARDWARE ECOSYSTEM ON ANDROID

Android is the operating system selected by many device manufacturers (OEMs) to produce phones and other mobile devices for all the existing price ranges (premium, mid-range, and low-end). This means the resources that an Android device can offer to an application can vary wildly in terms of memory, CPU, and GPU.

3 GRAPHICS DRIVERS ON ANDROID

GPU drivers can differ substantially from OEM to OEM, even though the underlying GPU would come from the same GPU vendor. Currently GPU vendors deliver their design to a silicon partner to be included into a System-on-Chip (SoC) along with a Device Driver Kit (DDK). The SoC partner may add some bits of proprietary code (e.g. a new pixel format) into the DDK to gain diferentiation. The SoC is then delivered to an OEM to produce the final device along with the modified DDK. The OEM may include additional bits for further diferentiation. As a result of this supply chain, the graphics drivers finally included in the device may differ substantially from those originally delivered by the GPU vendor.

In contrast to iOS[Apple [n.d.]] and Windows® [Microsoft [n.d.]b], Android currently does not impose a standard driver model to GPU manufacturers. Therefore there are no clear interfaces that allow to represent all these modifications in a modular manner. As a result, any changes to the driver (e.g. critical bug fixes or performance improvements) require traversing the aforementioned supply chain again. This process can take up to several months (some updates may even never happen), therefore making current driver updates very slow and unreliable for developers.

4 GPU PROFILING CHALLENGES ON MOBILE

Shifting from the console/PC to mobile platforms, game developers expect a high-quality tooling ecosystem around them. One important question is "How many resources is the GPU using?" (Profiling).

4.1 Driver Architecture

GPU profiling has historically been a complex problem since it involves dealing with GPU internal information, and hence requires

SIGGRAPH '19 Talks, July 28 - August 01, 2019, Los Angeles, CA, USA

ACM ISBN 978-1-4503-6317-4/19/07.
https://doi.org/10.1145/3306307.3328164

balancing Intellectual Property (IP) disclosure (for developers) with IP liability for their owners.

The problem becomes even harder on Android due to the lack of a well-established driver architecture. In comparison, Windows® has WDDM [Microsoft [n.d.]c] which defines a job submission interface (ie. *command streams*) to be implemented by all desktop GPUs. By knowing that, a profiler can assume the work is submitted in a well defined way. However this assumption is not valid on Android, where some GPUs do not use a compatible programming model, which makes expressing their performance characteristics even harder.

4.2 Tile-based GPUs

Mobile GPUs use a tile-based architecture[Arm 2016][ImgTec 2015] in order to save bandwidth (and thus reduce power consumption). Such architectures present additional issues for GPU profiling, since measuring time spent on one draw call (a heavily demanded use case by game developers) cannot easily be mapped to a tiler.

4.3 Memory management

Memory management constitutes another challenge for profiling on Android. Currently, memory is managed by vendor code (code outside the Android platform) and custom patches to the Android kernel. This means there is no standard way of bookkeeping GPU memory usage for profiling purposes.

4.4 Non-standard buffer display

Even simple things like displaying a buffer (*swapbuffers*[Microsoft [n.d.]a]) is a bit more complex on Android. EGL[Khronos [n.d.]] is provided by the vendor implementation, which then makes a private call back to the Android compositing system for final buffer display. This means that actual performance characteristics of *swapbuffers* is highly vendor dependent on Android, and can even vary from phone to phone. As a result, profiling of simple things like "why am i not 30fps" becomes tricky.

5 CONCLUDING REMARKS

In this talk we presented the current Android graphics stack and described some of the challenges it poses game developers when optimizing 3D-intensive games on Android. Inside the Android Graphics organization we are working on overcoming these issues by improving the graphics tool ecosystem on Android.

REFERENCES

Apple. [n.d.]. iOS. Retrieved February 1, 2019 from https://www.apple.com/ios/home

Arm. 2016. Tile-based Rendering. Retrieved February 1, 2019 from https://developer.arm.com/graphics/developer-guides/tile-based-rendering

ImgTec. 2015. A look at the PowerVR graphics architecture: Tile-based rendering. Retrieved February 1, 2019 from https://www.imgtec.com/blog/a-look-at-the-powervr-graphics-architecture-tile-based-rendering

Khronos. [n.d.]. EGL: Native Platform Interface. Retrieved February 1, 2019 from https://www.khronos.org/egl

Microsoft. [n.d.]a. SwapBuffers function. Retrieved February 1, 2019 from https://docs.microsoft.com/en-us/windows/desktop/api/wingdi/nf-wingdi-swapbuffers

Microsoft. [n.d.]b. *Windows*. https://www.microsoft.com/en-gb/windows

Microsoft. [n.d.]c. Windows Display Driver Model. Retrieved February 1, 2019 from https://docs.microsoft.com/en-us/windows-hardware/drivers/display