Measuring Per-Frame Energy Consumption of Real-Time Graphics Applications

Björn Johnsson Tomas Akenine-Möller
Lund University and Intel Corporation

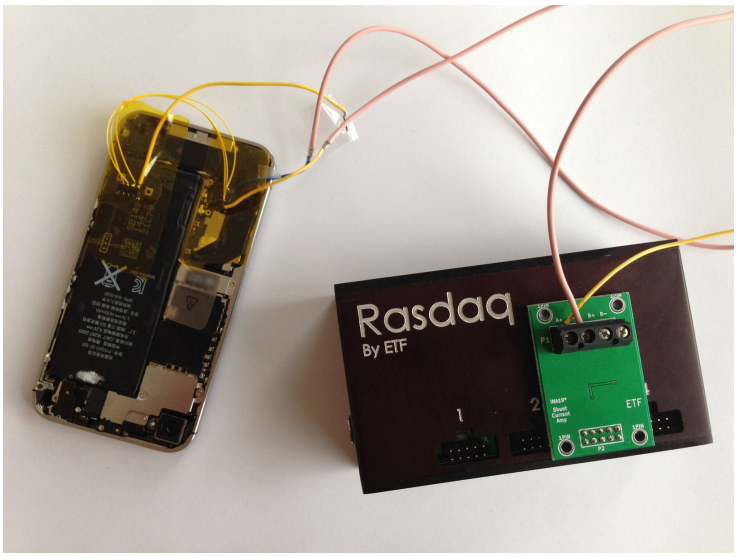


Figure 1. Our power measurement station connected to an iPhone 4S.

Abstract

Energy and power efficiency are becoming important topics within the graphics community. In this paper, we present a simple, straightforward method for measuring per-frame energy consumption of real-time graphics workloads. The method is non-invasive, meaning that source code is not needed, which makes it possible to measure on a much wider range of applications. We also discuss certain behaviors of the measured platforms that can affect energy measurements, e.g., what happens when calling `glFinish()`, which ensures that all issued graphics commands are finished executing. Measurements are done both on a smartphone and on CPUs with integrated graphics processors.

 ISSN 2331-7418

1. Introduction

The topic of both power and energy efficiency is increasingly important. It is recognized as a crucial design point for several reasons. Not only is the number of computing devices that are powered by batteries increasing, but heat dissipation, and thus power, is becoming one of the major design constraints for computer chips [Borkar and Chien 2011; Esmaeilzadeh et al. 2011].

Much work has been devoted to designing energy-efficient hardware and optimizing on a hardware level by, for example, performing operations with lower precision [Pool et al. 2008; Pool et al. 2011b; Pool et al. 2011a], power-gating, i.e., by turning off parts of the chip [Wang et al. 2011; Pool et al. 2011a], dynamic voltage and frequency scaling (DVFS) [Mochocki et al. 2006], or evolving the memory hierarchy [Fromm et al. 1997]. Even if some work advocates programmer input [Pool et al. 2011b], the main focus is still on hardware implementation.

Other work is focused on writing rendering software optimized for energy usage [Collange et al. 2009; Johnsson et al. 2012; Koduri 2011; Ma et al. 2013; Ribble 2012], suggesting that programmers should optimize for power and energy in combination with rendering times. Pool et al. [2010] and Johnsson et al. [2012] showed that it is not sufficient to measure at rendering times alone as an indicator for energy efficiency, since different algorithms can have similar rendering times, but substantially different energy consumption. Thus, it is necessary to also measure the power usage to accurately evaluate the energy used by an algorithm. Another path of research is to statistically model the power usage, as done by Nagasaka et al. [2010] and Pool et al. [2010].

However, there has been no real discussion or comparison of the measurement methods. After measuring power and energy efficiency in several projects, we have gathered advice for accurately measuring energy consumption using a simple method. Compared to our own previous method [Johnsson et al. 2012], we present and evaluate a simpler method, which avoids blocking the rendering thread, and is non-invasive. We also study the effects of blocking, to ensure that all submitted commands are executed until completion, and rendering a number of initial frames to achieve a steady state for the application before measuring commences. Detailed power measurements on both a mobile phone and on PCs with integrated GPUs are presented.

Similar to our previous method [Johnsson et al. 2012], the end result is per-frame energy measurements. Measuring over a number of frames to achieve a mean energy is easier and more straightforward; however, separating the energy on a per-frame basis is often beneficial, since it enables the possibility to correlate energy to other data collected per-frame and also to find bottlenecks and anomalies on a per-frame basis.

2. Hardware Setup

We have performed our measurements on either an iPhone 4S with a PowerVR SGX 543MP2 GPU, a PC with an Intel i7 3770k with an Intel HD4000 integrated GPU, or on a PC with an Intel i7 4850HQ with an Intel Iris Pro integrated GPU. The device is connected to a measurement station (Figure 1), similar to the one used by Johnsson et al. [2012], which is using two shunt current sensors that can accurately measure current up to 1 A for the iPhone 4S and through a Hall effect current sensor for the PC. Measurements on the iPhone are fed from a custom power supply with a known voltage of 3.9 volts. However, on the PC, as the ATX specification has a $\pm 5\%$ error tolerance, we simultaneously measure the voltage as well.

For the iPhone 4S, we intersect the current where the power supply connects to the battery. For the Intel i7 3770k, we intersect at the ATX $+12$ V power connector, which supplies the CPU with power, and for the Intel i7 4850HQ we intercept all current and voltage supplied to the motherboard. As a result of intersecting the power at different points, the measurements on each platform include different parts of that platform. This may need to be considered when drawing conclusions. During all measurements, we had the iPhone in airplane mode, i.e., all transmitting hardware was shut off. We also shut off the adaptive screen brightness and set the brightness to its lowest setting.

3. Measuring Method

Our previous method for measuring per-frame energy inserted timestamps at the beginning of each frame, and called `glFinish()` at the end, which blocked the rendering thread until all rendering commands were finished, so that only work done with respect to that particular frame was measured [Johnsson et al. 2012]. This may have some benefits, for example, if you are interested in only the energy for all the work done for a particular frame. In fact, something like this must be done if an architecture interleaves work over several frames, and if no energy from previous or future frames are to be included in the current frame's energy. For example, the Larrabee software rasterizer [Seiler et al. 2008] interleaved work over frames, and early mobile phone graphics architectures [Beets 2005] interleaved vertex shading (current frame) and pixel shading (previous frame) in sort-middle architectures.

However, in many cases, measuring from the start of one frame to the start of the next frame generates much more realistic energy consumption values, since that is how rendering is done most of the time, i.e., without blocking until all commands have finished executing and including interleaving effects (if any).

In the next section, we present a very simple method for measuring per-frame energy consumption by a real-time graphics application. The input is regularly spaced power measurements and a set of timestamps, either generated from the measured application or detected in the power measurements.

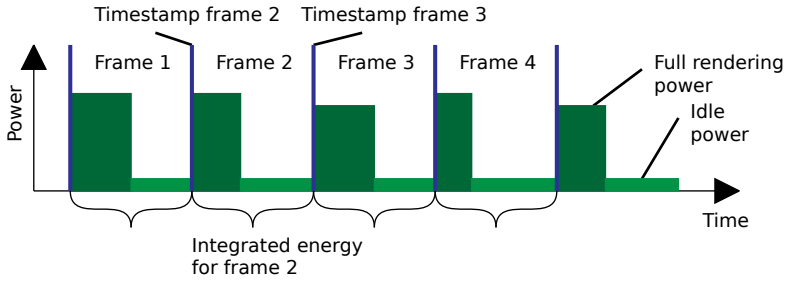


Figure 2. Our new method is illustrated here, with timestamps at the beginning of each frame. The integrated energy is from one frame's timestamp to the subsequent frame's timestamp. As can be seen, this also includes the idle energy when no rendering is done.

3.1. Our Method

Our new method records a timestamp at the beginning of each frame and integrates the power until the beginning of the next frame's timestamp. It, therefore, guarantees that no energy is unaccounted for during a sequence of frames, and it also includes possible effects from algorithms exploiting inter-frame coherency. A schematic of the method can be seen in Figure 2.

For workloads where the source code is not available, it is possible to detect the beginnings of the frames in a semi-automatic way, by searching for features on the curve. In most rendering workloads, it is easy for humans to visually detect the beginning of each frame. An example is shown in Figure 3. We have developed software

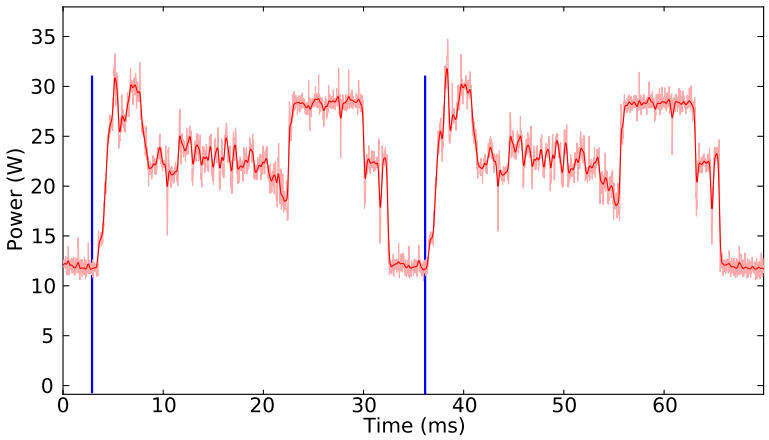


Figure 3. Two subsequent frames measured during a rendering on an Intel i7 3770k with the framerate capped to 30 frames per second. Note that the signature of the beginning of both frames are quite similar. By letting the user mark the beginning of one frame, an automatic search for the rest of the frame starts can be done.

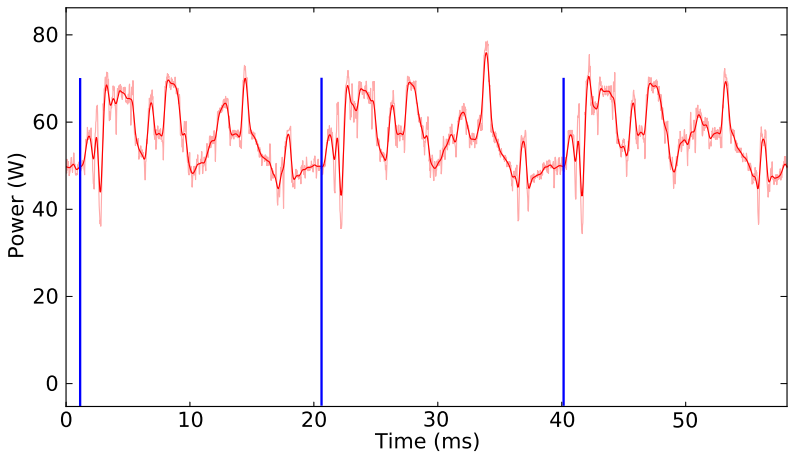


Figure 4. This sequence of three frames is a more difficult scenario than that shown in Figure 3. This is measured from Unigine's Heaven benchmark on an Intel i7 4850HQ with V-sync disabled.

that, given a manually selected sample, records a window of samples around it. It then searches the entire curve and marks samples with similar surroundings as the recorded window by comparing the summed square distance between the curves. As there most often are several subsequent samples where the surrounding is sufficiently similar, it is necessary to find a limited segment of the curve where the samples similarity is below a threshold and find the local minimum. As the characteristics of the frames can alter over the course of the rendering, it is often required to mark frame starts at different parts of the rendering and perform multiple searches.

We have been able to mark frames on a large collection of measurements with different configurations and from different applications. For PC platforms, it greatly simplifies the search if V-sync is enabled; however, for most cases, it is still possible to find frame starts with V-sync disabled. Figure 4 shows a power curve covering three frames, which are from a rendering of the Heaven benchmark by Unigine with V-sync disabled. For the most difficult cases, semi-automatic search is not possible. Our advice then is to enable V-sync, which is usually possible without changing the source code. For some difficult cases, it is still possible to find distinct features repeated in each frame. However, integrating between those features will provide an approximate result, since the distinct features may be offset from the frame start. In this case, our advice is to either accept the approximate result and avoid basing conclusion on frames close to abrupt energy changes, or enable V-sync.

The described technique is non-invasive since no source code is needed (so that timestamps can be generated and saved), and it makes it possible to measure energy consumption on many more applications. If source code is available, however, we can still insert our own timestamps.

3.2. Discussion

We also tried two other methods, which were less successful. One was a generalization of our previous method [Johnsson et al. 2012], where the same frame was rendered r consecutive times. The flush at the end was still used, but with higher values on r, the energy converged to the same as our new method. In another attempt, we tried to use b warm-up frames, followed by a timestamp, and then another r frames, followed by an ending timestamp at the beginning of a trailing frame. To get one measurement for one frame, $b+r+1$ frames were rendered. Some architectures may also detect inter-frame coherency, and either reuse calculations between frames when possible or avoid costly memory transactions [Han et al. 2009]. Note that for both of these methods, these optimizations will likely increase their level of success, which can skew the results. Also, both methods needed many more frames of rendering compared to our new method, which made them more time consuming.

4. Test Workloads

The rendering workload we used for our test on the iPhone 4S is a synthetic scene, consisting of between 23 and 62 geometric objects per frame. Each object is a pretessellated quadrilateral with 2048 triangles. We render a sequence of 24 frames, and the framerate is capped to 30 frames per second. The vertex shader performs three 4×4 matrix-vector multiplications, and the fragment shader performs two independent texture lookups, a dot product, and a normalization. We use two 256×256 RGB textures, sampled with linear minification and magnification filtering without mipmaps.

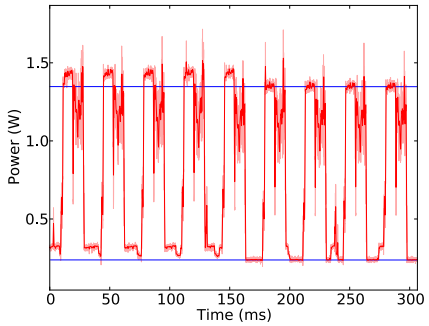
The main rendering workload used on PC is a camera path running through the Sponza scene, with 32 spotlights accumulated in a single rendering pass using forward rendering. It is capped at 30 frames per second and has been measured both with each frame ending with a blocking call to `glFinish()` to ensure that all rendering is finished, and without blocking. To show our ability to measure on real-world applications, we have also measured on the Heaven benchmark by Unigine. For our synthetic iPhone 4S application, we record timestamps in the application and synchronize them with the power curve. For our PC applications, both the Sponza scene and the Heaven benchmark, we detect the beginnings of the frames as described in Section 3.1.

5. Observations

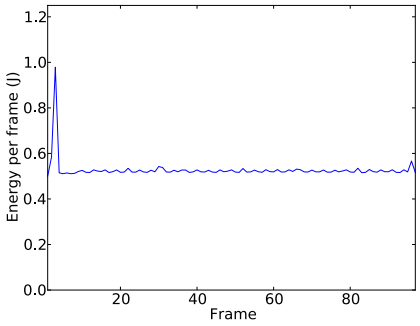
A set of important observations that we have made during several projects involving power and energy measuring for graphics workloads are presented in this section.

5.1. Application Launch

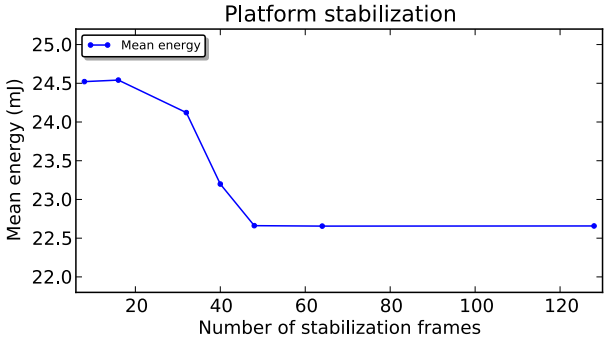
To get accurate results, we first note that it is important to avoid basing conclusions on the first few seconds after an application is launched. This is necessary since some platforms appear to have an increased energy consumption during that period. As seen in Figure 5(c), the difference between measuring immediately after launch, or waiting for the platform to settle, can be significant. This can also be seen in Figure 7(c), where our method, close to application launch has a rendering power and an idle



(a) A sequence of 9 frames, showing the power drop after a number of initial frames on an iPhone 4S.



(b) Rendering a number of identical frames on an Intel i7 3770k, showing that the energy settles after a few frames of rendering.



(c) Energy consumption on an iPhone 4S with different number of initial frames.

Figure 5. (a) A sequence of frames showing a sudden drop in power, that occurs a number of frames after the application is launched. Blue lines are added to ease comparison at both the idle level and top plateau level after the drop. (b) A sequence of identical frames on an Intel i7 3770k, measured directly after application launch, shows that the application settles after a few frames. (c) Mean per-frame energy for renderings on an iPhone 4S as a function of the number of disregarded initial frames. As can be seen, it is necessary to disregard about 48 frames after the application launch.

power that is approximately 0.05 W above the power obtained using our method with a frame measured a few seconds after the the application is launched (left). According to our measurements, the number of frames with an increased power is around 48 on an iPhone 4S (see Figure 5(c)). As the decrease in power is sudden (see Figure 5(a)), it is likely a result of a state change in the system on a chip, e.g., in its voltage and frequency scaling [Mochocki et al. 2006]. On an Intel i7 3770k, it is not necessary to perform more than a few frames for the application to settle. This can be seen in Figure 5(b), where a sequence of frames, directly after the application launch and with identical graphics workload, have been rendered and measured.

5.2. Effects of Pipeline Flushing

In Figure 6, we compare our new method (Section 3.1) and our previous method [Johnsson et al. 2012]. These methods differ in two ways. First, the previous method measures from the beginning of a frame until the frame has finished rendering. Our new method measures from the beginning of the frame until the beginning of the next frame. As a result, when using the previous method, the idle energy in between frames is not accounted for, and hence, the total energy of an entire rendering cannot be obtained by using a sum over per-frame energies. This is, however, possible with our new method, and this is usually what is desired.

The other difference is that the previous method calls `glFinish()` before recording the second, ending timestamp. `glFinish()` blocks until all commands submitted to the GPU have finished, which ensures that all rendering is performed before

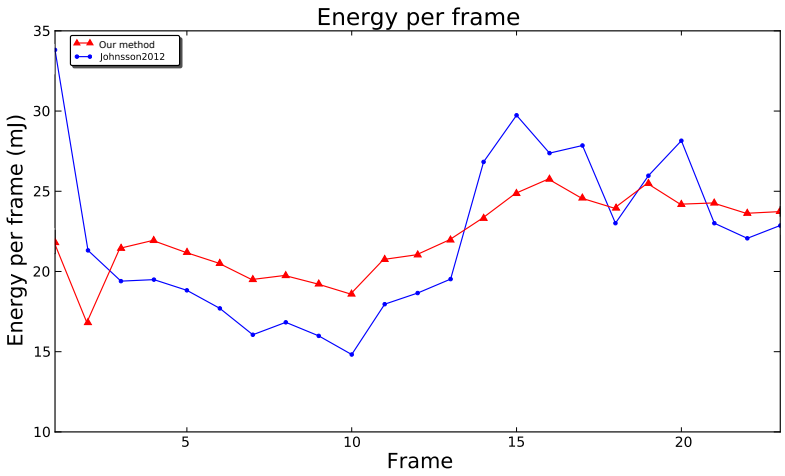
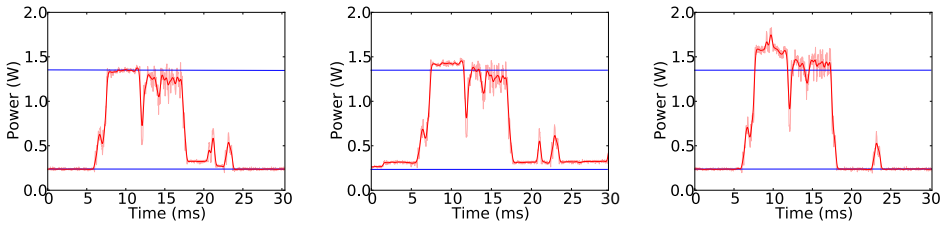


Figure 6. Comparison between the method used by Johnsson et al. [2012] and our new, improved method, showing measurements performed on an iPhone 4S, rendering our synthetic scene. It is interesting to note is that while the introduction of `glFinish()` increases the energy usage and not measuring the idle time between frames decreases it, the method used in Johnsson et al. [2012] can result in both higher and lower amounts of energy measured.



(a) Our method (b) At application launch (c) With `glFinish()`

Figure 7. Power measurements of frame 9 in the synthetic scene on an iPhone 4S. (a) A frame after the application has settled; (b) a frame rendered soon after application launch; (c) a frame ended with `glFinish()`, as in the method used by Johnsson et al. [2012]. Note that directly after the application launch, both the idle power between frames and the rendering power is approximately 0.05 W above the power of an identical frame after the application has settled. Horizontal blue lines, at the level of idle power and the plateau power level of our new method, are added to all three graphs, to aid the comparison. When calling `glFinish()` between frames, the idle power is the same as the idle power using our new method after the application has settled. However, the rendering power is significantly higher. The observable rendering times for all three frames are nearly identical.

returning. Surprisingly, the power usage on an iPhone 4S increases when `glFinish()` is used, which can be seen in Figure 7(c). This should be compared to our method, which is shown in Figure 7(a). We also note that the idle power is comparable. The only difference between these configurations is the introduction of `glFinish()` in the previous method. As `glFinish()` blocks the rendering thread, a consequence may be that the CPU is not able to sleep. In that case, that could be a possible explanation for the raised power consumption.

However, our measurements on an HD4000 graphics processor, both with and without `glFinish()`, revealed that there is very little difference. In our renderings with a capped frame rate, the mean energy per frame is 629.73 mJ with a pooled standard deviation of 21.07 mJ using `glFinish()`, and without flush, we obtain 630.18 mJ as mean energy per frame with a pooled standard deviation of 19.02 mJ. The difference in mean energy is 0.45 mJ, which is substantially lower than the pooled standard deviation, and hence a flush does not affect the measurements significantly.

Note also the small spikes at the end of the frames in Figure 7. These spikes are a semi-regular feature occurring on an iPhone 4S, where one spike occurs approximately 33 ms apart. However, the frequency differs slightly from the capped frequency of the rendering, and thus the spikes are not occurring at the same position within the frames. There are also spikes of this size that have no observable frequency or pattern. This is, with a high probability, impact from the operating system. However, as they have a relatively small impact on the measured energy (approximately

0.2 mJ per spike), and they usually occur $1-2$ per frame, the impact on the measurements is of no major concern. It is important not to base any conclusions on the placement or quantity of spikes in the presented frames, as they are both equally common within all methods and are irregularly positioned within the frames.

5.3. Operating System Interference

We also studied the standard deviation on the iPhone 4S as a function of the number of repeated measurements. Sometimes, we saw that the standard deviation increased abruptly even though the number of repeated measurements increased. Investigating this more closely, we saw that there were outliers that stretched over several frames in each measurement it occurred. Examples of such frames can be seen in the top of Figure 8, where the upper left is a normal frame and the upper right has an outlier.

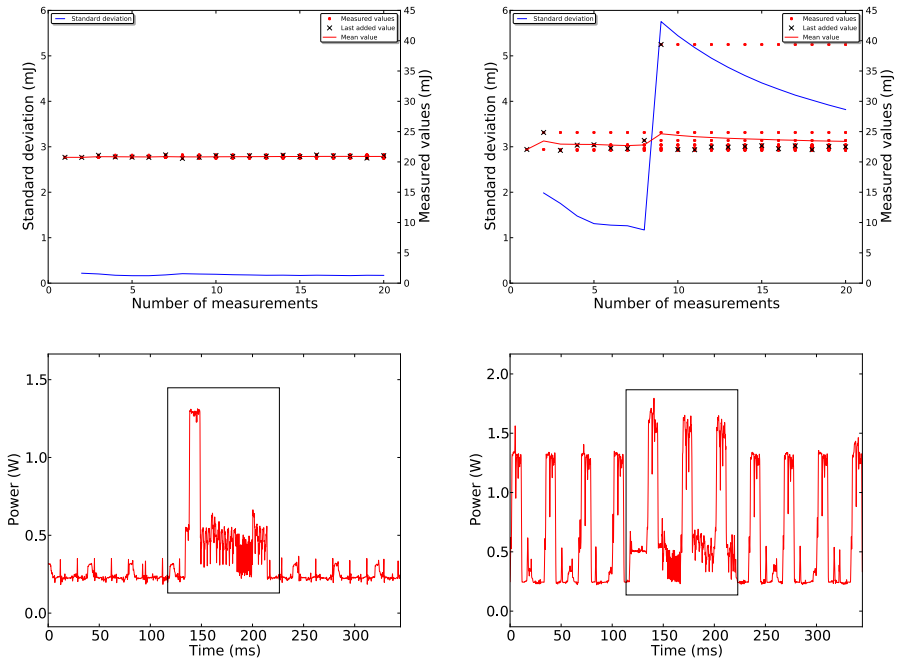


Figure 8. Top: standard deviation (blue curve) as a function of the number of performed measurements. Note that both figures use the same measurements, but represent different frames in the sequence. Top left: standard deviation slightly decreasing with more measurements. Top right: a decreasing standard deviation followed by a rapid increase as a result of an outlier. Bottom left: power curve from the iPhone 4S while it is not running any application. These spikes, with an approximate duration of 90 ms, occur every five seconds. Bottom right: each frame is supposed to be identical, however, a spike (bottom left) occurs in the middle and is overlaid on top of the graphics power. As the duration of the spike is approximately 90 ms, it affects 3-4 frames at 30 fps. Those spikes are the root cause of standard deviation increases (top right).

Examining the actual power curves of these measurements, we discovered groups of frames with a significantly higher power signature, seen in the bottom right of Figure 8. In the bottom left part of the same figure, we show a power measurement on the iPhone 4S without any applications running. As can be seen, similar behavior (distinct spikes) were measured. These spikes occur exactly five seconds apart.

Similarly, there are frames and series of frames when running on Intel i7 3770k, where the power or rendering time, and sometimes both, are substantially higher. However, there is not an easy pattern to distinguish, as on the iPhone 4S.

There are several methods to handle these outliers, depending on the purpose of the measurement. If the sought energy is the actual energy used when rendering, the outliers represent real energy usage that should be included. In that case, more measurements might be needed to get a stable result. If the purpose of the measurement is to isolate the actual graphics algorithm, it is possible to detect frames with unacceptably high standard deviation, identify the outlier, and exclude it from the measurement.

6. Conclusion

We have described a method which we have used extensively in our research, for measuring per-frame energy consumption for graphics workloads. The method consists of three straightforward steps:

1. Record power at a high frequency;
2. Find or sync a set of frame beginnings in the measured power recordings;
3. Integrate between frame beginnings to obtain per-frame energy.

Also, based on our experience with measuring energy, we have formed a set of best practices to avoid the pitfalls we ran into. These best practices include

- Know your platform;
- Avoid changes to the measured workload;
- Disregard the first few seconds of the workload;
- Be aware that the operating system might interfere.

Depending on the platform, changes to the software can have substantial effect on the energy consumption, for example, we have discovered that using `glFinish()`, as a mean for isolating the consumption of a single frame, raises the power consumption. General advice is to avoid changing the software at all, if possible, as it is hard to ensure that a change does not affect the power consumption. Our method does not depend on changes to the software. We have also experienced that on some platforms,

applications have a higher power for a brief period after launching. If the expected result is the general rendering power for an algorithm or applications, it is advisable not to measure, or not to base conclusions on, the first seconds after launch. In addition, we have also observed that the operating system can start some process that also substantially increases the measured energy.

However, these are only the pitfalls we have encountered. The most important advice is to get to know the measured platform, as that is the only way to avoid pitfalls that have not yet been encountered.

Acknowledgements

Thanks to Jim Nilsson, Chuck Lingle, and the Advanced Rendering Technology team at Intel. Tomas Akenine-Möller is a *Royal Swedish Academy of Sciences Research Fellow* supported by a grant from the Knut and Alice Wallenberg Foundation. The Heaven benchmark measured in this paper are courtesy of Unigine.

References

BEETS, K. 2005. Developing 3D Applications for PowerVR MBX Accelerated ARM Platforms. *Information Quartely, 4*, 3, 26–34. http://www.iqmagazineonline.com/magazine/pdf/v_4_3_pdf/v_4_3_pg-26-34.pdf. 62

BORKAR, S., AND CHIEN, A. A. 2011. The Future of Microprocessors. *Communications of the ACM, 54*, 5, 67–77. http://dl.acm.org/citation.cfm?id=1941507. 61

COLLANGE, S., DEFOUR, D., AND TISSERAND, A. 2009. Power Consumption of GPUs from a Software Perspective. In *9th International Conference on Computational Science*, Springer-Verlag, Berlin, Heidelberg, 914–923. http://dl.acm.org/citation.cfm?id=1560861. 61

ESMAEILZADEH, H., BLEM, E. R., AMANT, R. S., SANKARALINGAM, K., AND BURGER, D. 2011. Dark Silicon and the End of Multicore Scaling. In *38th International Symposium on Computer Architecture*, ACM, New York, NY, 365–376. http://dl.acm.org/citation.cfm?id=2000108. 61

FROMM, R., PERISSAKIS, S., CARDWELL, N., KOZYRAKIS, C., MCCAUGHY, B., PATTERSON, D., ANDERSON, T., AND YELICK, K. 1997. The Energy Efficiency of IRAM Architectures. In *International Symposium on Computer Architecture*, ACM, New York, NY, 327–337. http://dl.acm.org/citation.cfm?id=264214. 61

HAN, K., FANG, Z., DIEFENBAUGH, P., FOR, R., IYER, R. R., AND NEWELL, D. 2009. Using Checksum to Reduce Power Consumption of Display Systems for Low-Motion Content. In *IEEE International Conference on Computer Design*, IEEE Press, Piscataway, NJ, USA, 47–53. http://dl.acm.org/citation.cfm?id=1792366. 65

JOHNSSON, B., GANESTAM, P., DOGGETT, M., AND AKENINE-MÖLLER, T. 2012. Power Efficiency for Software Algorithms running on Graphics Processors. In *High Performance Graphics*, Eurographics Association, Aire-la-Ville, Switzerland, 67–75. http://dl.acm.org/citation.cfm?id=2383806. 61, 62, 65, 67, 68

KODURI, R. 2011. "Power" of Realtime 3D Rendering. In *Beyond Programmable Shading (SIGGRAPH course)*, ACM, New York, NY. http://bps11.idav.ucdavis.edu/talks/03-powerOf3DRendering-BPS2011-koduri.pdf. 61

MA, X., DENG, Z., DONG, M., AND ZHONG, L. 2013. Characterizing the Performance and Power Consumption of 3D Mobile Games. *Computer, 46*, 4, 76–82. http://dx.doi.org/10.1109/MC.2012.190. 61

MOCHOCKI, B., LAHIRI, K., AND CADAMBI, S. 2006. Power Analysis of Mobile 3D Graphics. In *Proceedings of the Conference on Design, Automation and Test in Europe*, European Design and Automation Association, Leuven, Belgium, 502–507. http://dl.acm.org/citation.cfm?id=1131617. 61, 67

NAGASAKA, H., MARUYAMA, N., NUKADA, A., ENDO, T., AND MATSUOKA, S. 2010. Statistical Power Modeling of GPU Kernels using Performance Counters. In *International Conference on Green Computing*, IEEE, Piscataway, NJ, 115–122. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.232.4758. 61

POOL, J., LASTRA, A., AND SINGH, M. 2008. Energy-Precision Tradeoffs in Mobile Graphics Processing Units. In *International Conference on Computer Design*, IEEE, Piscataway, NJ, 60–67. http://www.cs.unc.edu/~jpool/research/ICCD08/. 61

POOL, J., LASTRA, A., AND SINGH, M. 2010. An energy model for graphics processing units. In *International Conference on Computer Design*, IEEE, Piscataway, NJ, 409–416. http://www.cs.unc.edu/~jpool/research/ICCD2010/. 61

POOL, J., LASTRA, A., AND SINGH, M. 2011. Power-Gated Arithmetic Circuits for Energy-Precision Tradeoffs in Mobile Graphics Processing Units. *Journal of Low Power Electronics, 7*, 2, 148–162. http://www.cs.unc.edu/~jpool/research/JOLPE2010/. 61

POOL, J., LASTRA, A., AND SINGH, M. 2011. Precision Selection for Energy-Efficient Pixel Shaders. In *High Performance Graphics*, ACM, New York, NY, 159–168. http://www.cs.unc.edu/~jpool/research/HPG2011/. 61

RIBBLE, M. 2012. Power Friendly GPU Programming. In *Beyond Programmable Shading (SIGGRAPH course)*, ACM, New York, NY. http://bps12.idav.ucdavis.edu/talks/05_ribblePowerRendering_bps2012.pdf. 61

SEILER, L., CARMEAN, D., SPRANGLE, E., FORSYTH, T., ABRASH, M., DUBEY, P., JUNKINS, S., LAKE, A., SUGERMAN, J., CAVIN, R., ESPASA, R., GROCHOWSKI, E., JUAN, T., AND HANRAHAN, P. 2008. Larrabee: A Many-Core x86 Architecture for Visual Computing. *ACM Transactions on Graphics, 27*, 3, 18.1–18.15. http://dl.acm.org/citation.cfm?id=1360617. 62

WANG, P.-H., YANG, C.-L., CHEN, Y.-M., AND CHENG, Y.-J. 2011. Power Gating Strategies on GPUs. *ACM Transactions on Architecture and Code Optimization, 8*, 3, 13:1–13:25. http://dl.acm.org/citation.cfm?id=2019612. 61

Author Contact Information

Björn Johnsson
Department of Computer Science
Lund University
Box 118, 221 00 Lund, Sweden
bjorn.johnsson@cs.lth.se

Tomas Akenine-Möller
Department of Computer Science
Lund University
Box 118, 221 00 Lund, Sweden
tomas.akenine-moller@cs.lth.se

Björn Johnsson and Tomas Akenine-Möller, Measuring Per-Frame Energy Consumption, *Journal of Computer Graphics Techniques (JCGT)*, vol. 3, no. 1, 60–73, 2014
http://jcgt.org/published/0003/01/03/

Received:	2013-08-23		
Recommended:	2013-11-18	Corresponding Editor:	Marc Stamminger
Published:	2014-03-05	Editor-in-Chief:	Morgan McGuire