# TEAPOT: A Toolset for Evaluating Performance, Power and Image Quality on Mobile Graphics Systems

Jose-Maria Arnau
Computer Architecture Department
Universitat Politecnica de Catalunya
jarnau@ac.upc.edu

Joan-Manuel Parcerisa
Computer Architecture Department
Universitat Politecnica de Catalunya
jmanel@ac.upc.edu

Polychronis Xekalakis
Intel Labs
Intel Corporation
polychronis.xekalakis@intel.com

## ABSTRACT

In this paper we present TEAPOT, a full system GPU simulator, whose goal is to allow the evaluation of the GPUs that reside in mobile phones and tablets. To this extent, it has a cycle accurate GPU model for evaluating performance, power models for the GPU, the memory subsystem and for OLED screens, and image quality metrics. Unlike prior GPU simulators, TEAPOT supports the OpenGL ES 1.1/2.0 API, so that it can simulate all commercial graphical applications available for Android systems.

To illustrate potential uses of this simulating infrastructure, we perform two case studies. We first turn our attention to evaluating the impact of the OS when simulating graphical applications. We show that the overall GPU power/performance is greatly affected by common OS tasks, such as image composition, and argue that application level simulation is not sufficient to understand the overall GPU behavior. We then utilize the capabilities of TEAPOT to perform studies that trade image quality for energy. We demonstrate that by allowing for small distortions in the overall image quality, a significant amount of energy can be saved.

## Categories and Subject Descriptors

C.1.4 [**Parallel Architectures**]: Mobile Processors; C.4 [**Performance of Systems**]: Modeling techniques

## Keywords

Mobile GPU; low-power graphics; simulation infrastructure

## 1. INTRODUCTION

Mobile devices such as smartphones and tablets have been widely adopted in recent years, emerging as one of the most rapidly spread technology [14]. At the same time, mobile user interfaces have evolved from simple text-based displays to a point where interactive high definition 3D graphics is becoming the standard [11]. Mobile software excels in providing very rich user interfaces powered by graphics hardware, including a plethora of 2D/3D games, applications for maps or web browsing, and even hardware acceleration of the desktop [21]. Not surprisingly, previous studies have identified the GPU and the screen as the main battery consumers on a smartphone for many use cases [1, 3]. As the demand for more visually compelling graphics on mobile devices continues to grow, understanding the performance and power aspects of the graphics subsystem grows in importance.

Prior work on GPU simulation tools [2, 4, 10, 13] is mainly focused on accurately modelling desktop-like GPUs. Unfortunately, these simulators are not tailored towards the mobile segment. In fact, to the best of our knowledge, none of them have support for the OpenGL ES API [29], which means that they cannot run smartphone applications. Furthermore, most of them do not provide a power model. This is an important impediment, as all of the mobile segment devices are battery operated and thus power consumption is key design aspect. Moreover, none of the existing simulators provide energy estimations for the OLED screens. This is an important aspect of graphics simulation as the screen is one of the main battery consumers and its energy depends on the output generated by the GPU [5].

Full system GPU simulation is another important benefit provided by our simulator, that is not available in any existing GPU simulation infrastructure. This means that frequent GPU-related tasks such as image composition [21], or rendering of background applications (e.g., advertisements), are usually not taken into consideration. As we will show in a later section, this accounts for non-negligible discrepancies in the simulated use of the GPU. More specifically, we show that the OS GPU usage is non-negligible, representing up to 52% of GPU time and up to 48% of GPU energy for a set of commercial Android games.

In this paper we present TEAPOT, a toolset for evaluating the performance, energy and image quality of a mobile graphics subsystem. TEAPOT provides full-system simulation of Android applications. It includes a cycle-accurate GPU simulator, a power model for mobile GPUs, based on McPAT [8], and a power model for OLED screens [5]. Furthermore, it is able to provide image quality assessment using the models presented in [17].

In terms of the GPU microarchitecture, TEAPOT models both Tile-Based Deferred Rendering (TBDR) [35] and Immediate-Mode Rendering (IMR) [10]. While IMR is more popular for desktop GPUs, for GPUs targeting energy efficiency TBDR seems to be the design of choice (ARM Mali [24] and PowerVR [33] are TBDR). Prior work only focused on the IMR approach, which as it will be shown in Section 2.3.1 is significantly different from TBDR both in terms of performance and power.

This paper focuses on GPU simulation tools. Its main contributions are the following:

- We describe TEAPOT, a new toolset for analyzing mobile graphics systems in 4 dimensions: GPU execution time, GPU energy, OLED screen energy and image

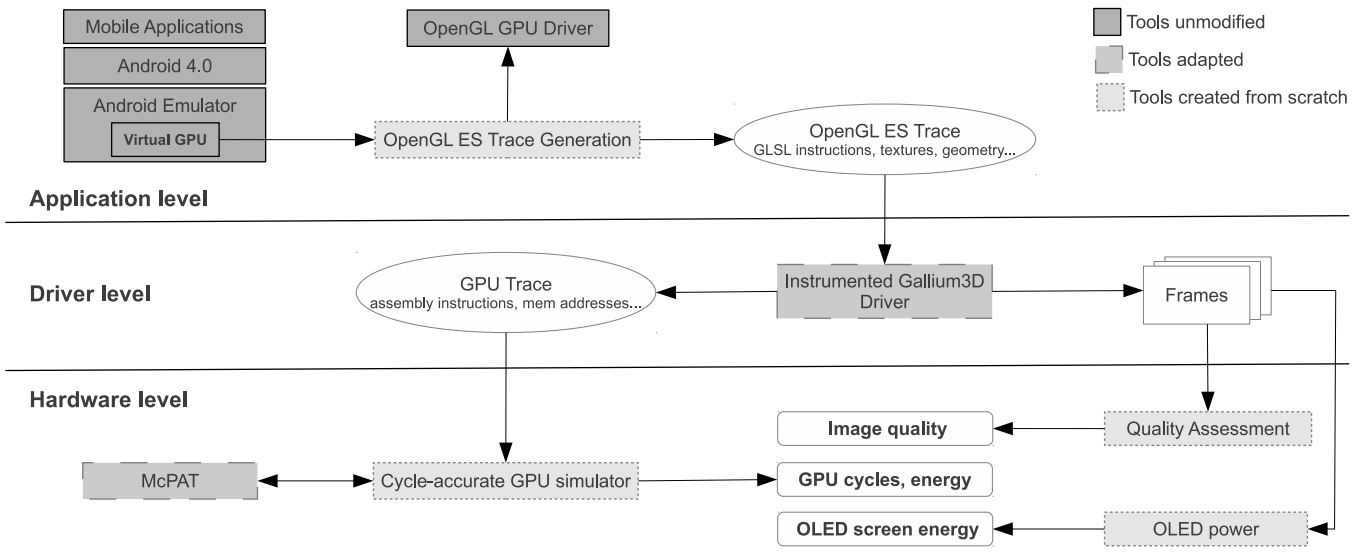



Figure 1: Mobile GPU simulation infrastructure.

quality.

- We show how TEAPOT can be employed to evaluate energy saving techniques that trade quality for energy. Our results demonstrate that up to 90% of GPU energy can be saved by reducing screen resolution, losing less than 13% image quality for a set of commercial Android games.

- We argue that full-system GPU simulation is crucial to achieve accurate results, since common OS tasks such as image composition consume 26% of GPU time and 24% of GPU energy on average for a set of Android games.

The remainder of this paper is organized as follows: The next section describes TEAPOT. In Section 3 we present our evaluation methodology. In Section 4 we illustrate the benefits of using full-system GPU simulation, analyzing the OS GPU usage. In Section 5 we evaluate different energy saving techniques that trade quality for energy. Section 6 contains a discussion of related work on GPU simulation tools. Finally, in Section 7 we provide some conclusions.

# 2. SIMULATION INFRASTRUCTURE

TEAPOT is a set of tools that can be used for evaluating mobile graphics systems. Figure 1 illustrates the overall infrastructure. TEAPOT leverages existing tools (e.g, McPAT or Gallium3D) that have been coupled with our GPU models and adapted for the low-power segment. The goal of TEAPOT is to drive the evaluation of new energy saving techniques for low power graphics.

TEAPOT is able to run and profile unmodified commercial Android applications. We have modified the Gallium3D driver in order to profile OpenGL ES commands and collect a complete GPU instruction and memory trace. This trace is then fed to our cycle-accurate GPU simulator with which we estimate the power and performance for the given application.

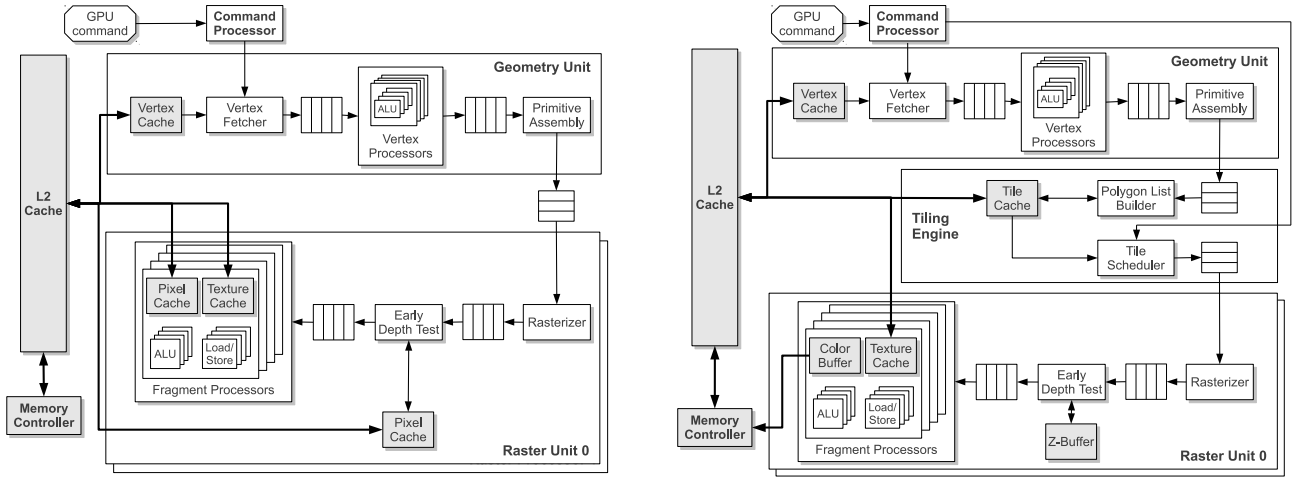
As mentioned earlier, TEAPOT also includes a power model for OLED screens. This is crucial for mobile devices as the energy consumed by the screen is a significant part of the overall energy, while it also depends on the images rendered by the GPU. Therefore, the tasks performed by the GPU driver and the GPU can significantly affect the energy consumed by the OLED screen. Thus, when analyzing an energy saving technique for mobile GPUs we must ensure that possible energy savings in the GPU are not compensated by an increment in screen energy and vice versa.

TEAPOT provides several image quality metrics, based on per-pixel errors or based on the human visual perception system. These metrics are useful when evaluating aggressive energy saving techniques that trade image quality for energy. Significant energy savings can be achieved by allowing small distortions on the output images, as shown in Section 5, and the visual quality metrics are necessary for evaluating the magnitude of the distortion. The next sections illustrate the workflow of the simulation infrastructure and provide more insights into the components of TEAPOT.

## 2.1 Application Level

TEAPOT uses the Android Emulator available in the last version of the Android SDK [18] for running mobile applications on a desktop computer. The Android Emulator is based on QEMU [34] and supports GPU virtualization [37]. Hardware acceleration is thus available for the guest Operating System running inside the emulator, Android in that case. When enabling GPU virtualization, the OpenGL ES commands issued by the applications are redirected to the GPU driver of the desktop machine. Since OpenGL ES is hardware accelerated, state-of-the-art 3D games run at high frame rates on the emulator. This also simplifies the GPU profiling since the OpenGL ES commands are not processed inside the emulator but they are redirected to the desktop GPU driver and, hence, they are completely visible to the host system.

The *OpenGL ES trace generator* component captures the OpenGL ES command stream generated by the Android ap-




(a) Immediate Mode Renderer.

(b) Tile-Based Deferred Renderer.

Figure 2: Mobile GPU architectures implemented in the simulator. Tile-Based Deferred Renderers are also referred in the literature as sort-middle architectures [9], whereas Immediate Mode Renderers are also named sort-last fragment [6].

plications. It saves the GPU commands in a trace file, and redirects them to the appropriate GPU driver. It consists on a library interposed between the Android Emulator and the desktop GPU driver that contains replacements for all the functions in the OpenGL ES API, so when a graphical application calls some OpenGL ES function, the one in the trace generator is selected instead of the one in the GPU driver. Hence the *OpenGL ES trace generator* is completely transparent for the emulator and it just causes a small frame rate decrease due to the time necessary for logging and redirecting the GPU commands.

The OpenGL ES trace file contains the GLSL vertex and fragment shaders [30], i. e. the code executed by the GPU, and all the data employed for rendering including textures, geometry and state information. Therefore, it contains all the necessary data for reproducing the OpenGL ES command stream. Furthermore, the thread identifier is stored in the trace together with each OpenGL ES command, so the cycle-accurate simulator can report per-thread statistics. Note that the *OpenGL ES trace generator* captures commands not from just one application but from all the applications concurrently using the GPU, including the Android OS, so TEAPOT provides full-system GPU simulation.

## 2.2 Driver Level

The Gallium3D [22] driver provides GPU functional emulation in TEAPOT. Gallium3D is an infrastructure for developing GPU drivers. It includes several front-ends for different graphics APIs, including OpenGL ES, and multiple back-ends for distinct GPU architectures. A modified version of Gallium3D is employed for executing the commands stored in the OpenGL ES trace file. A software-based backend is selected for rendering since it can be easily instrumented in order to get a complete GPU instruction and memory trace.

Gallium3D translates the high level GLSL shaders into an intermediate assembly code, TGSI [36]. The software backend of Gallium3D consists of an emulator for this TGSI assembly language. By instrumenting the TGSI emulator all the instructions executed by the GPU and all the memory addresses referenced are collected and stored in a GPU trace file. Note that a software renderer is different from real hardware, so special care is taken in order to trace just the addresses that would be issued in a real GPU. On the other hand, off-screen rendering is employed so the frames are written into an image file instead of being displayed on the screen.

## 2.3 Hardware Level

A cycle-accurate GPU simulator is employed for estimating the GPU execution time taking as input the instruction and memory traces generated by Gallium3D. GPU energy estimations are also provided by using a modified version of McPAT. The output frames generated by Gallium3D are then used for computing the energy consumed by the OLED screen. Finally, the image quality assessment module estimates the visual quality of the output images.

### 2.3.1 Cycle-Accurate GPU Simulator

Our GPU simulator is able to model low-power GPUs based on conventional Immediate Mode Rendering (IMR), such as the GeForce inside the NVIDIA Tegra 2 SoC [19], and Tile-Based Deferred Rendering (TBDR), such as the ARM Mali [24]. Figure 2a illustrates the IMR model modeled in TEAPOT. First, the *Command Processor* receives a command from the CPU and it sets the appropriate control signals so the input triangle stream is processed through the pipeline. Next, the *Geometry Unit* converts the input world-space triangles into a set of transformed 2D screen-space triangles. Finally, the *Raster Unit* computes the color of the pixels overlapped by the triangles. This architecture is called "Immediate Mode" because once a triangle has been transformed it is immediately sent down the graphics pipeline for further pixel processing. The main problem of this architecture is *overdraw*: the colors of some pixels are written multiple times into memory because of pixels from younger triangles replacing pixels from previously processed triangles.

In TBDR, shown in Figure 2b, the screen is divided into tiles, where a tile is a rectangular block of pixels. Trans-

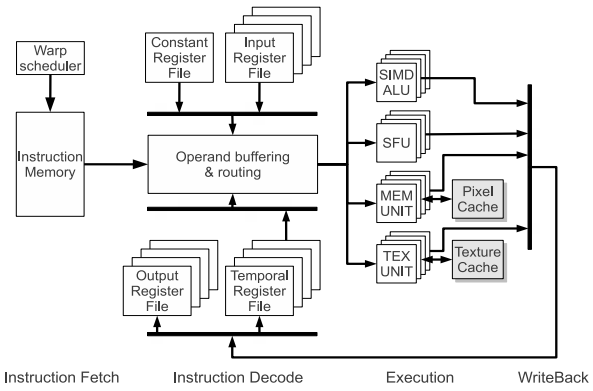



Figure 3: Fragment processor architecture. Vertex processors are similar but they do not include Pixel/Texture Caches and MEM/TEX units.

formed triangles are not immediately sent to the *Raster Unit*. Instead, the *Tiling Engine* stores the triangles in memory and sorts them into tiles, so that for each tile that a triangle overlaps a pointer to that triangle is stored. Once all the geometry for the frame have been fetched, transformed and sorted, the rendering starts. Just one tile is processed at a time in each *Raster Unit*, so all the pixels for the tile can be stored in local on-chip memory and they are transferred just once to the off-chip color buffer in system memory when the tile is ready, avoiding the *overdraw*. However, transformed triangles have to be stored in memory and fetched back for rendering, so there is a trade-off between memory traffic for geometry and memory traffic for pixels. TBDR is becoming increasingly popular in the mobile GPU segment as shown in Table 1.

Regarding the memory hierarchy, the mobile GPU employs several first level caches for storing vertices, pixels or textures. These caches are connected through a bus to a second level shared cache. In TBDR, local on-chip memories are employed for storing all the pixels for a tile. This information is not cached in the L2 but it is directly transferred to main memory.

Figure 3 shows the architecture for the programmable units of the GPU pipeline, the vertex and fragment processors. It consists of a fairly simple 4-stage in-order pipeline. A form of SMT is employed to hide the memory latency by interleaving the execution of several warps. A warp is a group of threads executed in lockstep mode, that is the same instruction is executed by all the threads but each thread operates on a different fragment/vertex. The *Warp Scheduler* determines from which warp to fetch an instruction by using a Round Robin policy. Next, the source operands are fetched from the register file. Each thread has available a set of input registers, for storing input parameters such as screen coordinates or texture coordinates, a set of temporal registers, for intermediate computations, and a set of output registers for storing the result of the fragment program. Furthermore, a set of constant registers is shared by all the threads, these registers contain global state information such as the number of lights enabled. Four types of functional units are included in each Fragment Processor: the SIMD ALU, the Special Functions Unit (reciprocal, square root), the Memory Unit and the Texture Unit (computes the color of a texture at a given coordinates). The

Table 1: Unlike in the desktop segment, TBDR has a remarkable presence in the mobile GPU market.

| Manufacturer | Chipset | GPU Architecture |
|---|---|---|
| NVIDIA | Tegra | Immediate-Mode [19] |
| ARM | MALI | Tile-Based [24] |
| Imagination | PowerVR | Tile-Based [33] |
| Qualcomm | Adreno | Hybrid [31] |

warps that are stalled waiting for a long latency operation are marked as blocked and are not selected for execution by the *Warp Scheduler*. In the last pipeline stage the result of the instruction is written in the temporal or in the output register file. No forwarding mechanism is present. However, consecutive instructions usually belong to different warps since Round Robin is employed.

Vertex Processors are similar to Fragment Processors, but they do not have to handle memory or texture instructions so they do not include Memory units, Textures Units or Pixel/Texture caches. We assume a non-unified architecture as opposite to a unified architecture where all the processors can handle both vertices and fragments. Usually, unified architectures offer better workload balance, whereas non-unified architectures can exploit the difference between vertex and fragment processing to build more specialized and optimized processors. For instance, the results obtained by using McPAT indicate that a Vertex Processor has just 64% of the area of a Fragment Processor. TEAPOT reports statistics per-frame and per-thread. Since GPU commands are tagged with the thread identifier, TEAPOT is able to assign fractions of GPU execution time and energy to each application.

### 2.3.2 Power Model

A modified version of McPAT [8] is used for estimating GPU energy consumption. During start-up, the GPU simulator calls to McPAT passing all the microarchitectural parameters, such as the number of processors or the cache sizes, so it can build the internal chip representation. McPAT estimates the dynamic energy required to access each one of the hardware structures and the leakage power. During simulation, the cycle-accurate GPU simulator collects statistics for each unit and, at the end of every frame, it submits all the activity factors to McPAT. The dynamic energy is computed by accounting for events in the GPU simulator and then multiplying these events by a given energy cost estimated by McPAT. The static energy is obtained by multiplying the total GPU leakage by the execution time. McPAT has been slightly modified so it can better model a low-power GPU. For instance, McPAT allows just for one private data cache for each core, whereas the Fragment Processors contain two data caches (Pixel and Texture caches). Furthermore, data caches are considered to be read/write, whereas Texture caches are read-only so they do not need some hardware structures such as the write-back buffer. Finally, we have extended McPAT in order to model Texture sampling units. The texture sampler is implemented by using a combination of Load units, for fetching the texels (texture pixels) from memory, and FP units, for applying the texture filtering.

On the other hand, TEAPOT employs the power model presented by Dong et. al [5] for estimating the energy con-

Table 2: Android games employed for the experiments.

| Game | Description |
|---|---|
| angrybirds | 2D puzzle game |
| badpiggies | 2D puzzle game |
| captainamerica | 3D beat'em up game |
| chaos | 3D helicopter simulator |
| crazysnowboard | 3D snowboarding simulator |
| hotpursuit | 3D racing game |
| ibowl | 3D bowling game |
| sleepyjack | 3D action game |
| templerun | 3D arcade game |

Table 3: General purpose mobile applications.

| Application | Description |
|---|---|
| anatomy | Interactive 3D human anatomy model |
| citysurf | 3D GIS/Mapping System |
| desktop | Android desktop with 3D widgets |
| droidiris | Image browser |
| facebook | Android facebook client |
| globaltime | 3D world clock |
| k9mail | E-mail client |

sumed by the OLED screen. OLED-based displays consume significantly different energy when displaying different colors due to their emissive nature. This characteristic creates a dependence between the GPU and the screen, since the energy consumed by the display depends on the images generated by the graphics hardware. The OLED power model provides three functions f(R), f(G) and f(B) that map pixel intensity into energy consumption for the red, green and blue components of a pixel based on [5].

#### 2.3.3 *Image Quality Assessment*

Generating high quality images comes at the cost of significant energy consumption which sometimes is not desirable in the mobile segment, specially for low-battery conditions. Significant energy savings can be achieved by allowing small distortions on image quality, as shown in Section 5. When trading quality for energy we need some way of evaluating the magnitude of the visual quality decrease. To this extent, TEAPOT provides several metrics for automatic image quality assessment. Image quality is evaluated by comparison with a reference image, usually as a result of a high quality rendering. The error is then estimated by comparing the high quality image with the distorted image. Two types of metrics are typically employed for image quality assessment, ones based on per-pixel errors and the other based on the human visual perception system.

Regarding the metrics based on per-pixel errors, TEAPOT implements the MSE (Mean Squared Error) [25] and the PSNR (Peak Signal-to-Noise Ratio) [32].

TEAPOT also includes the MSSIM (Mean Structural SIMilarity Index) presented in [17], a metric based on the human visual perception system. This metric is more desirable since the images generated by the GPU are interpreted by users. Hence, an error in a pixel is a problem just if it can be perceived by the human visual system, i. e. if it causes a degradation of structural information, since the human visual perception system is highly adapted for extracting structural information from a scene. An MSSIM value of 100% means perfect image fidelity, whereas a value of, for example, 90% indicates 10% of perceivable differences between the reference image and the distorted image. The original MSSIM metric only works on gray-scale images, but it can be adapted for RGB format [12].

Table 4: GPU parameters employed for the experiments. We model mobile GPUs based on Immediate Mode Rendering and Tile-Based Deferred Rendering.

| Parameter | Value |
|---|---|
| Architecture | IMR, TBDR |
| Frequency | 600 Mhz |
| Num. of Vertex Procs. | 4 |
| Num. of Frag. Procs. | 4 |
| Num. of threads per Proc. | 16 (4 warps, 4 threads/warp) |
| Tile Cache | 32 KB, 4-way, 64 bytes/line |
| L2 Cache | 128 KB, 8-way, 64 bytes/line |
| Pixel/Texture Caches | 8 KB, 2-way, 64 bytes/line |
| Vertex Cache | 16 KB, 2-way, 64 bytes/line |
| Main memory | 1 GB, 100 cycles latency, 64 bytes/line |
| Tile Size | 16 x 16 pixels |

## 3. EVALUATION METHODOLOGY

In order to illustrate how TEAPOT can be employed for analyzing mobile graphics systems we have generated traces for several Android applications. We have first focused on games since they represent an important segment of the market and games are applications that typically stress the GPU significantly. Table 2 shows all the games we have selected. We have tried to provide a good representation of commercial smartphone games, covering both simple 2D games and more complex 3D games. We have also included a set of general purpose applications shown in Table 3. Again, we have covered a wide range of commonly used applications for smartphones, including maps, a mail client, an image browser or the Android desktop. Although general purpose applications do not tend to be so GPU intensive as games, they also rely on hardware accelerated graphics for displaying very rich user interfaces. Table 4 shows the GPU configuration employed for the experiments. We have set up the timing simulator to model an Immediate-Mode Renderer similar to that of the NVIDIA Tegra 2 chipset [19] (shown in Figure 2a), and a Tile-Based Renderer similar to the ARM Mali GPU [24] (shown in Figure 2b).

### 3.1 Performance characterization

We have conducted several experiments to measure the overhead introduced by our instrumentation in the Android Emulator and Gallium3D. Furthermore, we have analyzed the slowdown caused by the GPU functional and timing simulators included in TEAPOT, compared to the performance offered by a mobile GPU.

Figure 4 shows the performance slowdown produced by the *OpenGL ES Trace Generator* of TEAPOT. The numbers were collected on an Intel i5-3210M 2.5 GHz, with 4 GB of main memory and an Intel HD Graphics 4000 GPU. The generation of the OpenGL ES trace produces a slowdown between 4% (*crazysnowboard*) and 20% (*sleepyjack*), with an average overhead of just 10% for our set of commercial

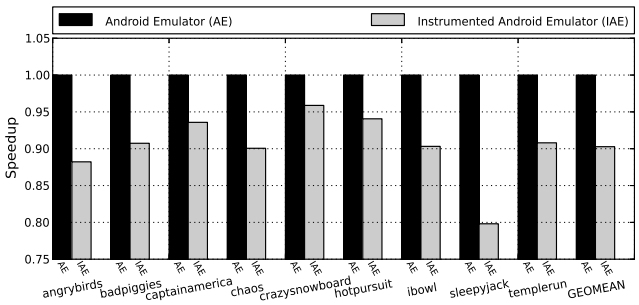



Figure 4: Overhead introduced in the Android Emulator. The *OpenGL ES Trace Generator* of TEAPOT produces a slowdown of 10% on average for our set of commercial Android games. Hence, the frame rate is not drastically reduced and games are still playable when generating traces.

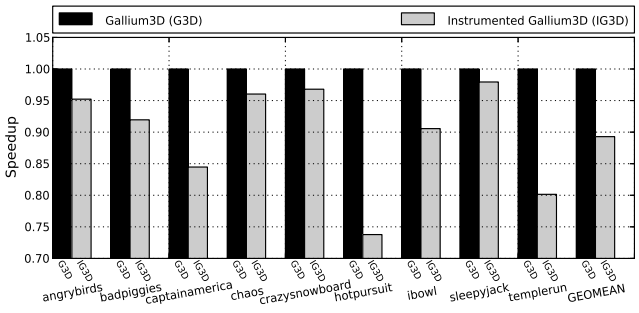



Figure 5: Overhead introduced in the Gallium3D *softpipe* driver. The instrumentation of the software renderer causes a slowdown of 11% on average in our set of commercial Android games.

Android games. Hence, the games are still playable in the Android Emulator despite all the OpenGL ES commands are saved in a trace file.

Figure 5 shows the overhead introduced by the instrumentation of the Gallium3D software renderer. Frames are rendered at WVGA resolution (800x480). The generation of the GPU instruction and memory traces causes a slowdown between 2% (*sleepyjack*) and 26% (*hotpursuit*), with an average overhead of 11% for our set of Android games.

Finally, Figure 6 shows the performance of the GPU functional simulator, the Gallium3D *softpipe* driver, and the performance of the timing simulator compared with a real GPU. We assume a frame rate of 40 FPS for the comparison, which is common on current mobile GPUs. The simulations were executed on a cluster where each node features an Intel Xeon E5-2630L CPU and 64 GB of RAM. The software renderer, Gallium3D, is an order of magnitude slower than real hardware. On the other hand, the GPU timing simulator causes an overhead between four and five orders of magnitude with respect to a real mobile GPU, for both architectures IMR and TBDR. These slowdowns are common for cycle-accurate microarchitectural simulations.

## 4. ANDROID GPU USAGE

Image composition is the process of merging the output surfaces from multiple sources together into a single surface, where a surface is a memory area to hold an image.

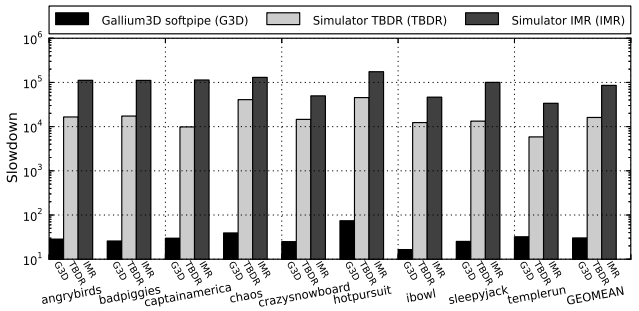



Figure 6: Slowdown of the functional simulator, Gallium3D, and the timing simulators for IMR and TBDR. We assume a frame rate of 40 FPS for the real GPU hardware.

In a multi-tasking mobile device many of the activities taking place simultaneously generate images for display on the screen. One of the components of Android, the *SurfaceFlinger*, takes care of merging all the application surfaces into a final image that is sent to the display. Composition is a compute intensive task that involves complex operations such as scaling, rotation, handling transparencies or color space conversions [20]. The *SurfaceFlinger* utilizes OpenGL ES to boost this process via hardware accelerated graphics. Using the GPU for image composition significantly improves the performance of the *SurfaceFlinger*, making it possible to display complex user interfaces at high frame rates in a smartphone. However, it can potentially reduce the performance of the graphical applications since the same GPU is employed for both rendering and composition.

Figure 7 shows the overhead of hardware accelerated image composition for several Android games. The graph depicts the percentage of GPU time and energy devoted to rendering the game and the image composition. *SurfaceFlinger* represents up to 41% of the total GPU execution time, with an average of 26% for a GPU based on IMR. Regarding the energy, *SurfaceFlinger* accounts for a non-negligible 24% of the GPU energy on average. For a GPU based on TBDR, the image composition represents 27% of the GPU execution time and 25% of the GPU energy on average.

Image composition is even more important for general purpose applications. As shown in Figure 8, it consumes 56% of the GPU time and 55% of its energy on average for IMR. These applications are not displayed in full screen, hence widgets from multiple applications are visible in the screen, increasing the number of surfaces that have to be merged during composition. For TBDR, 52% of GPU time and 51% of GPU energy are spent on average for image composition.

The Android GPU usage is non-negligible at all, representing an important percentage of GPU time and energy. Therefore, full-system GPU simulation is a hard requirement for achieving accurate results when evaluating a mobile GPU. Profiling just one target application provides fairly incomplete information, since multiple applications utilize the GPU concurrently and image composition makes intensive use of graphics hardware. Note that composition is still necessary even if there is just one graphical application running in full screen [21]. The GPU usage of each application can be evaluated in TEAPOT, since it captures all the OpenGL ES calls in an Android system and reports per-thread statistics.

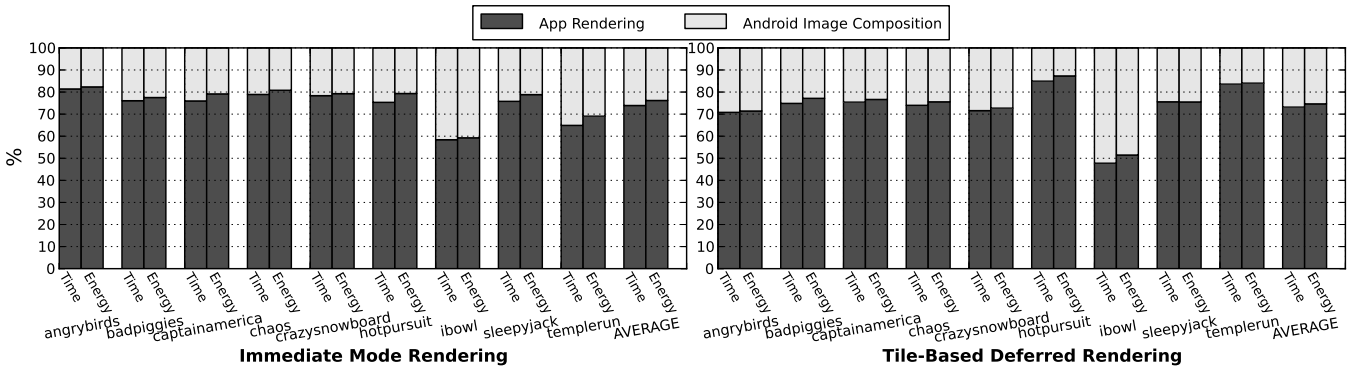



Figure 7: Percentage of GPU time and GPU energy (static and dynamic energy) for rendering and image composition, for several Android games.

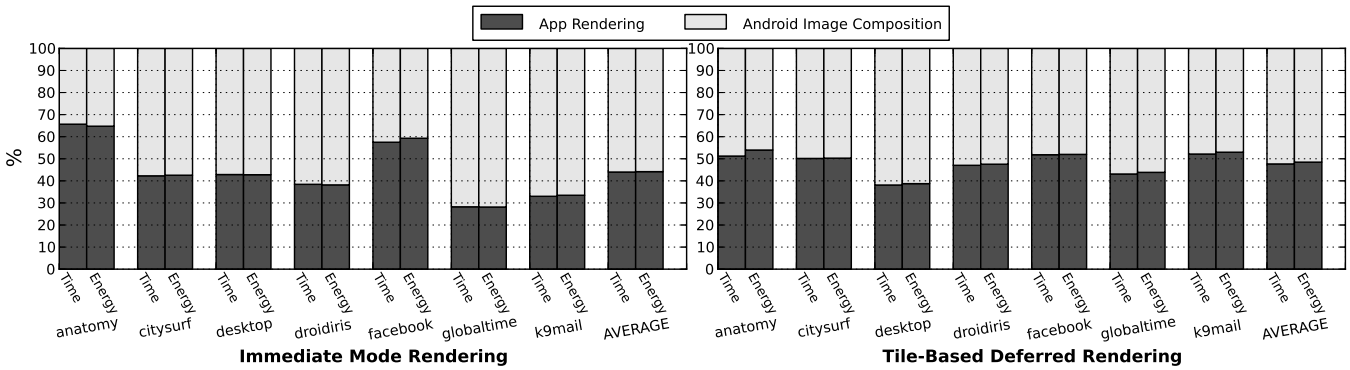



Figure 8: Percentage of GPU time and GPU energy (static and dynamic energy) for rendering and image composition, for several Android general purpose applications.

# 5. TRADING QUALITY FOR ENERGY

Current trends for mobile devices indicate a sustained increase of image quality and resolution. Image quality strongly depends on the amount of graphics processing, so it comes at the expense of higher energy consumption. Actually, the graphics subsystem consumes an important fraction of total energy on such devices [1, 3]. On the other hand, energy is a main constraint for battery-powered devices like smartphones. Hence, both image quality and operating time per battery charge are important design features, but these are unfortunately conflicting goals. Whether one or the other should prevail may depend on circumstances and even on personal preferences. For instance, when our device has recently been charged, we likely want it to operate at its best video quality. However, when battery is close to be exhausted, we may prefer to keep the phone alive and preserve most of its functionality (e.g. to avoid losing important phone calls), at least until we can plug it to recharge. In that case, we may likely be happy to sacrifice some video quality for energy. There is a large body of research on techniques for reducing GPU energy consumption, such as texture compression [15], prefetching [16], access/execute decoupling [1], etc.

As a way to illustrate the utility of TEAPOT, we have conducted a case study to evaluate the energy saving usefulness of trading image quality for energy. One simple way of trading video quality for energy is by reducing screen resolution. We have performed a series of experiments measuring both the energy consumed by the GPU (static and dynamic energy) and the quality of the produced images at different screen resolutions. TEAPOT calculates several well established image quality metrics (MSSIM [17], PSNR [32], and MSEv[25]). These metrics assess the quality of an image by computing its similarity to a given reference image. For the following experiments, the reference image frames are generated at HD resolution (1280x720), as illustrated in Figure 13, and the images under test are produced at WVGA (800x480), VGA (640x480), HVGA (420x380) and QVGA (320x240) resolutions.

Figure 9 shows the normalized GPU energy, including both static and dynamic energy, spent when running several games at these four different image resolutions. We found 55%, 64%, 81% and 90% average energy savings respectively, for a GPU based on IMR, and 54%, 63%, 78% and 88% average energy savings for Tile-Based Deferred Rendering. Figure 10 shows similar energy results for the general purpose applications, we found 60%, 70%, 83% and 92% average energy savings respectively for IMR, and 56%, 65%, 82% and 90% average energy savings for TBDR.

Figure 11 shows the corresponding image qualities, evaluated by using the MSSIM index. We found 3%, 5%, 7% and 13% of average quality loss for the Android games, respectively. The worst case image quality losses are around 20%, and they are observed by *captainamerica*, *chaos* and *hotpursuit* at QVGA resolution. However, the general purpose applications are less sensitive to the screen resolution than

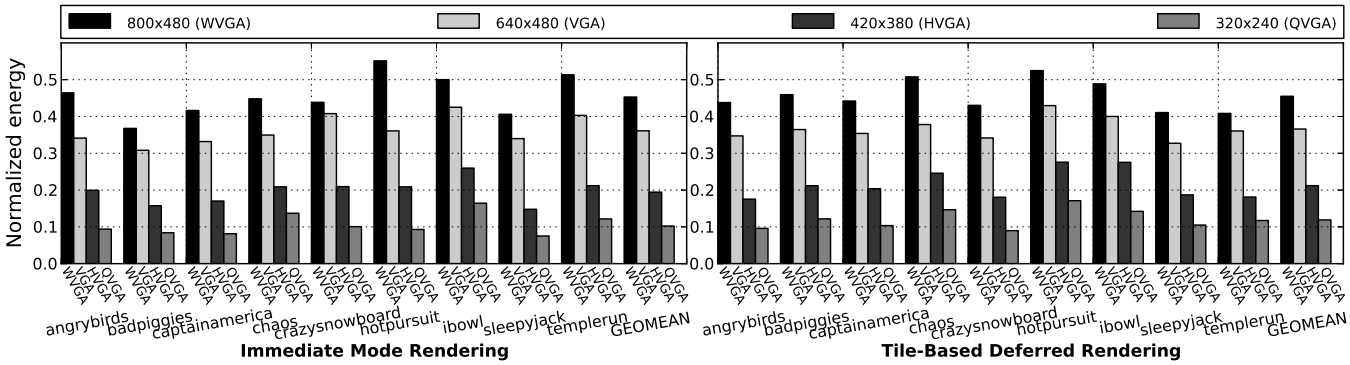



Figure 9: Normalized GPU energy (static and dynamic), in relation to the energy consumed when using HD resolution, for several Android games. The graph includes results for IMR (left) and TBDR (right).

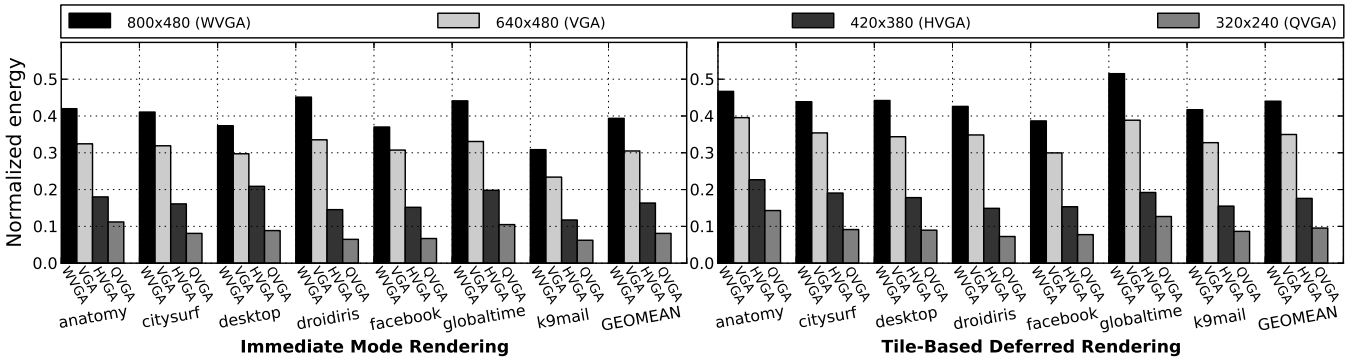



Figure 10: Normalized GPU energy (static and dynamic), in relation to the energy consumed when using HD resolution, for several general purpose applications. The graph includes the numbers for IMR (left) and TBDR (right).

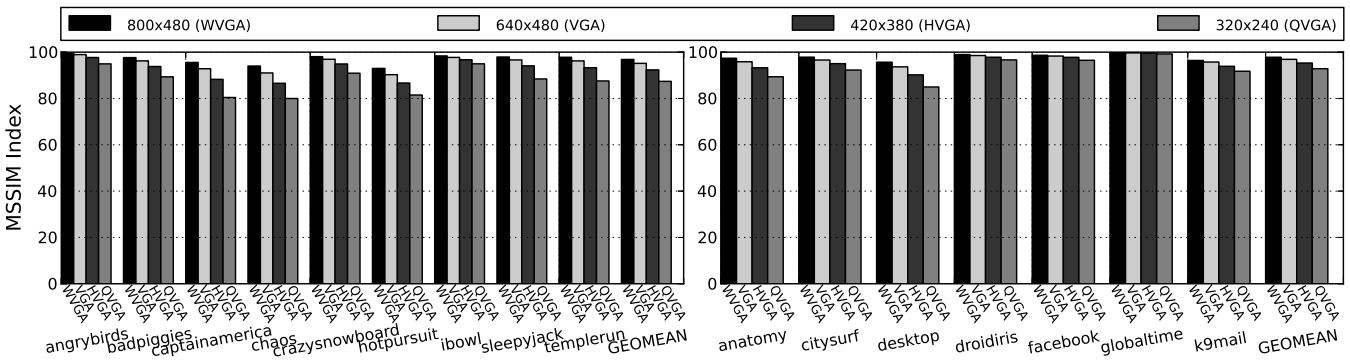



Figure 11: Image quality, compared with HD resolution, for several Android games (left) and general purpose applications (right). The MSSIM index [17] is employed as the metric for automatic image quality assessment.

games. We observed just 2%, 3%, 5% and 7% of average quality loss, respectively. Games tend to render more complex objects than the general purpose applications so they are more sensitive to screen resolution.

Figure 12 plots average quality versus energy, it includes just the numbers for IMR since very similar results were obtained for TBDR. The graph shows that reducing image quality provides at first a huge energy reduction, but further quality sacrifices tend to produce progressively lower energy savings. The main energy savings are obtained by the first reduction step (from HD to WVGA), which saves 55% the energy by just 3% loss in quality (Android games). In other words, it reduces 18,3% of energy for each 1% in quality, while further successive downgrading steps (to VGA, HVGA and QVGA) provide only additional reductions of 5.4%, 5.9%, and 1.8% of energy for each 1% in quality. Similar results are observed for general purpose applications. Moving from HD to WVGA provides 30% energy savings on average for each 1% in quality, whereas downgrading to VGA, HVGA and QVGA achieves just additional energy reductions of 9.7%, 8.9% and 3.3% for each unit of quality respectively.

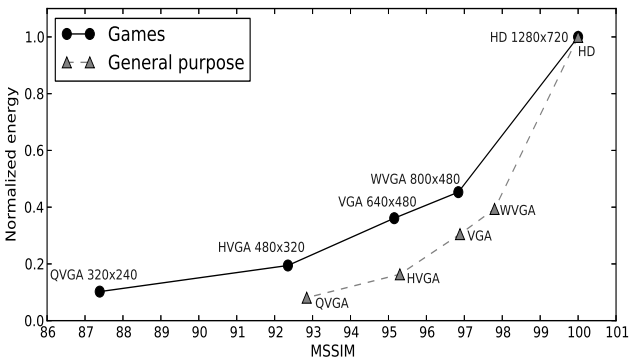
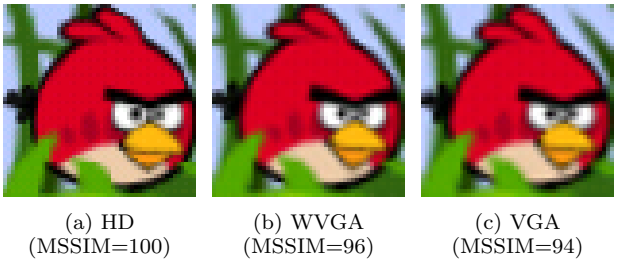



Figure 12: Image quality vs energy for different screen resolutions. The $x$ coordinate for each point is the geometric mean of the MSSIM for all the applications, the $y$ coordinate is the geometric mean of the normalized energy.



(a) HD (MSSIM=100) (b) WVGA (MSSIM=96) (c) VGA (MSSIM=94)

(d) HVGA (MSSIM=88) (e) QVGA (MSSIM=73)

Figure 13: Image quality is evaluated by comparison with a HD reference image. (a) is a 50x50 pix. sample crop of the HD frame. (b)-(e) show the same frame region at smaller screen resolutions (scaled up to highlight the differences).

Overall, this study shows that the graphics driver could save up to 90% of GPU energy by sacrificing less than 13% of quality for games, and up to 92% of GPU energy by losing less than 7% image quality for general purpose applications. Of course, this ability could be controlled by software so that it is activated only under predefined circumstances (e.g. battery level), or enabled by the user on demand.

## 6. RELATED WORK

The development of tools for evaluating the GPU has attracted the attention of the architectural community the last few years. Recent simulators, such as GPGPUSim [2] or Barra [4], model General Purpose GPU (GPGPU) architectures. These tools support CUDA [27] or OpenCL [28], but they do not support graphics APIs such as OpenGL. GPGPUSim includes a power model, GPUWattch [23], which is also based on McPAT as in TEAPOT. Both power models are similar, but GPUWattch focuses on GPGPU specific features whereas TEAPOT models more specialized graphics hardware. For instance, GPGPUSim models FP units that can be combined to execute 1 double-precision (DP) or 2 single-precision (SP) operations, but TEAPOT relies on SP units since DP is common in scientific workloads but not in games. On the contrary, TEAPOT models specialized Texture Sampling units since texture fetching instructions are frequent in graphical workloads.

ATTILA [10] provides an OpenGL framework for collecting traces of desktop games and a cycle-accurate GPU simulator. A Direct3D [26] driver is also included in the last versions. Although ATTILA provides full support for desktop games, it cannot run applications for smartphones. Furthermore, its GPU simulator models a desktop-like immediate-mode renderer, whereas TBDR [35] is very popular in the low-power segment. Finally, ATTILA does not include a power model. Qsilver [13] can also collect and simulate traces from desktop OpenGL games, and it includes a power model. GRAAL [7] also provides OpenGL support and a power model for GPUs. Furthermore, it models a low-power GPU based on TBDR. However, OpenGL ES support is not available in any of these simulators so they cannot run mobile applications for smartphones and tablets. Unlike the aforementioned tools, TEAPOT additionally provides image quality metrics for automatic image quality assessment and includes a power model for OLED screens, since it has been designed for analyzing graphical workloads in the low-power segment. In addition, TEAPOT supports full-system GPU simulation, being able to profile multiple applications accessing the GPU concurrently.

## 7. CONCLUSIONS

This paper presents TEAPOT, a toolset for evaluating mobile graphics systems. Unlike previous GPU simulators, TEAPOT is tailored towards the low-power segment. It provides full-system cycle-accurate GPU simulation for mobile workloads and a power model for both the GPU and the OLED screen. Moreover, it employs image quality metrics, allowing for image quality assessment. We have conducted two case studies in order to demonstrate some of the potential uses of TEAPOT. First, we have analyzed the importance of full-system GPU simulation, showing that the OS GPU usage is non-negligible. Our results demonstrate that Android image composition takes 26% of GPU time and 24% of GPU energy on average for a set of commercial games. We then turned our focus into analyzing techniques that trade quality for energy. We showed that significant amounts of energy can be saved by allowing for small distortions in image quality. Our results indicate that by reducing the quality of the graphics by 13%, energy savings of 90% are attainable.

## 8. ACKNOWLEDGMENTS

This work has been supported by the Generalitat de Catalunya under grant 2009SGR-1250, the Spanish Ministry of Economy and Competitiveness under grant TIN 2010-18368, and Intel Corporation. Jose-Maria Arnau is supported by an FI-Research grant.

## 9. REFERENCES

[1] J.-M. Arnau, J.-M. Parcerisa, and P. Xekalakis. Boosting Mobile GPU Performance with a Decoupled

Access/Execute Fragment Processor. In *Proc. of ISCA*, pages 84–93, 2012.

[2] A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. Aamodt. Analyzing CUDA Workloads Using a Detailed GPU Simulator. In *Proc. of ISPASS*, pages 163–174, 2009.

[3] A. Carroll and G. Heiser. An Analysis of Power Consumption in a Smartphone. In *Proc. of USENIX*, pages 21–21, 2010.

[4] S. Collange, M. Daumas, D. Defour, and D. Parello. Barra: A Parallel Functional Simulator for GPGPU. In *Proc. of MASCOTS*, pages 351–360, 2010.

[5] M. Dong, Y.-S. K. Choi, and L. Zhong. Power Modeling of Graphical User Interfaces on OLED Displays. In *Proc. of DAC*, pages 652–657, 2009.

[6] M. W. Eldridge. *Designing Graphics Architectures Around Scalability and Communication*. PhD thesis, 2001.

[7] B. Juurlink, I. Antochi, D. Crisu, S. Cotofana, and S. Vassiliadis. GRAAL: A Framework for Low-Power 3D Graphics Accelerators. *IEEE Computer Graphics and Applications*, 28(4):63–73, 2008.

[8] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In *Proc. of MICRO*, pages 469–480, 2009.

[9] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs. A Sorting Classification of Parallel Rendering. *IEEE Comput. Graph. Appl.*, 14(4):23–32, July 1994.

[10] V. Moya, C. Gonzalez, J. Roca, A. Fernandez, and R. Espasa. ATTILA: A Cycle-level Execution-Driven Simulator for Modern GPU Architectures. In *Proc. of ISPASS*, pages 231–241, 2006.

[11] K. Pulli, T. Aarnio, K. Roimela, and J. Vaarala. Designing Graphics Programming Interfaces for Mobile Devices. *IEEE Comput. Graph. Appl.*, 25(6):66–75, Nov. 2005.

[12] J. Rasmusson, J. Hasselgren, and T. Akenine-Möller. Exact and Error-Bounded Approximate Color Buffer Compression and Decompression. In *Proc. of EUROGRAPHICS Conf. on Graphics Hardware*, pages 41–48, 2007.

[13] J. W. Sheaffer, D. Luebke, and K. Skadron. A Flexible Simulation Framework for Graphics Architectures. In *Proc. of the EUROGRAPHICS Conf. on Graphics Hardware*, pages 85–94, 2004.

[14] A. Shye, B. Scholbrock, and G. Memik. Into the Wild: Studying Real User Activity Patterns to Guide Power Optimizations for Mobile Architectures. In *Proc. of MICRO*, pages 168–178, 2009.

[15] J. Ström and T. Akenine-Möller. iPACKMAN: High-Quality, Low-Complexity Texture Compression for Mobile Phones. In *Proc. of the EUROGRAPHICS Conf. on Graphics Hardware*, pages 63–70, 2005.

[16] J. Tang, S. Liu, Z. Gu, C. Liu, and J.-L. Gaudiot. Prefetching in Embedded Mobile Systems Can Be Energy-Efficient. *IEEE Comput. Archit. Lett.*, 10(1):8–11, Jan. 2011.

[17] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli. Image Quality Assessment: from Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[18] Android SDK. `http://developer.android.com/sdk/index.html`.

[19] Bringing High-End Graphics to Handheld Devices. `http://www.nvidia.com/content/PDF/tegra_white_papers/Bringing_High-End_Graphics_to_Handheld_Devices.pdf`.

[20] Color Space Conversions. `http://www.poynton.com/PDFs/coloureq.pdf`.

[21] Composition with Snapdragon. `https://developer.qualcomm.com/sites/default/files/composition-with-snapdragon.pdf`.

[22] Gallium3D. `http://en.wikipedia.org/wiki/Gallium3D/`.

[23] GPUWattch. `http://www.gpgpu-sim.org/gpuwattch/`.

[24] Mali-400 MP: A Scalable GPU for Mobile Devices. `http://www.highperformancegraphics.org/previous/www_2010/media/Hot3D/HPG2010_Hot3D_ARM.pdf`.

[25] Mean Squared Error. `http://en.wikipedia.org/wiki/Mean_squared_error`.

[26] Microsoft Direct3D. `http://en.wikipedia.org/wiki/Microsoft_Direct3D`.

[27] NVIDIA CUDA Programming Guide. `http://docs.nvidia.com/cuda/cuda-c-programming-guide/`.

[28] OpenCL. `http://www.khronos.org/opencl/`.

[29] OpenGL ES. `http://www.khronos.org/opengles/`.

[30] OpenGL Shading Language. `http://en.wikipedia.org/wiki/GLSL`.

[31] Qualcomm Adreno 320. `http://www.anandtech.com/show/6112/qualcomms-quadcore-snapdragon-s4-apq8064adreno-320-performance-preview`.

[32] Peak signal-to-noise ratio. `http://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio`.

[33] PowerVR Technology Overview. `http://www.imgtec.com/factsheets/SDK/PowerVR%20Technology%20Overview.1.0.2e.External.pdf`.

[34] QEMU. `http://wiki.qemu.org/Main_Page`.

[35] Tile-Based Deferred Rendering. `http://en.wikipedia.org/wiki/Tiled_rendering`.

[36] Tungsten Graphics Shader Infrastructure. `http://people.freedesktop.org/~csimpson/gallium-docs/tgsi.html`.

[37] Using hardware acceleration in the Android Emulator. `http://developer.android.com/tools/devices/emulator.html#acceleration`.