Power Consumption Model of a Mobile GPU Based on Rendering Complexity

Jarkko M. Vatjus-Anttila, Timo Koskela, Seamus Hickey
Center for Internet Excellence
P.O. Box 1001
FI-90014 Unversity of Oulu
Email:firstname.lastname@cie.fi

***Abstract*—This paper presents a mathematical model for predicting power consumption of a mobile device when it is rendering 3D graphics. The model is based on 3D primitives (triangles, render batches, texels), and hence is hardware agnostic. With the model, a complexity of any given 3D scene can be predicted already at a production phase without access to the actual target hardware. This paper describes how the power consumption model is derived. The model is verified with measurements of real-world content and hardware. With the given hardware, 3D data and given verification scenarios, the model is able to predict the total power consumption with an error ranging from 0.3% to 3.2%.**

Rendering, mobile device, power consumption, mathematical model

I. Introduction

Modern mobile graphics processing units (GPU) implement a fully programmable graphics pipeline, which allows a flexible way of graphics production [**?**]. Due to this fact, it is increasingly easy to implement 3D content on mobile devices, since the application programming interfaces (API) are similar to a standard PC. However, moving such a content to a mobile device is troublesome due to the complex nature of 3D content, as well as to the fact that mobile devices are energy limited [**?**]. Still, the users want to use their services with their mobile devices more than with regular PCs, as indicated also by facebook[1] in their fourth quarterly report in 2012. This creates a need to produce 3D services fluently and power efficiently on mobile devices.

In a programmable graphics API (such as OpenGL ES 2), the flexibility is based on an ability to run arbitrary program code (shaders) for each vertex and fragment. Geometry is defined as groups of points (vertices) in a 3D space together with geometric primitives built from them (e.g. triangles or lines). These groups are called geometry batches. In addition, various kinds of texture maps can be used to modulate the 3D surfaces during rendering. The complexity of the above mentioned parameters affect the rendering task of the mobile GPU, and eventually contribute to the device overall power consumption.

It would be beneficial to understand the behavior of a mobile GPUs and how their power consumption evolves based on the complexity parameters of a 3D scene (such as the number of triangles) in a hardware agnostic way. The ability to predict the power consumption of a 3D scene at the content production phase would allow the user, or a similar agent, to select, remove or adapt 3D scenes based on their power consumption on the mobile device. As a consequence, this ability would allow designing better performing 3D scenes contributing to both longer battery life of mobile devices and better user experience.

In this paper, a mathematical model is presented which is based on the complexity parameters of the 3D scene (vertices, primitives, texels). This mathematical model can be used to calculate the expected power consumption of different 3D scenes at the production phase and consequently to provide a power use profile which can be used by other applications. For most API's, the power consumption model of the 3D scene does not have to be 100% accurate, but should be accurate within certain limits.

The rest of the paper is organized as follows: First, a view on the existing literature is presented. Second, a hypothesis of a power consumption model is presented. Third, a measurement setup is presented and finally, a verification tests are done with real-world material to verify how the mathematical model compares to target hardware. Finally, a discussion of the results and the future work is provided.

II. Previous Research

In a mobile device, nearly all constrains circulate around its physical size limitations. The size of the device limits the size of any physical component inside it, such as a display or a battery. Batteries have a relationship between their capacity and physical size. Eventually, the capacity of a battery is defined by its energy density and physical volume. The problem primarily is that even though the energy density is increasing all the time as technology evolves, it is not doing so fast enough, given that the physical volume is not increasing. In other words, roughly 4% annual increase in the energy density is not enough to satisfy the increased processing power needs. [**?**] [**?**] [**?**] Also, at the same time, users demand better services and longer battery life from their mobile devices. As discovered by Bloom et al. [**?**] in their user acceptance research, the users expect that their mobile devices are able to operate longer, and are unsatisfied when this does not happen. Similar trends in users' expectations were likewise identified by Nurminen et al. [**?**] in their more recent research on users' expectations about their mobile device battery lifetime behavior.

There exist a number of approaches to measure how a mobile device behaves in various situations from the power consumption perspective. Thiagarajan et al. [**?**] measure the

[1]http://investor.fb.com/releasedetail.cfm?ReleaseID=736911

978-0-7695-5090-9/13 $26.00 © 2013 IEEE
DOI 10.1109/NGMAST.2013.45

power consumption through a web browsing use case. As this is a complete end-to-end use case, the measurements include power consumption of all parts of the mobile device. Balasubramanian et al. [**?**] show similar research from a more generic network usage perspective by comparing different network technologies against each other. Likewise, Gil and Trezentos [**?**] focus on network transmissions, but take a closer look at exact transmission data formats how they affect the overall power consumption. While all these are related to 3D based networked services, there does not seem to exist detailed information available on how the previously mentioned research translates into usage of 3D graphics. There are detailed research and measurements available, done by Mochocki et al. [**?**] about how a number of 3D graphics related low level rendering parameters affect the overall power consumption. Likewise, Pool et al. [**?**] show how the power efficiency during 3D rendering can be gained by directly altering the calculation precision. Each of these studies show important results as such in their corresponding field, but it takes more to be able to formulate a complete model how the low level parameters really affect the overall power consumption in a long term, during 3D graphics reproduction.

Johnsson et al. [**?**] show a comprehensive and external measurement setup for power consumption measurements. The motivation for building such a setup is in being able to measure the rendering process of various systems, including mobile devices. In our research, the focus is in the creation of an power consumption model while using a similar kind of setup as in [**?**].

III. Mobile graphics reproduction and Power Consumption

A. Graphics reproduction on mobile devices

Figure 1 presents the major building blocks and data flow of a programmable mobile graphics unit (GPU) [**?**]. OpenGL ES 2 API provides functionality to control mobile GPU memory usage, shader program management and various other stages of the graphics pipeline. From a data flow perspective, the first stage is vertex processing followed by primitive assembly. Fragment shading comes after geometry processing stages and is followed by various per-fragment operations before the given fragment is drawn on the display.

From the API usage perspective, the process of GPU utilization is divided into the following (chronological) steps. First, the application gathers groups of geometry, which share the same surface material information (and hence share the same shader programs and their global parameters). These groups are called ”batches”. Second, the batches are passed to the GPU, and the vertex shader and primitive assembly stages start to execute for each vertex. At this phase, all vertices are processed by the shader, but 3D primitives (lines, triangles, etc.) are discarded based on their visibility and orientation status. Third, the remaining visible primitives are passed to the fragment shader and various per-fragment operations, which determine whether the given fragment should be written to the frame buffer or not. If there are texture maps involved, those

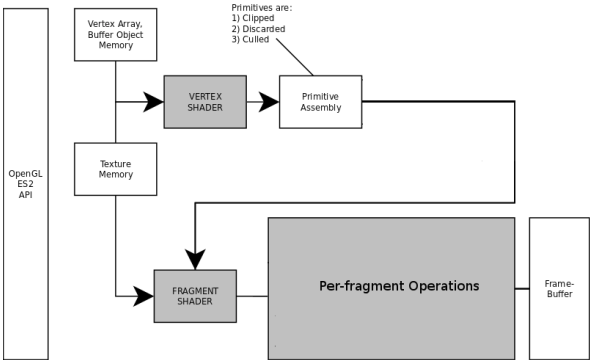

Fig. 1. OpenGL ES 2 programmable graphics pipeline [**?**]

are usually sampled at the fragment shading stage, even though the vertex shader can utilize them as well too puhekielinen [**?**]. Finally, if the fragment passes all of the per-fragment tests, it is written to the frame buffer after which it appears as a lit pixel on the display.

There are various steps in the graphics pipeline which will affect the total processing workload of the mobile GPU. The favored approach is to drop the graphics primitives as early as possible from the pipeline, because this allows the GPU to skip unneeded processing steps. This sets requirements onthe input data and how it is organized, because badly organized data will cause the GPU to do obsolete (non-visible) work. However, from this papers perspective, it is assumed that the data itself is already in its optimal form, and only the incoming graphic primitives matter.

B. Power consumption model

From the power consumption point of view, it is assumed that there is a hardware specific lower (P_{min}) and upper (P_{max}) limit. P_{min} is achieved when nothing is rendered (i.e. a 3D scene is rendered without any content in it). Likewise, the P_{max} is achieved when the hardware works close to a 100% duty cycle. The rendering of 3D primitives cause a contribution (P_{prim}) to the total consumption (P_{tot}), which is accumulated to the idle consumption. Eventually, the consumption saturates towards the maximum hardware consumption limit.

Based on the fact that 3D scenes are built from geometric primitives (mostly triangles), surface image data (texture image pixels, i.e. texels) and issued render calls (batches), it is assumed that the overall power consumption can be predicted using these three key primitives as parameters. These primitives were chosen since they describe the 3D scene complexity exactly, and at the same time, are hardware agnostic. Equation symbols of *t*, *b* and *i* are chosen for the number of triangles, render batches and addressed texels, respectively. Hence, in general, the form of the consumption model can be expressed as:

$$P_{tot}(t, b, i) = P_{min} + P_{prim}(t, b, i) \quad (1)$$

The term P_{prim} from (1) can be expressed as a weighted sum of its contributions from triangles (P_t), render batches (P_b)

and addressed texels (P_i), each with their dedicated weight coefficients W_t, W_b and W_i, respectively. The weight coefficients exist, because the mentioned primitives cause a contribution in co-operation, and are not simply accumulated. The sum of the coefficients will equal to one, and second, the weighted sum of the contributions will not exceed the maximum hardware consumption P_{max}. Equation (1) can be written as follows:

$$P_{tot}(t,b,i) = P_{min} + W_t P_t(t) + W_b P_b(b) + W_i P_i(i) \quad (2)$$

Since the consumption saturates towards the maximum consumption limit in a real hardware, the consumption is expected to behave as inverted exponential function when the rendering load is increased. Hence, the terms P_t, P_b and P_i in (2) can be presented in general form as:

$$P_x(X) = (P_{max} - P_{min})(1 - \mathrm{e}^{\frac{-\lambda_x * X}{X_n}}) \quad (3)$$

where P_{max} and P_{min} are maximum and minimum hardware consumption limits, respectively, X is the amount of selected primitive (triangles, batches or texels), X_n is a factor which scales the number of the selected primitive into the range of [0..1] and allows saturation when exceeding the limit of 1. Finally, λ_x is a primitive specific slope control for the specific exponential function.

Combining (2) and (3) with (1), the final form for the consumption model is expressed as:

$$\begin{aligned} P_{tot}(t,b,i) = P_{min} + (P_{max} - P_{min})[W_t(1 - \mathrm{e}^{\frac{-\lambda_t t}{X_t}}) + \\ W_b(1 - \mathrm{e}^{\frac{-\lambda_b b}{X_b}}) + W_i(1 - \mathrm{e}^{\frac{-\lambda_i i}{X_i}})] \end{aligned} \quad (4)$$

For the definition of the complete consumption model, the hardware specific values of P_{min}, P_{max}, W_t, W_b, W_i, X_n, λ_t, λ_b *and* λ_i need to be defined. This can be done with a static measurement setup, as explained in the next section.

IV. Measurement setup and methods

In rendering, power savings are primarily gained by letting the hardware sleep as much as possible between consecutive frame updates. The faster the data is organized and submitted for the GPU, the faster the GPU is able to draw the data on the display, and the greater is the amount of the time that can be used for sleeping before the next frame update. For this research, a fixed frame update time of 50ms was used, which results in 20 frame updates per second (FPS). If the CPU and GPU manage to handle a single frame faster than the defined limit, the remaining time was used to put the hardware to sleep. Update rate of 20 FPS was considered enough for the use-cases used in this research to produce a fluent animation in 3D space, and at the same time, allow measurement of the power savings.

A. Measurement devices and tools

All of the measurements were rendered using a custom built C++/OpenGL ES 2 rendering engine, which is available as an open source solution. [**?**] The custom rendering engine allowed full control over the GPU parameters and geometry culling methods. It also implemented an efficient view frustum culling algorithm [**?**] to ensure that a minimal amount of off-screen data was drawn. The rendering engine also offered a service for recording the rendered primitives, which were then used for calculating the power consumption estimates. All of the measurements were run using a TrimSlice NVidia Tegra2 hardware with Ubuntu Linux OS. The rendering resolution for all test cases was 1280x720. The consumed power was measured using an external UNI-T UT61E digital multimeter, which allowed 500ms sampling interval and data recording. All power measurements excluded any network interfaces, external hardware peripherals and the display. Only the power consumption of TrimSlice hardware is included in the measurements.

B. Definition of hardware specific parameters

For enabling the prediction of the power consumption, the first step was to define the hardware specific parameters for the power consumption model. In this case, the power consumption contribution of each of the primitives (triangles, batches and texels) were measured separately with five levels of complexity. In each complexity levels, a synthetic data was created to force the hardware to work at a certain level. For example, when measuring triangles, a mesh with a required number of triangles was created artificially, which satisfied the number of triangles requirements. The complexity levels were chosen to utilize roughly 0%, 5%, 25%, 50% and 100% of the hardware rendering capabilities. The 100% load was assumed when the rendering with the corresponding primitive performed at 5 FPS in the target hardware, and the other complexity levels were scaled down in relation to the result, as presented in Table **??**. Each complexity level produced a value for the consumed power, which was eventually used in calculating the hardware specific constants for the mathematical model.

C. Verification test case

In the verification test case, a 3D model of a city was rendered and at the same time the power consumption was measured. These measurements were then compared to the mathematical model. In this case, the 3D content was rendered in three different scenarios: 1) the camera circulates around the city, viewing 20%-100% of the city depending on the camera location and orientation, 2) the camera flies through the city, starting from a 100% view and ending with a 0% view, and 3) the camera rotates in the center of the city around the world up -axis viewing roughly 40% of the city at a time.

Finally, the verification results were compared to the mathematical model. For this, two different comparisons were made. First, the comparison was done directly with the measured results and second, with compensated results. It was anticipated that due to the depth testing and the triangle backface culling operations conducted by the GPU, not all requested triangles necessarily get drawn, and hence do not directly contribute to the power consumption. To compensate for this, an estimation was calculated with lowered triangle, batch and texture counts. The hypothesis was that 50% of the 3D content could be left

TABLE I
NUMBER OF RENDERED PRIMITIVES IN STATIC MEASUREMENT TEST CASE

	Test 1	Test 2	Test 3	Test 4	Test 5
Level	0%	5%	25%	50%	100%
Triangles	1	79202	387069	774192	1534384
Batches	1	2	8	16	32
Texels	1	1048576	4194303	8388608	12482912

TABLE II
3D CITY MODEL COMPLEXITY

Primitive type	Number in the 3D scene
Total triangles	628436
Total batches	59
Total texels	757888

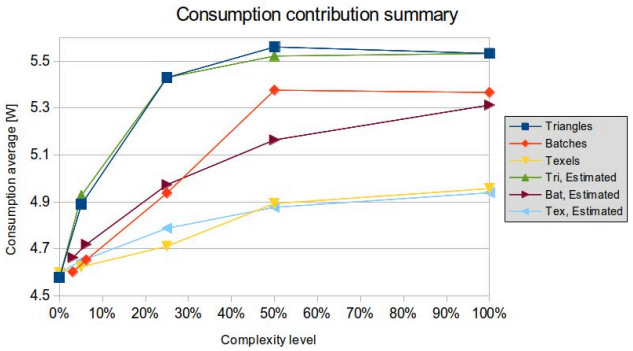

Fig. 2. 3D primitive power consumption contributions and their estimations.

unacknowledged due to backface triangle culling (due to 3D buildings being box shaped and hence always roughly 50% back facing), and another 10% due to depth testing (depending on the draw order objects behind other objects may or may not get drawn). Hence, in total, a compensation of 45% of the 3D content was considered to be a good estimation to take backface culling and depth testing effects into account. The results are presented together with the direct comparison.

Each test case was run ten times, and the results were averaged. A single test run lasted 30 seconds. In total, counting together all combinations of test cases, 230 test runs were executed totaling in 6,900 seconds of rendering. The characteristics of the 3D city model are as presented in Table **??**.

V. TEST RESULTS

A. Hardware specific parameters

First, the measurement setup was used to determine the hardware specific parameters for the power consumption model. Table **??** summarizes the complexity steps for each of the chosen primitives: triangles, batches and texels. Fig. 2 summarizes the measurement results. The largest recorded power consumption average was set as the hardware maximum consumption limit (P_{max} = 5.531W), which was derived from the triangle test case. Minimum consumption (P_{min}) was measured with an empty scene and resulted in 4.552W. Additionally, the scaling factors X_n for each primitive were set to the maximum number of each primitive respectively: 1534384 for triangles, 32 for batches and 12482912 for texels. Using these measurements, (5), (6) and (7) were defined:

$$P_t(t) = (5.531 - 4.552)(1 - e^{\frac{-\lambda_t * t}{1534384}}) \quad (5)$$

$$P_b(b) = (5.531 - 4.552)(1 - e^{\frac{-\lambda_b * b}{32}}) \quad (6)$$

$$P_i(i) = (5.531 - 4.552)(1 - e^{\frac{-\lambda_i * i}{12482912}}) \quad (7)$$

The slope controls λ_t, λ_b and λ_i were calculated for each of the functions by estimating the total error. Each equation was evaluated in 5 points (complexity levels). For each primitive the total error is hence given by (8).

$$\epsilon_x = \sum_{n=0}^{4} |M_n - (P_{min} + P_x(X_n))| \quad (8)$$

Where M_n are the reference measurement points and P_x is the estimation equation for the given primitive, as described above. This is a minimization problem, where we wanted to find a λ_n for each equation in a way that it would minimize the total error for each of the complexity levels. By solving the equations for each primitive, we get λ_t = 8.917, λ_b = 2.662 and λ_i = 3.004. Graphical presentation of the original measurements and the estimation equations are presented in Figure 2.

The final step was to solve weighting co-efficients W_p, W_b and W_i. As explained, it was assumed that the cumulative contribution of the three co-efficients will not exceed the maximum hardware consumption level (P_{max}) and the sum of the weight co-efficients equals to one. Using the complexity level 100% figures from the table **??**, the weighting co-efficients of W_t = 0.216, W_b = 0.260 and W_i = 0.524 will satisfy the mentioned conditions.

B. Verification with a 3D city test scene

In order to verify the created mathematical model, measurements were performed using a 3D model of a city to verify the created mathematical model. As defined, three test cases were rendered: camera-circle, camera-flyby and camera-rotation. For each case, there are figures 3, 4, and 5 presenting the results respectively. In each figure, three curves are presented; first, the actual measurement; second, the corresponding result from the mathematical model; and third, results from the compensated mathematical model. The errors of the estimation and compensated equations are presented in Table **??**. In the table , it can be seen how the compensated estimation model work more accurately keeping the error in 3.2% range. Without the compensation, the error range is higher being in 6.3% range.

Interestingly, the mathematical model works most accurately (0.3% error) in the camera rotation test case. This test case differs from the other two by having a relatively constant amount of 3D content visible all the time during the rendering process. Circular and fly through camera routes have higher variability on the visible 3D content, but they also include sudden changes in the visible objects, i.e. large parts of the 3D scene can be hidden or revealed at once. Such a phenomenon

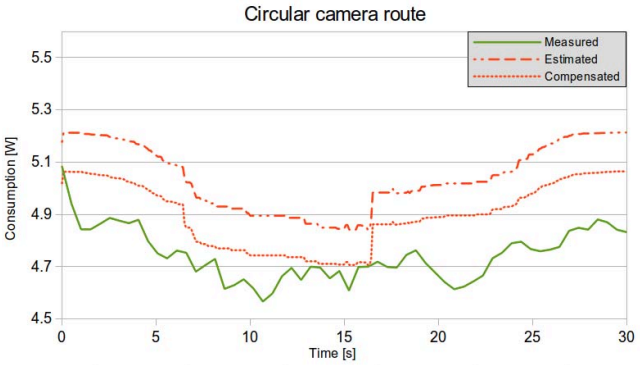

Fig. 3. Circular camera route power consumption measurement,estimation and compensation

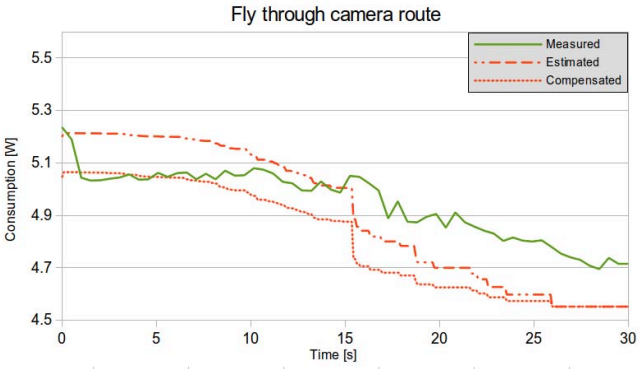

Fig. 4. Fly through camera route power consumption measurement,estimation and compensation

can be seen in the middle of Figure 3, for example. Even though the mathematical model reacts to such sudden changes immediately, in reality, the hardware does not. This should be taken into account in the future research.

VI. Conclusion

In this paper, a mathematical model was presented which is able to estimate the power consumption of a given mobile hardware using 3D scene complexity characteristics (number of triangles, render batches and addressed texels) as parameters. The model is hardware specific, but includes parameterization which allows it to be fit for any required hardware, hence making it highly reusable. In this paper, the model was verified with Trimslice NVidia Tegra2 hardware by rendering a 3D city model.

Fig. 5. Rotating camera route power consumption measurement,estimation and compensation

TABLE III
Estimation errors against the actual measurement

Test case	Direct est.	Compensated est.
Camera-circle	6.25%	3.21%
Camera-flyby	0.74%	2.82%
Camera-rotation	3.33%	0.28%

The model was used to estimate the consumption of the given 3D test scene with the given hardware. Two estimations were calculated, one directly with the rendered primitives, and another with compensating the number of primitives due to expected GPU per-fragment optimizations (triangle backface culling and depth testing). Without the compensation, the model was able to predict the power consumption within 6.3%, and with compensation, within 3.2% error range. This level of accuracy can be considered to be accurate enough to open way for further research on even more accurate mathematical models for the purpose. This model allows content creators to estimate the power consumption of the 3D content productions without actually having any access to the actual target hardware. Being based on 3D primitive types, integration of such an application in the existing 3D modeling programs would be trivial.

The model works most accurately when the amount of visible 3D content does not change too rapidly, and stays relatively constant. In the verification cases, sudden large changes in the visible 3D content make the mathematical model react immediately, but in reality, the hardware does not do that.

The future work would include adding a frame update rate as an estimation parameter, as well as deepening the knowledge of GPU per-vertex and per-fragment operations, which affect the total power consumption of the mobile device. Verifying the same model with a varying and heterogeneous set of mobile devices would be interesting as well. These, and other types of continuance work, can be performed with the open sourced rendering code [**?**].

VII. Acknowledgments

The authors would like to thank the staff at the Intel and Nokia Joint Innovation Center for their help and assistance. This work has been carried out in the Tekes Chiru Project.

References

[1] D. Ginsburg A. Munshi and D. Shreiner. *OpenGL ES 2.0 Programming Guide*. Addison-Wesley, 2009.
[2] U. Assarsson and T. Möller. Optimized view frustum culling algorithms for bounding boxes. *J.Graph.Tools*, 5(1):9–22, jan 2000.

[3] K. Lahiri B. Mochocki and S. Cadambi. Power analysis of mobile 3d graphics. In *Proceedings of the conference on Design, automation and test in Europe: Proceedings*, DATE '06, pages 502–507, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association.

[4] L. Bloom, R. Eardley, E. Geelhoed, and M. Manahan. Investigating the relationship between battery life and user acceptance of dynamic, energy-aware interfaces on handhelds. MobileHCI, LNCS 3160:13, 2004.

[5] B. Gil and P. Trezentos. Impacts of data interchange formats on energy consumption and performance in smartphones. In *Proceedings of the 2011 Workshop on Open Source and Design of Communication*, OSDOC '11, pages 1–6, New York, NY, USA, 2011. ACM.

[6] Google Inc. Android dashboards, http://developer.android.com/about/dashboards/ index.html, April, 2013.

[7] B. Johnsson, P. Ganestam, M. Doggett, and T. Akenine-Möller. Power efficiency for software algorithms running on graphics processors. In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics conference on High-Performance Graphics*, EGGH-HPG'12, pages 67–75, Aire-la-Ville, Switzerland, Switzerland, 2012. Eurographics Association.

[8] Y. Kryachko. *Using Vertex Texture Displacement for Realistic Water Rendering*, pages 269–282. GPU Gems 2. Addison-Wesley, 2005.

[9] N. Kularatna. Rechargeable batteries and their management. *Instrumentation Measurement Magazine, IEEE*, 14(2):20–33, 2011.

[10] A. Mammela, A. Kotelba, M. Höyhtyä, and D. P. Taylor. Relationship of average transmitted and received energies in adaptive transmission. *Vehicular Technology, IEEE Transactions*, 59(3):1257–1268, 2010.

[11] A. Balasubramanian N. Balasubramanian and A. Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 280–293, New York, NY, USA, 2009. ACM.

[12] J. K. Nurminen and M. Heikkinen. Consumer attitudes towards energy consumption of mobile phones and services. In *IEEE 72nd Vehicular Technology Conference (VTC2010-Fall)*, 6-9 September 2010.

[13] J. Pool, A. Lastra, and M. Singh. Precision selection for energy-efficient pixel shaders. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, HPG '11, pages 159–168, New York, NY, USA, 2011. ACM.

[14] T. B. Reddy and D. Linden. *Linden's handbook of batteries, Fourth edition*. McGraw-Hill, 2010.

[15] N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh, and J. Pal Singh. Who killed my battery?: analyzing mobile browser energy consumption. In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 41–50, New York, NY, USA, 2012. ACM.

[16] J. M. Vatjus-Anttila. Opengles2 render engine for energy measurements. http://chiru.fi/?udpview=12357&sid=5&src=db23044, July, 2013.