Power Analysis of Mobile 3D Graphics

Bren Mochocki
Dept of CSE, University of Notre Dame
Notre Dame, IN
bmochock@cse.nd.edu

Kanishka Lahiri
NEC Laboratories America
Princeton, NJ
klahiri@nec-labs.com

Srihari Cadambi
NEC Laboratories America
Princeton, NJ
cadambi@nec-labs.com

***Abstract*— The world of 3D graphics, until recently restricted to high-end workstations and game consoles, is rapidly expanding into the domain of mobile platforms such as cellular phones and PDAs. Even as the mobile chip market is poised to exceed production of 500 million chips per year, incorporation of 3D graphics in handhelds poses several serious challenges to the hardware designer. Compared with other platforms, graphics on handhelds have to contend with limited energy supplies and lower computing horsepower. Nevertheless, images must still be rendered at high quality since handheld screens are typically held closer to the observer's eye, making imperfections and approximations very noticeable.**

In this paper, we provide an in-depth quantitative analysis of the power consumption of mobile 3D graphics pipelines. We analyze the effects of various 3D graphics factors such as resolution, frame rate, level of detail, lighting and texture maps on power consumption. We demonstrate that significant imbalance exists across the workloads of different graphics pipeline stages. In addition, we illustrate how this imbalance may vary dynamically, depending on the characteristics of the graphics application. Based on this observation, we identify and compare the benefits of candidate Dynamic Voltage and Frequency Scaling (DVFS) schemes for mobile 3D graphics pipelines. In our experiments we observe that DVFS for mobile 3D graphics reduces energy by as much as 50%.

I. Introduction

Traditionally, 3D graphics applications have been developed for either desktop computers or dedicated gaming consoles. However, with the increasing popularity and capabilities of mobile computing devices such as PDAs and cellular phones, many 3D graphics applications such as gaming, GPS-backed maps and animated chats emerge as possible applications for current and future mobile platforms. Since the mobile market far exceeds the PC market, a very large volume opportunity exists for 3D graphics. The mobile gaming industry already reports revenue in excess of $2.6 billion worldwide, and is expected to exceed $11 billion by the year 2010 [1]. In [2], the author predicts that all but the least expensive cellular phones will feature 3D processing capabilities by 2006, creating a sales opportunity of over 500 million chips per year.

Similar to the evolution of console gaming, players of mobile games will expect high-quality 3D experience. Yet, several major differences between mobile graphics processors and traditional gaming consoles make this a challenge. First, handheld devices have slower processors that are less capable of handling large compute-intensive workloads. Second, handheld batteries have limited lifetimes, necessitating low-power schemes for compute-intensive applications such as 3D graphics. Finally, although handheld screens typically have lower resolutions than desktops or laptops (leading to lower energy consumption), each pixel must still be rendered accurately since the screen is held close to the observer's eye; making imperfections and approximations easily noticeable. For example, effects such as anti-aliasing and imperfect shading can result in jagged edges and reduced image quality.

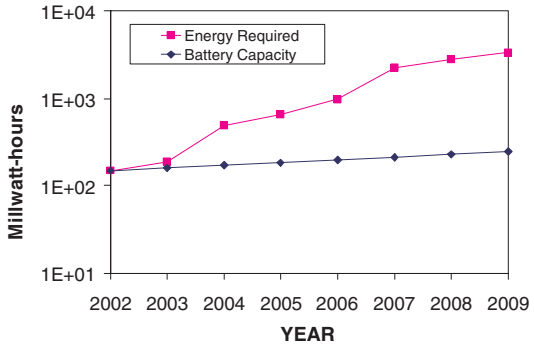


Fig. 1. Disparity between mobile 3D graphics energy and battery capacity.

In this paper, we present a quantitative study of the different factors that influence energy consumption of a handheld device while running a 3D graphics application. From our our analysis, we observe that a significant opportunity exists for dynamic voltage and frequency scaling (DVFS) in mobile 3D graphics processing. We then experimentally investigate the applicability of different DVFS schemes for energy-efficient 3D graphics processing in handheld devices.

To understand the need for power-efficient 3D graphics, consider Figure 1, which illustrates an increasing disparity between the energy requirements of 3D graphics processing on handhelds, and energy available from state-of-the-art batteries. It is estimated that battery capacities (assuming the weight of the battery is constant) will increase at rates of 5 to 10% per year [3]. Concurrently, however, screen resolution and frame rates will also increase. Current resolutions for handhelds are 176 x 144 (QCIF), 320 x 280 (QVGA), 320 x 320 (Palm Treo [4]) and 480 x 320 (Sony Clie [5]). [6] reports that a 4-inch LCD screen with a resolution of 800x600 pixels (SVGA) is being developed and is expected to be available in the next few years. Based on this, it is reasonable to expect that SVGA screens will be prevalent in handhelds by the end of the decade. For Figure 1, we assumed (i) screen resolutions increase uniformly from QCIF in 2002 to SVGA in 2009, (ii) average frame rates increase from 10 frames per second (fps) in 2002 to 20 fps in 2009 and (iii) voltage and clock frequencies vary as speculated by the ITRS [7]. It is clear that improvements in battery technology alone cannot be expected to satisfy the energy requirements of 3D graphics processing in future handheld devices.

Due to the above trends, the area of low-power mobile 3D graphics processing has recently started to receive interest. Application-level techniques for improving energy efficiency of graphics processing while trading off the quality of the

3-9810801-0-6/DATE06 © 2006 EDAA

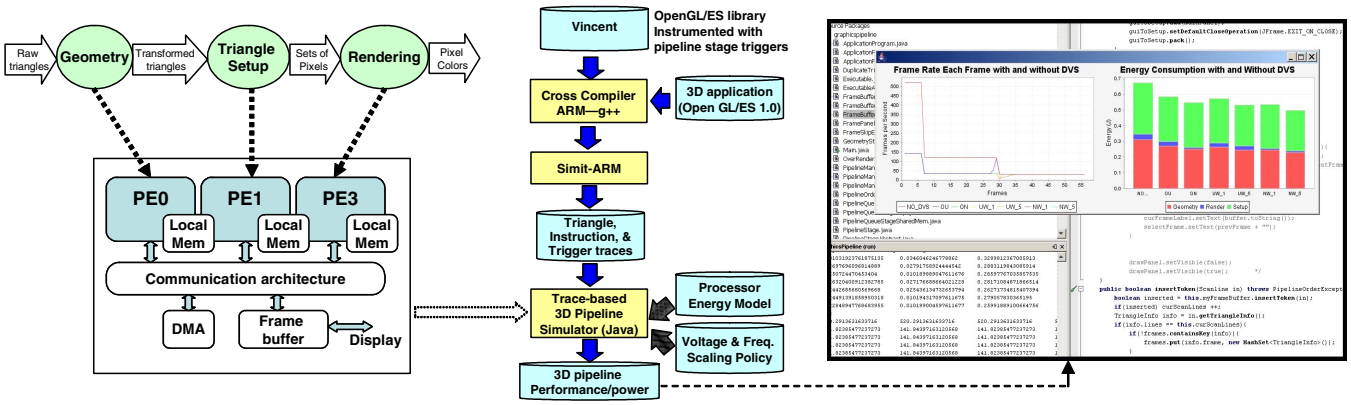


Fig. 2. Framework for analyzing the 3D graphics pipeline: (a) pipeline stages and mapping to an example system architecture; (b) methodology for performance/power analysis; (c) screenshot of front-end of trace-based simulator.

rendered frames include novel texturing and Level-of-Detail (LoD) management strategies [8], [9]. A related area is the prediction and analysis of 3D workload requirements [10], [11]. However, most work in this area is targeted towards high-end gaming platforms such as PCs and gaming consoles. On the hardware side, several low-power graphics processors (GPUs) have started to appear featuring conventional power management schemes. The GPU described in [12] uses clock gating to deactivate unused components, and optionally can activate a "step mode", which prevents multiple stages of the graphics pipeline from executing at the same time. A GPU that utilizes Dynamic Frequency Scaling (DFS) is presented in [13], which includes three performance states, but does not consider voltage scaling. Recently, performance and power modeling techniques for GPUs have been proposed [14]. Similar to [14], the analysis framework we use for our work is also based on trace-driven simulation. More widespread availability of such modeling and analysis frameworks will spur further research on power-efficient graphics processing.

Numerous techniques for Dynamic Voltage and Frequency Scaling (DVFS) have been proposed over the last decade [15], leading to commercial products today that feature DVFS, including processors for mobile handsets [16]. It is known that general-purpose DVFS policies (such as those based on utilization measurements) often do not perform well for specific applications [17], resulting in more specialized techniques that are tailored to important application domains [18], [19]. No prior work that we are aware of has investigated the application of DVFS for power-efficient 3D graphics processing.

The rest of this paper is organized as follows. In Section II, we briefly review 3D graphics pipelines. In Section III, we describe the methodology we used to analyze the performance and energy consumption of the 3D graphics pipeline, along with motivational studies. In Section IV, we present detail experimental studies that analyze the effects of different factors on pipeline energy consumption. Based on these observations, in Section V, we identify and compare candidate DVFS schemes for 3D pipelines, and conclude in Section VI.

II. 3D Graphics Pipeline

In typical graphics processors all surfaces are fundamentally represented with triangular patches [20]. During rendering, each triangle is drawn in a series of three basic steps, which can be implemented as three stages in a pipeline architecture (Figure 2(a)).

1) The *geometry* stage applies geometric transforms to each triangle, and computes its perspective projection onto the screen. It also culls triangles that will not be seen and computes shading information for each vertex.
2) The *triangle setup* stage determines which pixels lie within the projection of each triangle. Along each triangle edge, it uses interpolation and lookup to obtain shading, texture and z-values for each pixel.
3) The *rendering* stage computes the actual color for each pixel, and also performs hidden-surface elimination and the final interpolation of shading and texture values.

Several factors affect the perceived quality of 3D graphics. Key quality factors discussed in this paper include:

- *Resolution* – either the total number of pixels on the screen, or the number of pixels per inch of screen. Note that handheld devices are typically held close to the eye, and consequently demand a higher number of pixels per inch than desktop monitors.
- *Frame rate* – the rate with which the scene is redrawn. To create an illusion of smooth motion, this should be at least 10 to 12 frames per second. Below this rate, motion appears jerky, and games feel unresponsive. It is important to note however, that the frame rate can vary considerably during the course of an animation without spoiling the illusion of motion.
- *Level of detail* – the sampling used to represent curved shapes with triangular patches. This determines how smooth the shapes' silhouettes appear, and if the individual triangles are visible on the surface due to shading.
- *Lighting model* – the type of lighting applied to the scene. Possible lighting models include spot lighting (illumination only in specified areas of the scene), point lighting (illumination from nearby light sources) and parallel lighting (illumination from distant light sources).
- *Texture model* – how textures, usually in the form of 2D images, are applied to surfaces. There are several different methods of applying textures to surfaces, with different tradeoffs between computation and visual quality.

III. Evaluation Framework

Motivated by the emergence of commercial multi-core application processors in the mobile handset domain [21], we consider a system architecture consisting of three embedded processors, each of which executes a specific stage of the 3D graphics pipeline (Figure 2(a)). The different pipeline stages communicate with each other through shared buffers mapped to local memories. DMA engines transfer data between stages.

The methodology we used for analyzing the 3D graphics pipeline is illustrated in Figure 2(b). We adapted a software library [22] that implements OpenGL/ES [23], a standard interface for developing 3D graphics applications on resource-constrained systems. OpenGL/ES is a variant of the popular OpenGL standard (aimed at high-performance devices). The applications we used consisted of ones obtained from [24] as well as custom benchmarks. We selected benchmarks that clearly exhibited specific 3D effects (*e.g.,* types of lighting) as opposed to complex animation sequences, in order to isolate and better understand the impact of individual factors. The library was modified to generate a trace of triggers indicating start and completion points of each pipeline stage for each triangle being processed. The applications and the modified library were cross-compiled and linked to the target architecture [25]. To drive the subsequent analysis, traces of executed instructions, triggers, and triangles were collected using fast, cycle-accurate, instruction-level simulation [26]. The resulting traces were provided to a trace-based analysis tool that was developed to analyze performance and energy consumption of the different pipeline stages under a given system architecture. The simulator uses an instruction-level energy model that was developed using measurement of actual current drawn by a commercial processor [27]. The processor can be operated at 11 different voltage and frequency levels, with an associated overhead of 150μs each time the operating point is changed. Different voltage/frequency scaling policies and their impact on the 3D pipeline are easily evaluated using this framework. Note that, in our work, we use performance and energy models developed for a general-purpose embedded processor (ARM). However, the framework is flexible, and could be enhanced to incorporate models developed for custom GPUs as well [1]. JFreechart [28] is used to graphically display the analysis results (Figure 2(c)).

Imbalance in the Graphics Pipeline

We used the above framework to compare the workload across the pipeline stages for three different benchmarks: *TexCube*, which features the rotation of a textured cube on one of its diagonals, *RoomRev*, which depicts a revolving room containing curved geometric objects, and *MovSphere*, which displays a lit sphere moving toward the camera with increasing LoD. Figure 3 reports the results of these studies, for animation sequences aimed at a screen resolution of 176x220. From the figure we observe that the workload in each benchmark *varies significantly across stages*. In the case of the *TexCube* [24] benchmark, rendering consumes 11x more cycles than geometry and setup combined. However, the results also show that contrary to expectation, rendering is not always the bottleneck. For the *RoomRev* [24] benchmark, geometry has the maximum workload, exceeding rendering by 5X, while in the *MovSphere* benchmark, setup dominates rendering by 9X. Clearly, large, application-dependent imbalances may exist between different pipeline stages. In general, we observe that the imbalance may shift dynamically, depending on the animation sequence, leading to variable "hot spots". In the next section, we study the factors on which such imbalance depends in more detail.

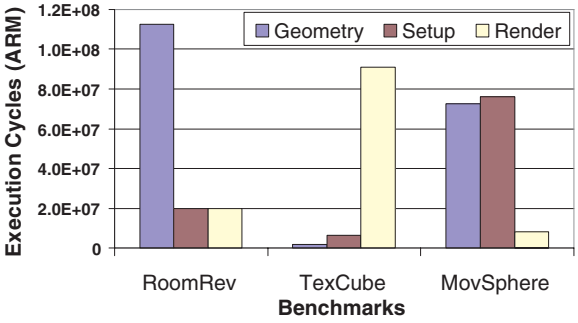


Fig. 3. Analysis of the workload of different stages of the 3D graphics pipeline for different applications.

IV. Performance and Energy Analysis

Resolution, level of detail (LoD), lighting, texture maps and frame rate are factors that play a vital role in determining the quality of 3D graphics. We analyze in detail how these factors affect energy in handhelds, and how the graphics hardware developer can make design choices to lower energy consumption but maintain the quality of the 3D images. Significantly, the inherent *pipeline imbalance* exposed by our analysis translates to an opportunity for the hardware designer to employ dynamic voltage and frequency scaling. Our analysis also provides information for the application developer to write energy-efficient mobile 3D graphics applications.

A. Effect on Performance

We start by analyzing how the above factors affect handheld graphics processing requirements, and use this information to devise energy-saving strategies. Our measurements are represented by execution cycles obtained from the framework described in Section III.

Resolution: Figure 4 shows the performance with increasing resolution for the MovSphere benchmark with a LoD of 3. In our setup, an LoD of 3 means that 256 (4^{3+1}) triangles are used to represent the surface of the sphere. The resolutions chosen are currently used in handhelds and gaming consoles (the latter will likely be used in future handhelds). The three different curves show the number of execution cycles for each pipeline stage. The execution time for rendering increases linearly with the number of pixels. Thus, at high resolutions, the predominant performance bottleneck and power consumption unit is the rendering engine.

Level of Detail (LoD): Figure 5 shows the performance with increasing LoD for the MovSphere benchmark. A LoD of d implies that 4^{d+1} triangles are used to represent the sphere. We note that the geometry and triangle setup execution times increase exponentially with LoD (i.e., linearly with the number

[1] Our experimental results in absolute terms, are specific to ARM. However, we believe that the observed trends and conclusions drawn from them are sufficiently general, and are applicable to pipelines consisting of custom processing elements as well.

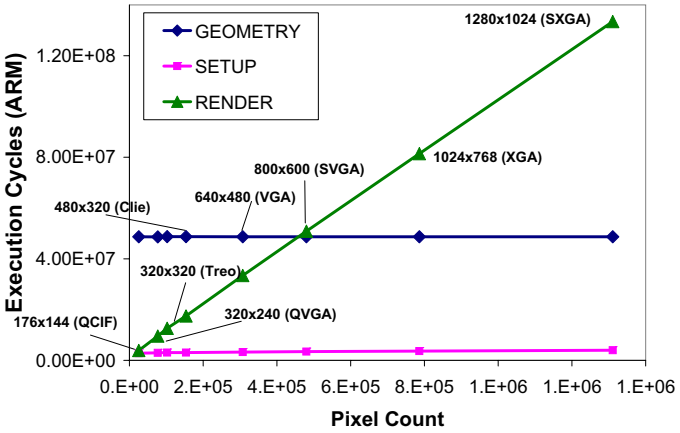


Fig. 4. Mobile 3D graphics performance with increasing resolutions.

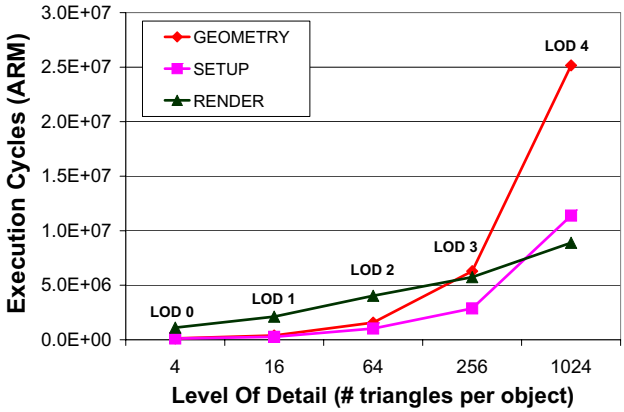


Fig. 5. Mobile 3D graphics performance with increasing Level of Detail.

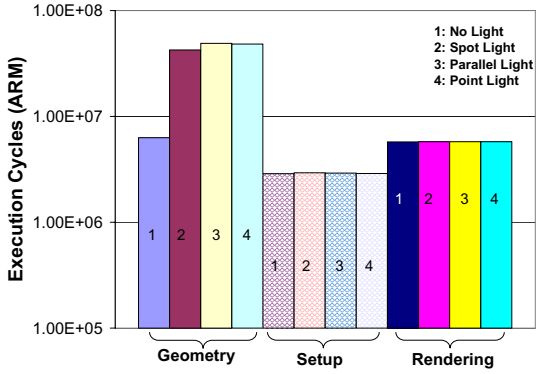


Fig. 6. Mobile 3D graphics performance for different lighting schemes.

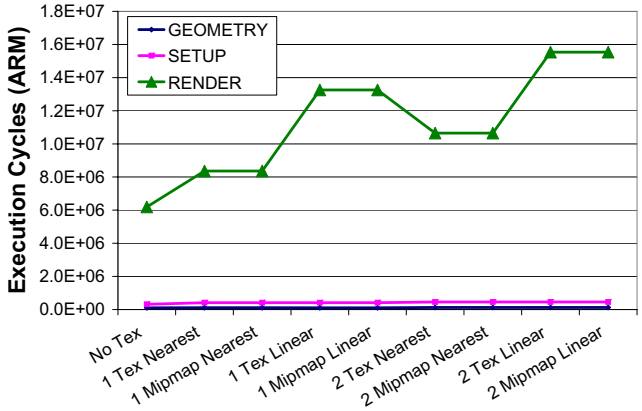


Fig. 7. Mobile 3D graphics performance for different texturing schemes.

of triangles). This is because the number of vertices on which the geometry stage operates is linearly proportional to the number of triangles. On the other hand, the rendering stage execution time increases more slowly. Thus, the overwhelming effect on performance with increasing LoD is from the geometry and triangle setup stage. In other words, if the LoD can be reduced without affecting picture quality, performance can be significantly enhanced.

Lighting: Figure 6 shows the performance for three different lighting schemes, namely, spotlight, parallel light and point light, all with an LoD of 3. The execution times for the different lighting schemes are roughly equivalent, but activating any of the lighting schemes comes with a large cost to the geometry stage, since lighting is done on a per-vertex basis. We also observed that a spotlight requires a larger LoD to produce better picture quality; this is because a spotlight has a distinct boundary, which when depicted with few triangles shows up in a "jagged" manner resulting in poor image quality. Therefore, spotlights necessitate the use of a larger LoD, while point and parallel lights do not. Since increasing the LoD results in an exponential increase in execution time, it is clear that graphics application programmers should minimize the use of spotlights to enhance handheld graphics performance.

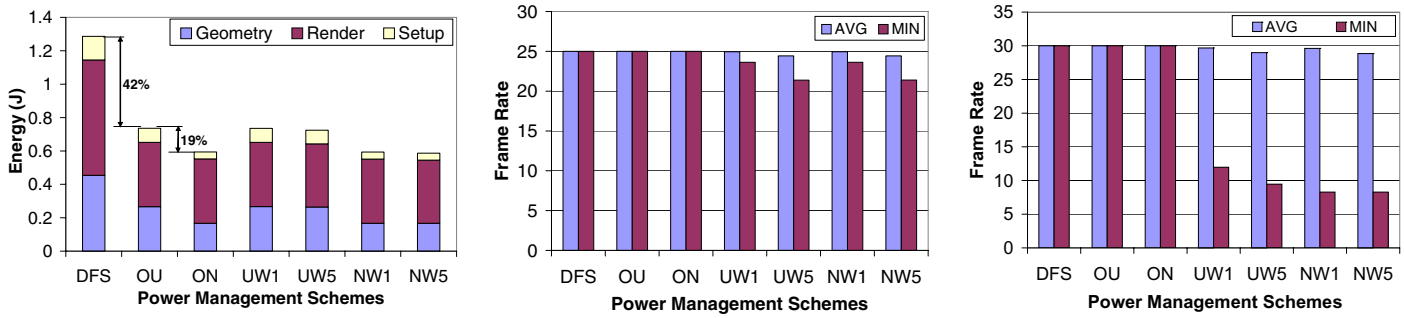
Texture Maps: Figure 7 shows the performance for different texturing schemes. The first set of 4 points use a single texture map, while the second set of 4 points use 2 texture maps. We evaluate the following four texturing schemes, popularly used in OpenGL/ES. Textures are represented by texture maps, consisting of texels (pixels in the texture map).

- **Nearest:** In this scheme, the renderer picks a single representative texel for each pixel that has to be rendered.
- **MipMap Nearest:** Mipmapping involves making multiple copies of the original texture map, with each copy having half the resolution of the previous one. Depending on the distance of the object from the observer, a suitable texture map is selected. For example, if the object is far away, a low-resolution texture map is chosen. A representative texel from the chosen texture map is picked (as in the "Nearest" scheme).
- **Linear:** Here a weighted average of four texels closest to the pixel is chosen for texturing. The weights are based on the Manhattan distance between the texel and pixel.
- **MipMap Linear:** This is a combination of the mipmap and linear schemes.

From the figures, we note that while mipmapping improves image quality, it has little effect on execution time. However, bilinear texturing schemes increase execution time significantly. Hence, we believe it is better for handhelds to employ mipmapping texturing schemes.

B. Influence on Energy

We now discuss how the above analysis can help design more energy-efficient mobile graphics processors. In particular, we find that a direct result of tuning the resolution, lighting, level of detail and texturing is a change in the balance between the three main stages of the graphics pipeline. In other



(a) (b) (c)

Fig. 8. Comparing energy savings and frame rates for the different power management schemes

words, the performance bottleneck shifts from one stage to the other depending on the factors being changed. We observe the following from our analysis:

- **Resolution:** The execution time of the rendering stage increases the fastest with resolution, making it the performance bottleneck at high resolutions. This creates an imbalance between the rendering stage and other stages.
- **LoD:** Since a larger LoD implies more triangles per object, we see that with increasing LoD, the geometry stages becomes the performance bottleneck. However, it must be noted that in most applications, as resolution increases, so does the LoD. This means that the geometry and rendering bottlenecks (caused by increasing LoD and increasing resolutions respectively) might offset each other during the course of the application. Thus, the imbalance could be "dynamic" and application-dependent.
- **Lighting:** The cost for activating lighting is very large in the geometry stage but has little impact on other stages. However, the change in workload across the different lighting schemes is minimal. Since most applications use lighting to emphasize 3D features of objects in the scene, it is usually difficult to avoid this cost. However, it may be possible in some cases to replace the lighting calculation with "pre-lit" texture maps, thus shifting the workload for some objects from the geometry to the render stage.
- **Texture Maps:** The commonly used linear and mipmapping texturing schemes make the rendering stage the bottleneck, with the linear scheme proving to be more computationally expensive.

These variable performance bottlenecks provide an opportunity for power reduction via dynamic voltage and frequency scaling (DVFS), whose application we consider next.

V. Power Saving Techniques

To effectively exploit the imbalance between different stages of the graphics pipeline via DVFS, accurate prediction of future pipeline workload is necessary. In this section, we compare a few different workload predictors and the savings they achieve in terms of pipeline energy consumption.

A. Workload Prediction Schemes

In the presence of imbalance, a basic technique that can be used to reduce energy is dynamic frequency scaling (DFS), which is used in [13], where the frequencies of all stages are reduced as much as possible so that the target frame rate is met. In our work, we use dynamic voltage *and* frequency scaling for mobile 3D graphics, with two further innovations. First, we use non-uniform DVFS where each stage in the graphics pipeline uses a different voltage and frequency. Second, we use history-based workload predictors for 3D graphics to set voltage and frequency. In all the schemes we consider, the voltage and frequency can be changed at most once per frame.

1) *Oracle Uniform (OU):* "Oracle" implies a perfect workload predictor. OU refers to a DVFS technique where voltage and frequency of the geometry, setup and rendering stages are all set to the same value, based on perfect knowledge of future workloads. The voltage and frequency is set to the maximum of the three stages.
2) *Oracle Non-uniform (ON):* In this technique, the voltage and frequency of the each pipeline stage is set independently, again based on perfect knowledge of future workloads. Thus, the geometry, setup and rendering engines could have different frequencies and voltages.
3) *Uniform Window-based Predictor 1 (UW1):* The oracle schemes can tell us how well voltage scaling works and how useful non-uniform voltage scaling is. However, history-based prediction schemes are more practical than the oracle schemes. In the UW1 scheme, the workload of each pipeline stage in the current frame is predicted to be the same as the corresponding workloads in the previous frame. Using this prediction, the voltage and frequency of the each stage are set to the same value.
4) *Uniform Window-based Predictor 5 (UW5):* This is the same as the UW1 scheme, except the workload of the current frame is determined using the average of the workloads of 5 previous frames.
5) *Non-uniform Window-based Predictor 1 (NW1):* In this scheme, the workloads for the geometry, setup and rendering stages are predicted to be the same as the corresponding workloads in the previous frame. Then, the frequency and voltage of each stage is set independently.
6) *Non-uniform Window-based Predictor 5 (NW5):* This is the same as NW1 except the workload prediction is based on the average of the previous 5 frames.

B. Results for DVFS-based Energy Reduction Schemes

Figure 8(a) shows energy estimates for the above DVFS schemes for the *multicube* example with a target frame rate of 25 fps. The *multicube* example is an extension of the *texcube* example consisting of 3 cubes revolving around a central frustum; the cubes have different texture maps and lighting to resemble real applications. Workloads predictions are made in terms of ARM execution cycles. Comparing "Oracle Uniform"

against baseline DFS shows that voltage scaling alone reduces energy by 41%. Our technique of non-uniform voltage scaling ("Oracle Non-uniform") further reduces energy by 19%. The history-based predictors perform almost as well as the oracle predictors, with the non-uniform predictor (NW5) achieving a 19% energy reduction over the uniform predictor (UW5). Overall, with non-uniform voltage scaling with history-based workload prediction, we see an energy reduction of 54% compared to DFS (used in [13]).

Nevertheless, like all predictors, the estimates of history-based predictors can be inaccurate. For instance, if the actual workload turns out to be larger than the predicted workload, the resultant frame rate will not meet the target since voltage and frequency will have been reduced according to the prediction. Figure 8(b) shows this data for the *multicube* example. Two important points emerge from the figure. First, the average frame rate for history-based predictors is close to the target of 25 fps. Second, the minimum frame rate is around 22 fps. Our measurements indicate that the minimum frame rate occurs less than 9% of the time. Despite this, the difference between 22 and 25 fps was imperceptible to the human eye, which was verified using on-screen emulation. Consequently, for this benchmark, the history-based scheme does not affect animation quality.

We also note that the overall energy improvement is almost equal to the sum of the reductions from voltage scaling and non-uniform voltage scaling. Therefore the effect on energy due to the missed target frame rates is negligible. In other words, the fact that the history-based scheme occasionally misses the frame rate due to mis-predictions does not reduce energy consumption. Finally, it is worth noting that the voltage transition overhead of 150μs is small enough not to affect the frame rates, since voltage changes are applied once per frame.

Figure 8(c) shows frame rates for the *MovSphere* example. As the sphere draws closer to the observer, the level of detail (LoD) increases, resulting in increased workload. The history-based predictors have a downward spike in the frame rate between the frames where the LoD switch occurs. When an LoD switch occurs, the speed needs to be increased, but the history-based predictors do not detect that until it is too late. Therefore, the frame rate drops significantly, in this case to below 10 fps, which is noticeable. In this case, history-based predictors with a window of 1 can correct themselves faster than predictors with a window of 5. Nonetheless, such history-based predictors are not suitable when workloads change quickly by large amounts.

VI. Conclusions

In this paper we presented a detailed quantitative analysis of the workload variations and imbalances of different stages of a mobile 3D graphics pipeline, and the potential for DVFS based power savings that exploit such variations and imbalances. Our studies show that history-based DVFS strategies achieve success for examples with slowly changing graphics workloads, with savings of over 50%. Our studies also motivate more specialized prediction techniques. We believe that this paper will prove to be a valuable starting point for future work in low-power mobile 3D graphics.

Acknowledgments: The authors are grateful to Matthew M. Miller of NEC Laboratories America for useful discussions, and to the anonymous reviewers for their feedback.

References

[1] "Mobile Games Industry Worth US $11.2 Billion by 2010." http://www.3g.co.uk/PR/May2005/1459.htm, May 2005.

[2] P. Glaskowsky, "3D Poised for Market Expansion." http://www.mdronline.com/publications/mpw/issues/mpw111.html#item1, Aug 2003.

[3] I. Buchmann, *Batteries in a Portable World*. Cadex Electronics, Inc., 2001.

[4] Palm Corporation, "Treo 650 smartphone details." http://www.palm.com/us/products/smartphones/treo650/details.epl, 2004.

[5] Sony Electronics, "Peg-th55, the ultimate handheld for power users." http://sonyelectronics.sonystyle.com/micros/clie/models/th55.html, 2005.

[6] The Register UK, "Fujitsu creates 800x600 pda lcd." http://www.theregister.co.uk/2003/07/15/fujitsu_creates_800x600_pda_lcd%/, 2003.

[7] "International Technology Roadmap for Semiconductors (ITRS 2004)." http://www.itrs.net/Common/2004Update/2004_000_ORTC.pdf, 2004.

[8] N. Tack, F. Moran, G. Lafruit, and R. Lauwereins, "3D Graphics Rendering Time Modeling and Control for Mobile Terminals," in *Proceedings of Int. Conf. on 3D Web technology*, pp. 109–117, 2004.

[9] T. Akenine-Moller and J. Strom, "Graphics for the Masses: A Hardware Rasterization Architecture for Mobile Phones," in *Proceedings of ACM SIGGRAPH*, pp. 801–808, 2003.

[10] T. Mitra and T. Z. Chiueh, "Dynamic 3D Graphics Workload Characterization and the Architectural Implications," in *Proc. Intl. Symp. on Microarchitecture (MICRO-32)*, pp. 62–71, Nov. 1999.

[11] M. Wimmer and P. Wonka, "Rendering Time Estimation for Real-Time Rendering," in *Eurographics Symposium on Rendering 2003*, pp. 118–128, 2003.

[12] M. Kameyama, Y. Kato, H. Fujimoto, H. Negishi, Y. Kodama, Y. Inoue, and H. Kawai, "3D Graphics LSI Core for Mobile Phone Z3D," in *Graphics Hardware*, pp. 60–66, 2003.

[13] R. Woo, S. Choi, J.-H. Sohn, S.-J. Song, and H.-J. Yoo, "A 210-mW Graphics LSI Implementation Full 3-D Pipeline With 264 Mtexels/s Texturing for Mobile Multimedia Applications," *IEEE Journal of Solid-State Circuits*, vol. 39, pp. 358–367, February 2004.

[14] J. W. Sheaffer and D. P. Luebke and K. Skadron, "A flexible simulation framework for graphics architectures," in *Proc. of the Eurographics Conference on Graphics Hardware*, pp. 85–94, Aug. 2004.

[15] T. Pering, T. Burd, and R. Brodersen, "The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms," in *Proc. Int. Symp. Low Power Electronics & Design*, pp. 76–81, Aug. 1998.

[16] "Intel Application Processors [Online]." http://developer.intel.com/design/pca/applicationprocessors.

[17] D. Grunwald, P. Levis, K. Farkas, C. B. Morrey, and M. Neufeld, "Policies for Dynamic Clock Scheduling," in *Proc. Symp. Operating Systems Design and Implementation*, pp. 73–86, Oct. 2000.

[18] S. Mohapatra et al, "Integrated Power Management for Video Streaming to Mobile Handheld Devices," in *ACM Int. Conf. on Multimedia*, pp. 582–591, Nov. 2003.

[19] V. Raghunathan, C. L. Pereira, M. B. Srivastava, and R. K. Gupta, "Energy Aware Wireless Systems with Adaptive Power-Fidelity Tradeoffs," *IEEE Trans. VLSI Systems*, vol. 13, pp. 211–225, Feb. 2005.

[20] A. H. Watt, *3D Computer Graphics*. Addison-Wesley, 2000.

[21] S. Torii et al, "A 600MIPS 120-mW 70-μA Leakage Triple-CPU Mobile Application Processor Chip," in *ISSCC*, pp. 136–138, 2005.

[22] Hans-Martin Will, "A 3-D Rendering Library for Mobile Devices." http://ogl-es.sourceforge.net/, 2004.

[23] Khronos Group, "OpenGL ES Overview." http://www.khronos.org/opengles, 2005.

[24] J. R. Villar, "Typhoon Labs-OpenGL ES Tutorials." http://www.khronos.org/devu/opengles_challenge/, 2004.

[25] B. Gatliff http://www.billgatliff.com, 2005.

[26] W. Qin, "Simit-ARM: Very fast functional and cycle-accurate simulators for ARM." http://http://sourceforge.net/projects/simit-arm/, 2004.

[27] A. Sinha and A. Chandrakasan, "JouleTrack: A Web Based Tool for Software Energy Profiling," in *Proceedings of the 38th Design Automation Conference*, pp. 220–225, 2001.

[28] D. Gilbert and T. Morgner, "JFreeChart." http://www.jfree.org/jfreechart/, 2005.