View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Repositorio Universidad de Zaragoza

DOI: 10.1111/cgf.13483
Eurographics Symposium on Rendering 2018
T. Hachisuka and W. Jakob
(Guest Editors)

Volume 37 (2018), Number 4

On-the-Fly Power-Aware Rendering

Yunjin Zhang[1*] Marta Ortin[2*] Victor Arellano[2] Rui Wang[1†] Diego Gutierrez[2] Hujun Bao[1]

[1] State Key Lab of CAD&CG, Zhejiang University [2] Universidad de Zaragoza, I3A
*Joint first authors †Corresponding author: rwang@cad.zju.edu.cn

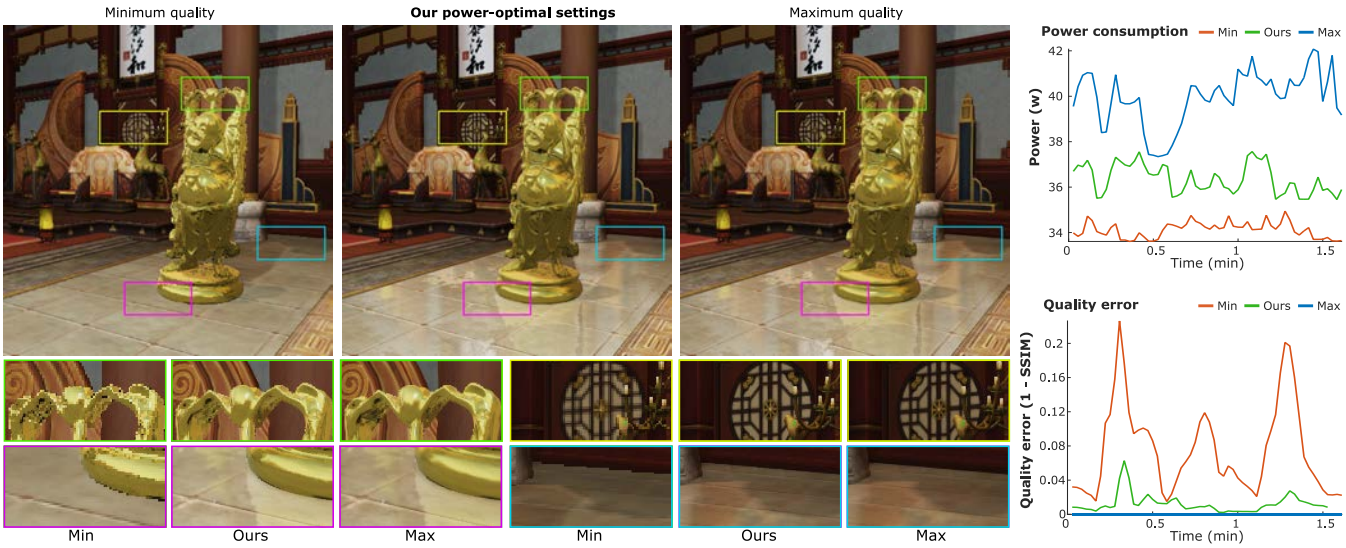

Figure 1: *We propose a novel on-the-fly, power-aware framework that selects the optimal rendering configuration to maximize visual quality, while keeping GPU power consumption within a power budget. Different from existing approaches, our method requires only a few minutes of initialization executed once per platform. The figure shows results for the Hall scene, where our power-optimal settings yield images of similar quality as Maximum Quality, with significantly lower power consumption. The charts on the right show power consumption and quality error (measured with the perceptually-based SSIM metric).*

Abstract

Power saving is a prevailing concern in desktop computers and, especially, in battery-powered devices such as mobile phones. This is generating a growing demand for power-aware graphics applications that can extend battery life, while preserving good quality. In this paper, we address this issue by presenting a real-time power-efficient rendering framework, able to dynamically select the rendering configuration with the best quality within a given power budget. Different from the current state of the art, our method does not require precomputation of the whole camera-view space, nor Pareto curves to explore the vast power-error space; as such, it can also handle dynamic scenes. Our algorithm is based on two key components: our novel power prediction model, and our runtime quality error estimation mechanism. These components allow us to search for the optimal rendering configuration at runtime, being transparent to the user. We demonstrate the performance of our framework on two different platforms: a desktop computer, and a mobile device. In both cases, we produce results close to the maximum quality, while achieving significant power savings.

CCS Concepts

•***Computing methodologies*** → ***Rendering;***

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd. Published by John Wiley & Sons Ltd.

1. Introduction

Current mobile phones and other battery-powered devices incorporate increasingly complex functionalities and applications, which in turn lead to higher power consumptions. Advances in computer graphics have produced highly sophisticated real-time rendering algorithms, which are used in games, data visualization, or virtual reality. To extend the limited battery life, energy saving becomes a primary goal [AMS08, JGDAM12]. A lot of research effort has recently been oriented towards characterising the power consumption of rendering algorithms and finding strategies to control the amount of expended energy [SPP*15, PLS11, APX14, WYM*16].

Wang et al. [WYM*16] proposed a state-of-the-art power-saving framework, which traded power consumption for image quality at rendering time. The system was capable of producing high-quality images, while expending significantly less energy. Unfortunately, the system required a pre-processing step of several days, which had to be performed for every different scene to be rendered. Moreover, as a consequence of such precomputation, it could not handle dynamic scenes, and required many memory accesses to fetch the stored data, hampering performance.

In this paper, we propose a novel real-time, power-saving framework that finds the optimal tradeoff between power consumption and image quality on-the-fly, with only a few minutes of initialization. This is a significantly harder problem, which in turn makes the framework useful for any new game or application without any additional initialization, and enables handling dynamic scenes for the first time. It predicts power consumption and estimates the quality error of different rendering configurations at runtime, and leverages those predictions to adjust the quality level of different shaders, in order to tune the expended energy and keep it within a user-given power budget.

The main challenge for such real-time power-efficient rendering framework is running without affecting user experience. Key to solving this problem are our runtime power prediction and quality error estimation strategies: First, our novel power prediction model (Section 5) allows us to anticipate the power consumption for every rendering configuration, *without having to measure* the actual energy expended. Second, our quality error estimation (Section 6) obtains the error for all configurations *without the need to render them*. These two components yield extremely accurate predictions, which completely remove the need for the time consuming precomputation of the entire camera-view-space, required for every different scene in Wang et al.'s framework [WYM*16].

We show results with an in-house, OpenGL prototype implementation that includes six different shaders, with three different quality levels for each one, which yields 729 different shader combinations. We demonstrate the flexibility of our approach by running it on two different platforms: a desktop PC and a mobile device.

2. Related Work

The reduction of power consumption is a growing concern in many different areas, including both algorithms and hardware architecture [KY14]. Many recent examples have been shown regarding display technology [MWDG13, CWC*14, CCC*16], user interfaces [DCZ09], or cloud photo enhancement [GSC*15], to name a few. This issue is specially relevant in mobile devices with limited battery life [ILMR03]. We focus here on the particular aspects more closely related to our work: energy saving in rendering and GPU power modeling.

Energy saving in rendering. The power efficiency of several existing graphics algorithms has been extensively examined on different GPUs, as a first step towards reducing the power consumption associated to rendering [JGDAM12]. Power limitations are specially relevant in GPUs for mobile devices, and power reduction techniques such as tiling architectures and data compression have been broadly explored [AMS08]. Stravrakis et al. [SPP*15] employ dynamic voltage scaling based on framerate, and implement an energy-aware balancing algorithm that dynamically selects the rendering parameters (geometrical complexity and texture resolution) to save power. Reducing the precision of arithmetic operations can also effectively reduce energy consumption in pixel shaders [PLS11]. With respect to hardware-based optimizations, Arnau et al. [APX14] observe that many fragments are repeatedly rendered in different frames, and exploit this redundancy using fragment memoization.

GPU power modeling. GPU power can be modeled by considering the static and dynamic power of each one of its architectural units (floating point unit, ALU, cache, memory...) [HK10]. Instead, we aim at predicting power consumption using only rendering information, in order to obtain a model directly related to scene complexity. Vatjus-Anttila et al. [VAKH13] proposed a model for GPU power consumption taking into account the contributions of three different primitives separately (batches, triangles, and texels), and combining them as a weighted sum. Different from this approach, our model includes render passes, takes into account all primitives simultaneously, includes the number of fragment shader invocations instead of texels as a better predictor of power consumption, and adapts in real-time to changes in the scene. Besides, Vatjus-Anttila et al. need to include an estimated percentage of backfacing and depth culled primitives in order to improve the accuracy of their model. In contrast, we obtain the precise number of primitives used in each stage of the GPU pipeline and use them directly. This allows us to handle more complex, dynamic scenes, and leads to much higher prediction accuracy.

Recently, Wang et al. developed a power-optimal rendering framework for mobile devices [WYM*16], based on Pareto frontiers in power-error space. Despite the very good results, the method requires several days of precomputation of the whole camera-view space for *each* scene. This is impractical, limits the application of the method to static scenes only, imposes large memory requirements, and forces all novel views produced at runtime to be interpolated, which may lead to large errors even in static scenes with large content changes between views (e.g. unoccluded objects).

In contrast, we introduce a novel real-time power prediction model and an error estimation mechanism, which can handle new games and applications without any specific precomputation. It adapts to the scene being rendered in real-time, thus being able to handle dynamic scenes and effects while providing very high accuracy for scenes with different characteristics and complexity.

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

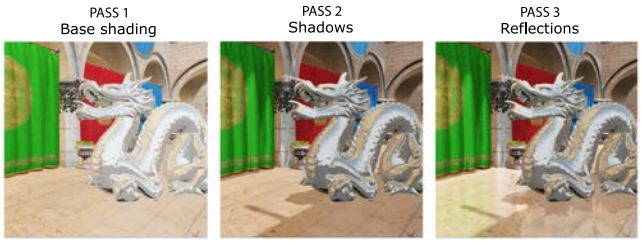

Figure 2: *Example of the rendering process composed of three rendering passes: base shading, shadows, and reflections.*

3. Problem definition

We consider the rendering process as composed of multiple rendering passes that define the visual effects (shadow mapping, reflections, antialiasing...), as illustrated in Figure 2. Each pass is executed with a shader with a particular quality level. The input of the rendering process is thus a rendering configuration $\mathbf{s}$ (a vector describing the sequence of shaders corresponding to each rendering pass), and the camera parameters $\mathbf{c}$ (position and view). In the $\mathbf{s}$ vector, the i^{th} component represents the shader quality level used for the i^{th} pass. The contributions from all the passes are combined by a function f to generate the final image; generalizing the rendering process as a function f allows us to implicitly include forward and deferred rendering in our framework. Table 1 sums up all the symbols used the paper.

Different rendering configurations yield results of varying visual quality. Let $\mathbf{s_{best}}$ denote the rendering settings that generate the best quality image. Similar to the recent work of Wang et al. [WYM*16], we can define the quality error $e(\mathbf{s},\mathbf{c})$ of any image produced by different rendering settings as

$$e(\mathbf{s},\mathbf{c}) = \int\int_{xy} ||f(\mathbf{s_{best}},\mathbf{c}) - f(\mathbf{s},\mathbf{c})||dxy \qquad (1)$$

where x, y define the pixel domain of the image, and $||\cdot||$ indicates the chosen norm.

Besides yielding varying quality errors, different $\mathbf{s}$ and $\mathbf{c}$ vectors also result in different power consumption $P(\mathbf{s},\mathbf{c})$. In general, higher quality images require more power, which generates a trade-off between power and error. Therefore, given a power budget P_{bgt}, we look for a vector $\mathbf{s}$ such that $e(\mathbf{s},\mathbf{c})$ is minimized, while $P(\mathbf{s},\mathbf{c})$ remains within the budget:

$$\mathbf{s} = \arg\min_{\mathbf{s}} e(\mathbf{s},\mathbf{c}) \qquad \text{subject to} \qquad P(\mathbf{s},\mathbf{c}) < P_{\text{bgt}} \qquad (2)$$

Different from Wang's work, we demonstrate in this paper how to predict $P(\mathbf{s},\mathbf{c})$ and estimate $e(\mathbf{s},\mathbf{c})$ *in real-time*. This is a significantly more difficult problem, since Wang's framework relied on a time-consuming precomputation (in the order of a few days) of the entire camera-view-space, to be performed for each particular game or scenario. Our implementation includes six different shaders (resolution, base shading, reflections, shadows, metals, and antialiasing), with three quality levels each, generating a total of 729 different rendering configurations.

$\mathbf{s}$	Rendering configuration: vector with shader quality level for each pass
s_i	Shader for pass i
$\mathbf{s_{best}}$	Rendering configuration that generates best quality images
$\mathbf{c}$	Camera position and view
$f(\mathbf{s},\mathbf{c})$	Image rendering function with $\mathbf{s}$ and $\mathbf{c}$.
$e(\mathbf{s},\mathbf{c})$	Image quality error with $\mathbf{s}$ and $\mathbf{c}$, simplified as $e(\mathbf{s})$.
$P(\mathbf{s},\mathbf{c})$	Rendering power with $\mathbf{s}$ and $\mathbf{c}$, simplified as $P(\mathbf{s})$.
P_{bgt}	Power budget
P_m	Minimum power consumption of the GPU
P_M	Maximum power consumption of the GPU
b, v, f	Batches, vertices, and fragments used to render a frame
B, V, F	Batches, vertices, and fragments that saturate the GPU
k_b, k_v, k_f	Coefficients for batches, vertices, and fragments
Ins_{vi}, Ins_{fi}	Instructions in vertex and fragment shaders for pass i
Tex_{vi}, Tex_{fi}	Texel accesses in vertex and fragment shaders for pass i
χ	Cost asociated to the execution of one instruction
ψ	Cost asociated to the one texel access
l_j	Quality level j used for a given pass
l_{max}	Worst quality level for a given pass
$\mathbf{s^0}$	Rendering configuration where every pass uses shader quality level 0, same as s_{best}
$\mathbf{s_i^l}$	Configuration where every pass uses shader quality level 0 except for pass i, which uses level $l (l > 0)$
$\mathbf{s_i^{l_{max}}}$	Configuration where every pass uses shader quality level 0 except for pass i, which uses the worst quality level
k	Coefficient that relates the quality error of two shaders for the same pass

Table 1: *Symbols used throughout the paper, and their definition.*

4. Algorithm Overview

Our algorithm is based on two key components, depicted on Figure 3: a power prediction model, and a quality error estimation mechanism. Our power prediction model fits a set of coefficients in an equation describing scene complexity, requires minimal initialization, and adapts in real-time to the content being displayed. In order to select the optimal rendering configuration within a power budget, we introduce a strategy to reuse fitted coefficients to predict power consumption in new configurations with minimal computation.

For our quality error estimation, we first compute the error of a frame with several rendering configurations (one per pass, six in total in our prototype implementation) by running the renders in the background and calculating the SSIM perceptual quality metric. We then use the obtained values to estimate the error for all the other configurations (729 in our case).

Our on-the-fly power-efficient rendering framework makes use of these two components to produce a final image with the highest possible quality, within a given power budget. Sections 5 and 6 explain the details of our power prediction model and error estimation mechanism, respectively, while Section 7 describes how the power prediction and error estimation steps are combined at runtime to obtain the optimal rendering configuration.

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

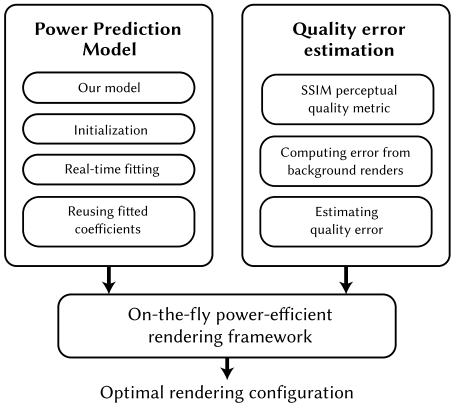

Figure 3: *Main components of our algorithm: power prediction model, and quality error estimation. Both components are combined in our on-the-fly, power-efficient rendering framework to generate the optimal rendering configuration.*

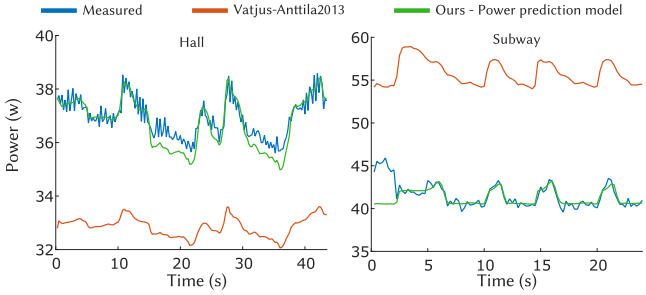

Figure 4: *Power consumption for the Hall and Subway scenes. Our power prediction closely matches the ground truth, measured power consumption. For comparison, we also show the prediction using the model proposed by Vatjus-Anttila et al. [VAKH13].*

5. Power Prediction Model

We introduce our power prediction model based on scene complexity, describe the initialization and real-time fitting process, show how we can predict the power for all our rendering configurations by fitting our model for only one of them, and describe implementation details.

5.1. Our model

The rendering pipeline starts running when we command the GPU to draw a group of triangles (called *a batch*) already uploaded to the GPU memory. Those triangles go through several consecutive stages: first, the vertex shader executes per-vertex processing operations; then, rasterization runs per-primitive processing to cull hidden primitives; finally, the fragment shader interpolates per-fragment parameters, texturing, and coloring to generate the final pixel color [Hen]. When setting a fixed frame rate, the complexity of the scene determines the load imposed on the GPU, and power savings are achieved when the GPU is idle between consecutive frames. Our power prediction model takes into account consumption at the different stages of the GPU pipeline, according to scene complexity; it includes the number of batches b, vertex shader calls v, and fragment shader calls f (in the rest of the paper we will refer to these variables as *primitives*).

Similar to previous work [VAKH13], we observe that GPU power consumption follows an inverted exponential function between a minimum P_m and a maximum P_M power, as the rendering load increases. Given a rendering configuration $\mathbf{s}$ and camera parameters $\mathbf{c}$, we thus propose the following power consumption model:

$$P(\mathbf{s},\mathbf{c}) = P_m + (P_M - P_m)(1 - exp^{-\alpha}) \qquad (3)$$
$$\alpha = k_b \frac{b}{B} + k_v \frac{v}{V} + k_f \frac{f}{F}$$

Each one of the b, v, and f primitives is normalized by the number of elements that causes the GPU to saturate to its maximum capacity (B, V, and F, respectively). They are additionally weighted by coefficients k_b, k_v, and k_f, to take into account the relative impact of each one on the total power consumption. All the parameters depend on the rendering configuration $\mathbf{s}$; additionally, the camera parameters $\mathbf{c}$ are implicitly included in the primitives b, v, and f; we omit these explicit dependencies in the rest of the paper for the sake of clarity.

Since the complete rendering process is composed of several passes, we extend our previous power model to represent each pass individually:

$$P = P_m + (P_M - P_m)(1 - exp^{-\Sigma_i^N \alpha_i}) \qquad (4)$$
$$\alpha_i = k_{bi} \frac{b_i}{B_i} + k_{vi} \frac{v_i}{V_i} + k_{fi} \frac{f_i}{F_i}$$

where N is the number of passes, and the subindex i for each variable indicates its per-pass value. Figure 4 depicts results for two example scenes (*Hall* and *Subway*) with ground truth measured power, showing how our equation yields a good prediction of power consumption: an average of only 1% error in *Hall* and 2% in *Subway*. The model proposed by Vatjus-Anttila et al. provides worse power predictions (11% error in *Hall* and 37% error in *Subway*) because it does not model the contribution of each rendering pass, it does not consider the number of fragment shader invocations, it uses an estimated correction factor to compute the number of primitives, and it does not adapt to the scene being rendered in real-time. More examples can be found in the supplementary material.

5.2. Initialization

Given our power model, we first obtain P_m by sending an empty scene to the GPU; we then progressively increase the complexity of the scene to find the values of P_M, B_i, V_i, and F_i that saturate the GPU. This is an offline process that needs to be performed only once per hardware platform, and requires only 3 minutes, in contrast with the costly precomputation of the camera-view-space required for every scene in Wang et al.'s proposal [WYM*16]. In addition, we also obtain the number of instructions and texel accesses for the rendering passes with each quality level, which will be explained in Section 5.4.

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

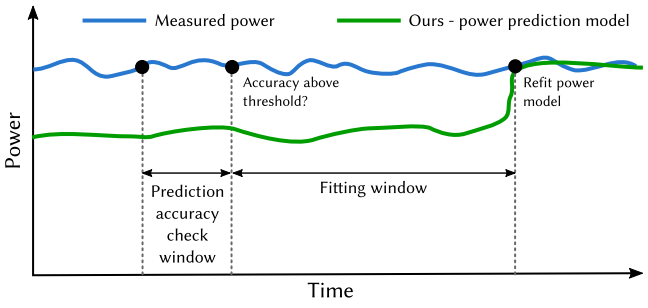

Figure 5: *Timeline showing how refitting works in our model. First, rendering samples are collected during a prediction accuracy check window. We then check if the accuracy of the predicted power is above a given threshold. If the predicted power needs to be updated, we collect new samples during a fitting window, which are used to refit the power model and yield a new prediction.*

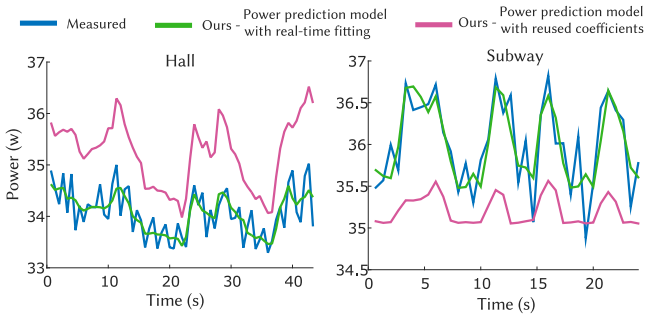

Figure 6: *Power consumption of the Hall and Subway scenes, for rendering configuration* $\mathbf{s_A} = (l_0, l_2, l_1, l_1, l_2, l_2)$. *We show ground truth measured data, predicted power with our real-time fitting, and predicted power with coefficients reused from the fitting of a different configuration* $\mathbf{s_B} = (l_1, l_1, l_1, l_1, l_1, l_1)$.

5.3. Real-Time fitting

A key aspect of our method is our *real-time power fitting* process, which allows us obtain very high prediction accuracy without imposing a penalty in performance. When the scene starts running, we collect rendering samples[†] during a *fitting window*, and use them to fit coefficients k_{bi}, k_{vi}, and k_{fi} using a linear regression on the power consumption and number of primitives (see Equation 4).

After the fitting takes place, we check periodically during runtime if the process has to be triggered again, to improve the accuracy of the prediction due to scene changes: First, we collect rendering samples during a *prediction accuracy check window*, and compare our predicted power consumption with the actual consumption measured in the GPU. If the average difference (computed during the frames of the prediction accuracy check window) is above a set threshold, we trigger the real-time fitting process again, and update the coefficients of our power model. This process is illustrated in Figure 5.

5.4. Reusing the fitted coefficients for other configurations

At any given moment, we are rendering the scene and fitting our power model with a single rendering configuration. However, each configuration leads to a different power consumption. Therefore, in order to find the optimal rendering configuration, we need to predict the power consumption for *every one* of them (729 in our prototype implementation); fitting the model for every configuration would obviously be impractical, and too computationally expensive.

To solve this problem, we leverage what our coefficients represent: k_{bi}, k_{vi}, and k_{fi} express the cost associated to batches, vertices, and fragments, respectively. In particular, k_{bi} is the cost of a rendering request and all the exchange of information between CPU and GPU required to perform the rendering task. In general, this cost is fixed for all the shader quality levels of a given pass, so we can reuse the same k_{bi} for all our rendering configurations. On the other hand, k_{vi}, and k_{fi} are related to the number of executed instructions and texel accesses [‡], so we can express the coefficients associated to pass i with shader quality level s_i as:

$$k_{vi}(s_i) = \chi Ins_{vi}(s_i) + \psi Tex_{vi}(s_i) \quad (5)$$

$$k_{fi}(s_i) = \chi Ins_{fi}(s_i) + \psi Tex_{fi}(s_i) \quad (6)$$

where $Ins_{vi}(s_i)$, and $Ins_{fi}(s_i)$ represent the average number of executed instructions for vertices and fragments in pass i, with shader quality level s_i; $Tex_{vi}(s_i)$ and $Tex_{fi}(s_i)$ are the average number of texel accesses for vertices and fragments in pass i with shader quality level s_i; χ and ψ are the costs associated to an instruction and a texel access. Since vertex shaders usually have the same number of instructions per triangle for each pass regardless of the quality level, and they do not access texels, we can simplify Equation 5 as $k_{vi} = \chi Ins_{vi}$. We obtain Ins_{vi}, $Ins_{fi}(s_i)$, and $Tex_{fi}(s_i)$ during the initialization step, which is performed only once per platform (Section 5.2). We instrument the shaders while they are being loaded into the GPU to include atomic counters that automatically count the number of executed instructions and texel accesses for each primitive, and compute the average after running a dummy scene for a few minutes[§]. Therefore, the only unknowns are χ and ψ, which we obtain by solving the inconsistent overdetermined system of equations with a linear regression.

This strategy has one key advantage: By fitting only one rendering configuration, we obtain the coefficients k_{bi}, χ, and ψ, which do not depend on that particular rendering configuration. We can then

[†] A sample includes the measured power for a frame, and its corresponding number of batches, vertices, and fragments for each rendering pass.

[‡] Even though any memory fetch could be issued from vertex and fragment shaders, we use the term *texel access* because they generally constitute the vast majory of memory fetches.

[§] Any scene can be used to obtain the necessary information, as long as all the shaders in the rendering engine are executed. At 30 fps, running the scene for a couple of minutes allows us to obtain stable, averaged results.

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

reuse them together with Equations 4, 5 and 6, to obtain power predictions for all our configurations. Thus, the power consumptions associated to different rendering configurations depend only on the number of executed instructions, texel accesses, and primitives. Figure 6 shows our resulting prediction reusing coefficients from a different configuration.

5.5. Implementation details

Our prediction accuracy check window has a length of 10 frames, and our refitting window lasts 30 frames. These values are selected to enable a fast fitting process while collecting enough data to ensure the robustness of our fitted model. The complete fitting process and reuse of coefficients is completed in less than 1.5 s after the prediction accuracy check is launched. We set the accuracy threshold to 10% of the difference between P_m and P_M. The linear regression to fit our power model takes an average of 2.6 ms, and the computations to reuse the coefficients for other configurations require 0.7 ms. To ensure that they do not interfere with the rendering process, we execute them in a separate thread on the CPU, while the GPU continues rendering the scene.

6. Quality Error Estimation

To select the optimal rendering configuration, we need to assess the quality error of any frame, by comparing it with its corresponding reference frame φ_r, rendered with the highest quality. Similar to Wang's budget rendering framework [WYM*16], we use the perceptually-based Structural Similarity Index (SSIM) [WBSS04], with error given by $e = 1 - SSIM$.

Let φ be the current frame for which we want to obtain the error, and let φ_s represent all other alternative renderings using all other rendering configurations. In Wang's previous offline approach, every high-quality reference frame φ_r, all their alternative renderings φ_s, and their associated quality errors had been precomputed in advance, based on a dense partitioning of the camera-view space of the scene. We face a much harder problem, since we aim to perform all necessary computations at runtime, on a dynamic scene. This involves, apart from obtaining the reference frame φ_r, rendering frames φ_s with all other rendering configurations, and calculating their associated quality error. In the rest of the section, we first describe how error is *computed*; however, given the large space of all rendering configurations, it is impossible to compute the error for all of them without visibly affecting performance. We thus introduce our approach to accurately *estimate* most errors, without the need to explicitly compute them.

6.1. Computing quality error

Since error computation should not interfere with the user experience, φ_r and all φ_s are rendered in the background, to a secondary frame buffer (not shown on the screen); φ_r is saved in a texture, while each φ_s is rewritten in successive renderings after its associated error has been calculated. To avoid a visible drop in the frame rate from rendering φ_r and all φ_s consecutively, we distribute the task over time. We save the rendering settings used to obtain φ, as well as the positions of moving objects[¶], and restore them with an *error computation frequency* to render one frame in the background.

Distributing the rendering tasks over time avoids a sudden drop in performance, but in turn it makes the process excessively long for all the different rendering configurations. To overcome this, the quality error can be computed for just a small subset of rendering configurations. Since this step takes place after power prediction, such configuration subset can be selected from the configurations with higher power below the threshold, which are more likely to produce high quality images. The selection of the optimal rendering configuration would then choose the best configuration among the available ones. Alternatively, we propose an approximation to obtain *estimated* quality error values for *all* the configurations, without the need to compute all of them. This approximation is suitable for our application, since it allows us to obtain relative estimations to compare different configurations.

6.2. Estimating quality error for all rendering configurations

We make two important observations that allow us to *estimate* the error for all 729 rendering configurations by rendering and computing the error for *only six* of them (one per rendering pass).

For the following discussion, we define $\mathbf{s^0}$ as the rendering configuration where every pass uses the highest shader quality (level 0, l_0), and $\mathbf{s_i^l}$ as the configuration where every pass uses l_0 except for pass i, which uses level $l (l > 0)$. Our two observations are:

- First, we can approximate the quality error for a rendering configuration by adding up the error introduced by each individual pass. This means that the total error for any rendering configuration can be expressed as a sum of errors using only $\mathbf{s_i^l}$ rendering configurations. For example, with three rendering passes, the error for rendering configuration $\mathbf{s} = (l_2, l_0, l_1)$ can be obtained as:
$$e(\mathbf{s} = (l_2, l_0, l_1)) = e(\mathbf{s_0^2}) + e(\mathbf{s_2^1}) \tag{7}$$
- Second, given two rendering configurations, $\mathbf{s_i^{l_1}}$ and $\mathbf{s_i^{l_2}}$, with best quality shaders except for one pass i using shaders l_1 and l_2, their associated quality errors follow:
$$e(\mathbf{s_i^{l_1}}) = k e(\mathbf{s_i^{l_2}}) \tag{8}$$
The set of all coefficients k depends only on the rendering engine used, not on the particular scene being rendered, and thus can be computed beforehand (together with the initialization of our power model).

Combining these two observations, we can estimate the quality error for all our configurations by computing only the error for all $s_i^{l_{max}}$, that is, one configuration per pass. Figure 7 shows the accuracy of the estimated error using these simplifications. Note that all error computations and estimations are performed in real-time; only the k coefficients have to be obtained beforehand.

[¶] In our implementation, we identify moving objects by the presence of animated skeletal meshes.

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

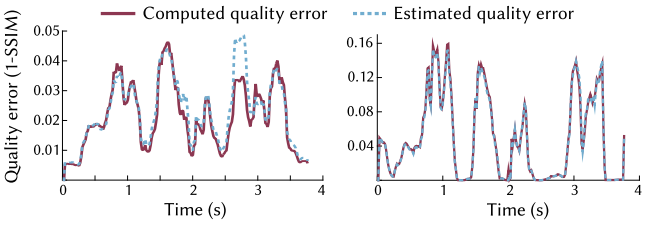

Figure 7: *Computed and estimated quality error for the Subway scene, using our observations in 6.2. **Left:** Quality error for rendering configuration* $\mathbf{s} = (l_0, l_2, l_2, l_1, l_0, l_0)$*, approximated by adding up the error from each individual pass (Equation 7). **Right:** Quality error of one rendering pass with medium quality (configuration* $\mathbf{s} = (l_0, l_0, l_0, l_1, l_0, l_0)$*), using the observation in Equation 8 and coefficient k obtained from the Sponza and Valley scenes. Please refer to the digital version to distinguish the overlapping computed and estimated quality errors.*

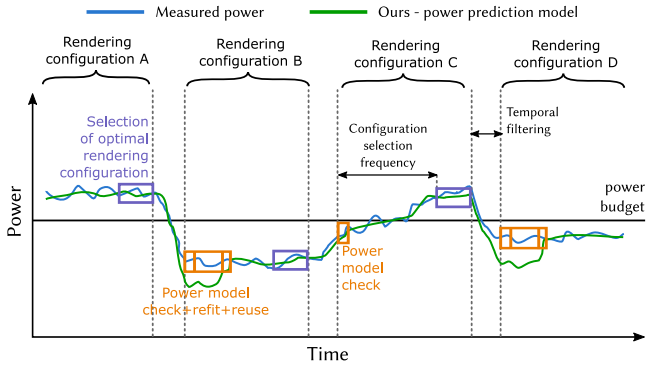

Figure 8: *Timeline illustrating power consumption during rendering (measured and predicted with our model), and how our algorithm is executed. Given a configuration selection frequency, a new optimal rendering configuration is selected (purple box). Our temporal filtering is executed to transition to the new configuration. Immediately after that, the power accuracy check is performed, followed if necessary by the real-time fitting, and the reuse of fitted coefficients (orange boxes). In this example, when switching to rendering configuration B, the power check step detects an above-threshold gap between the measured and the predicted power, so fitting and reuse are activated. However, when switching to rendering configuration C, the power check confirms that the gap is below threshold, and refitting and reuse are not launched. Please refer to the text for more details.*

6.3. Implementation details

We compute the error for a frame φ_s once very 10 frames. This error computation frequency was selected to minimize the length of the error computation and estimation process while guaranteeing that the GPU is able to keep up with the target frame rate. Alternatively, the error computation frequency can be adjusted at runtime based on the current and target frame rates.

Obtaining the SSIM index is computationally expensive, taking an average of 0.05 s. Therefore, after a frame φ_s has been rendered in the background, the quality error with SSIM is computed in parallel on a separate thread, while the GPU continues rendering the game. The GPU-CPU communication takes 0.02 seconds in the worst case, which corresponds to the exchange of data to compute the SSIM index for 2048x2048 resolution.

7. On-the-fly Power-Efficient Rendering

In the previous sections we have described our power prediction model and quality error estimation mechanism. We now show how those components are combined at runtime to select the optimal rendering configuration. Our periodic selection for the optimal configuration is followed by a temporal filtering to gradually transition to the new configuration, as illustrated in Figure 8. When the new configuration is set, we start the real-time fitting of our power model.

7.1. Selection of the optimal rendering configuration

Given our power predictions and quality error estimations, we aim to find the optimal rendering configuration for a given scene and camera parameters, minimizing quality error while meeting our power budget, as formulated in Equation 2. This selection process is triggered periodically, with a *configuration selection frequency*.

We first use our power prediction model to obtain the power consumption for the current frame with all possible rendering configurations. Wang et al. [WYM*16] precompute the power and error for all configurations, producing a large two-dimensional power-error space. To simplify their runtime search for the optimal configuration, they also precompute the Pareto frontier to reduce their two-dimensional exploration of the power-error space to a one-dimensional search along the Pareto frontier. Instead, we predict the power consumption and estimate the error at runtime. This is difficult, as argued in the paper, but in turn it offers an additional advantage: since the power budget is known in advance, when we predict power consumption we can discard all the configurations with a power consumption higher than our budget. The costly two-dimensional search in power-error space is then reduced to a one-dimensional search in error space; this means that we can completely eliminate the need to compute the Pareto frontier.

For the configurations that meet our power budget, we estimate the quality error following the process described in Section 6: We render φ in the background with configurations s^0 and $s_i^{l_{max}}$, and compute their error (according to the error computation frequency to ensure a constant frame rate), and use Equations 7 and 8 to estimate the quality error for the rest of the configurations. Finally, since we already discarded all the configurations above the power budget, we simply need to choose the rendering configuration with the lowest quality error. This process corresponds to the purple box in Figure 8, and is illustrated by Figure 9, which shows how our rendering configurations are distributed in power-error space, and how our strategy is effective in selecting the one with lowest error within the power budget.

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

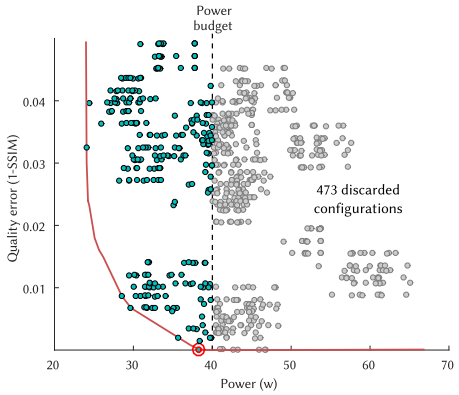

Figure 9: *Our rendering configurations drawn in power-error space. We discard all the configurations over the power budget, and perform just a one-dimensional search in error space on the remaining configurations, selecting the one with lowest error (marked with a red circle). This eliminates the need to compute the Pareto frontier in our framework (shown in red for reference only).*

Since we do not rely on scene-specific precomputed data, and different scenes may have very different power requirements, which are thus not known in advance, setting the power budget as an absolute predefined value (as in Wang et al.'s proposal [WYM*16]) is not practical. Therefore, we define the power budget as a percentage between the minimum and maximum power consumption of the scene, which is a very intuitive value to represent the trade-off between power consumption and image quality. For example, if our power budget is 40%, only configurations with predicted power lower than $P_m + 0.4(P_M - P_m)$ will be eligible.

7.2. Temporal filtering

To avoid a sudden change in image quality when a new rendering configuration is selected, the transition to the new configuration is performed smoothly with the temporal filtering introduced by Wang et al. [WYM*16]. During an interpolation interval T, while the framework transitions from $\mathbf{s_{old}}$ to $\mathbf{s_{new}}$, the effective rendering configuration used for rendering $\mathbf{s_{eff}}$ is computed as:

$$\mathbf{s_{eff}} = [(1 - \frac{t}{T})\mathbf{s_{old}} + \frac{t}{T}\mathbf{s_{new}}] \quad (9)$$

where the brackets denote the closest integer and t is the time after starting the transition to the new configuration.

7.3. Real-time fitting of the power model and reusing coefficients

Every time a new rendering configuration is set, our power prediction accuracy check is triggered (small orange box after temporal filtering in Figure 8). If the accuracy of our prediction is below a threshold, we refit our power model, as explained in Section 5.3. This happens twice in Figure 8, and is represented by larger orange boxes. The newly fitted coefficients are then used to update the power model for all other configurations by obtaining the cost associated to each instruction and texel access (Section 5.4 and third small orange box in Figure 8).

7.4. Implementation details

The configuration selection frequency triggers the process to select a new optimal rendering configuration 200 frames after the previous configuration was set. This frequency allows us to quickly detect changes in the scene while minimizing the impact of the associated computations. Refitting the power model and reusing the coefficients for other configurations (3.4 ms), predicting the power for all configurations (0.02 ms), and estimating the error and selecting the optimal configuration (0.03 ms) are executed on separate threads. The temporal filtering interval used for interpolation is 2 seconds.

8. Implementation

To show how our on-the-fly power-budget framework adapts to different hardware, we have implemented it on two different platforms: A desktop PC with an Intel Core i7-7700 and an NVIDIA Quadro P4000, and a mobile Qualcomm Snapdragon 660 (with a 8x Kryo 260 CPU and an Adreno 512 GPU).

8.1. Power Measurement

To measure the power usage of the graphics card in the desktop PC, we use the NVIDIA Management Library (NVML) [NVM15], which allows us to directly access the power usage of the GPU and its associated circuitry. The specifications report an accuracy of 5%. In our mobile device, we use an external source meter to directly supply the power of the device. We use a Keithley A2230-30-1, which provides APIs to access the instantaneous voltage and current (same setup used by Wang et al. [WYM*16]). During the stages of our algorithm when we have to collect rendering samples (for the power prediction accuracy check and real-time refitting), we measure the power consumption for every frame. In order to reduce variance, in our graphs we report the average power measured over 30 frames.

8.2. Rendering Configurations

Our rendering framework runs at 30 frames per second in the desktop PC and at 10 frames per second in the mobile device, and has six passes, each one with shaders of three different quality levels; this amounts to a total of 729 different rendering configurations. The complete set of parameters and values of these shaders is given in Table 2. In particular, we have included:

Resolution: When setting the resolution of a frame, the number of fragments for other passes are proportionally scaled.[‖]

[‖] The resolution is technically not a pass, it sets the screen resolution, which affects other passes. However, it is included in the list of passes for convenience, because it has an effect on power consumption and quality error, and has to be considered as an additional degree of freedom when selecting the optimal rendering configuration.

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

Passes	Parameters	Values
Resolution	buffer resolution	60%, 80%, 100%
Base shading	specular reflections	cheap spec., improved point lights, microfacet spec.
Reflections	(samples, kernel)	off, (16,1), (64,9)
Shadows	map resolution	512, 1024, 2048
Metals	samples	2, 6, 60
Antialiasing	steps	off, 2, 32

Table 2: *List of parameters and values forming the space of rendering settings.*

Demos	Scene Statistics			Rendering
	Triang.	Objects	Scene Size	Duration
Hall	229.4 k	23	22.33 MB	1.6 min
Sponza	262.1 k	381	25.4 MB	2.7 min
Valley	143.3 k	61	17.0 MB	1.5 min
Subway	526.6 k	453	77.5 MB	2 min

Table 3: *Statistics for our four demo scenes, including number of triangles, number of objects, size on disk of each scene, and duration of the demo.*

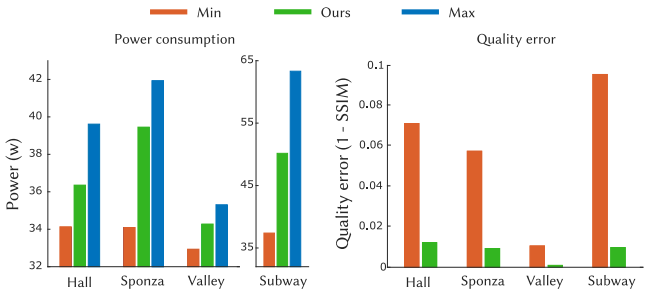

Figure 10: *Average power consumption per frame and quality error in our four demos. Note that the quality error for maximum quality is zero.*

Base Shading: The simplest level is a cheap specular shader, which is improved with a better model for point lights in the next level. The best quality level implements microfacet-based shading.

Reflections: For objects with specular materials; it is a multi-pass shader where quality levels increase the number of generated secondary rays, and the kernel size for color filtering [Sta15].

Shadows: The quality level is given by the resolution of the shadow map.

Metals: It is an importance sampling algorithm where quality levels are defined by the number of samples.

Antialiasing: We rely on the FXAA morphological antialiasing to detect edges in the pixel shader [Lot09, JGY*11].

When using Equation 4, we consider the following: i) The resolution pass has no associated primitives, only having an effect on the number of primitives used in other passes. Therefore, we do not include that pass specifically in the power model formula. And ii) The antialiasing pass works on the final image, and thus does not depend on the number of batches and vertices, it is only affected by the number of fragments.

9. Results and Evaluation

We have tested our power-efficient rendering framework on two different platforms (a desktop PC and a mobile device), with four scenes of different complexity, to verify its efficiency in a wide range of scenarios; refer to Table 3 for a summary of their main characteristics. In every case, we are able to maintain the predefined 30 frames per second (10 fps in the mobile device). Our framework supports free exploration of the scene, but we use predefined camera paths to facilitate comparisons and measurements with different qualities, and show the potential of our framework in the long run. For each demo, we specify the preset power budget used to guide our optimal configuration selection process.

Figure 10 shows the average power consumption and average quality error of the four scenes, with maximum and minimum quality, and using our framework with the power budgets reported in this section. It can be seen how we significantly reduce power consumption, while keeping visual quality very close to the maximum.

In the following, we show images from our four scenes with the maximum and minimum quality rendering configurations, together with the result of our power-aware framework. Zoomed-in insets allow to better appreciate details, showing how our results are close to the maximum quality, at a reduced power cost (shown in the accompanying plots). In addition, the supplemental video shows the full demo, including split-screen comparisons.

Hall: This scene has a spotlight acting as a lamp and is composed of diffuse objects, except for the reflective floor and two metallic buddha statues. It has a high polygon count but very few objects, thus being useful to test scenarios with a small number of batches. Results for our framework running under a power budget of 40% are shown in Figure 1.

Sponza: We use one directional light as the Sun, and one spot light as a candle, both casting shadows rendered by our shadow pass. There are two metallic lion head ornaments (rendered with our metal pass), and the rest of the scene is diffuse (rendered with our base shading pass). The floor of the scene is slightly reflective. Figure 11, top, shows the results for a power budget of 60%.

Valley: This relatively low-poly scene is illuminated using the Sun as a directional light, without any spotlights. There are no reflective or metallic objects, hence demonstrating the effectiveness of our model on scenarios where some passes do not affect quality error. This also leads to a smaller difference between maximum and minimum power consumption; although this challenging scene limits the range for improvement, our framework still manages to save considerable energy with minimal image degradation when setting the power budget to 40% (Figure 11, middle).

Subway: This complex, high-poly scene is used to test our method in high power usage scenarios. The scene is located underground, so it has no Sun. All the lighting comes from a spotlight located in a lightbulb. The floor of the scene is reflective. The two fighting soldiers are metallic, while the remaining objects are dif-

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

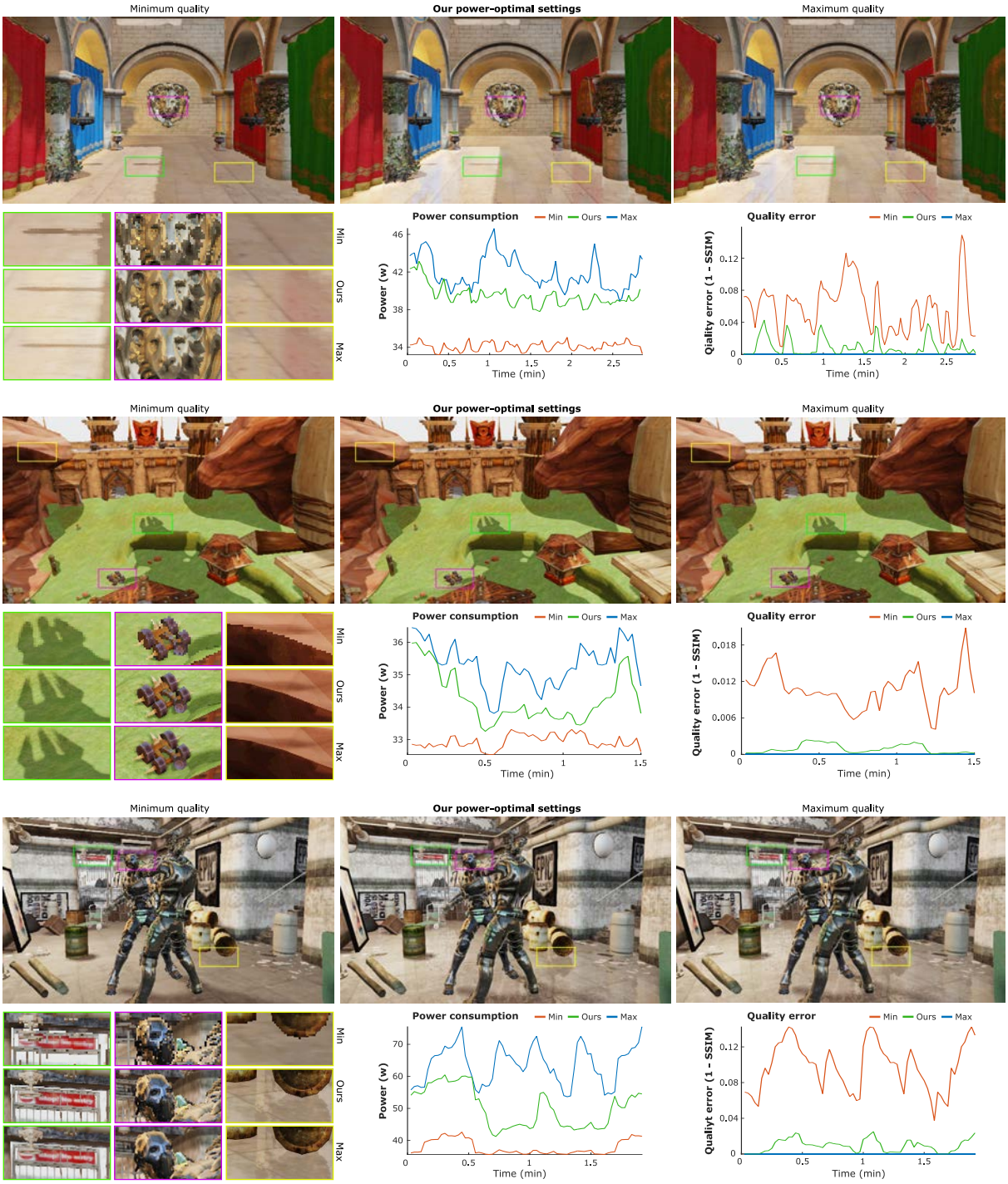

Figure 11: *Sponza (top), Valley (middle) and Subway (bottom) demo scenes, executed on a desktop PC. We compare the minimum and maximum quality rendering configurations against our power-optimal configuration. For Sponza, we use a power budget of 60% (which corresponds to a percentage of the difference between the Min and Max power consumptions); for Valley we use 40%, and for Subway we use 50%. Our method generates images very similar to those rendered with the maximum quality configuration, while keeping power consumption lower. Please refer to the supplementary video for the full demos.*

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

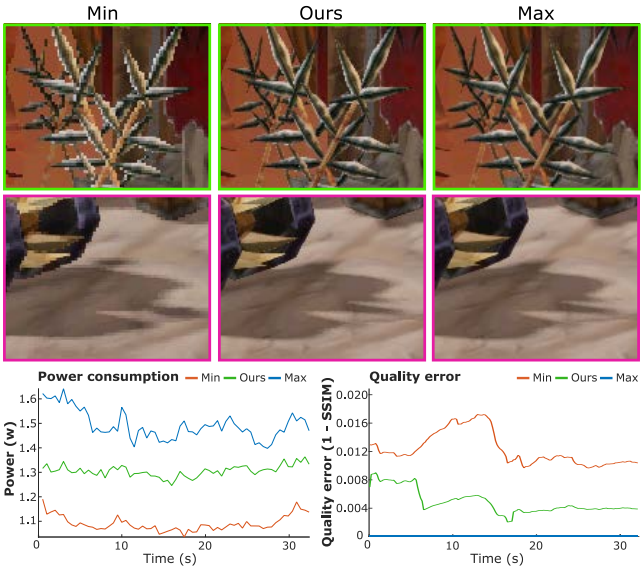

Figure 12: *Valley demo scene executed on our mobile device. We compare the minimum and maximum quality rendering configurations against our power-optimal configuration with a 50% power budget. Our method generates images very similar to those rendered with the maximum quality configuration, while keeping power consumption lower. Please refer to the supplementary video for the full demo.*

fuse. The soldiers are animated using skeletal meshes, allowing us to test our framework on a dynamic scene. Results with a power budget of 50% are shown in Figure 11, bottom.

Additionally, in the supplementary material we show the results for different power budgets applied to the *Sponza* scene.

To demonstrate the efficiency of our framework on mobile devices, we show additional results for the *Valley* scene running in our mobile phone, with a power budget of 50% (Figure 12). We are again able to keep power consumption within our budget with image quality very close to the maximum quality.

10. Discussion

Our on-the-fly power-aware rendering framework successfully addresses the two key limitations of previous work: it does not require any precomputation, and it can handle dynamic scenes. We have shown results for four different scenes of different characteristics, demonstrating large power savings while maintaining image quality close to maximum quality. Analysing the optimal configurations chosen by our framework, we notice that image resolution is rarely lowered, since it leads to high quality errors. Our algorithm does not lead to any degradation of the frame rate, as we demonstrate in our supplemental video.

Our power prediction model may have other applications beyond budget rendering. For example, by detecting an increase in power consumption (which indicates higher rendering complexity),

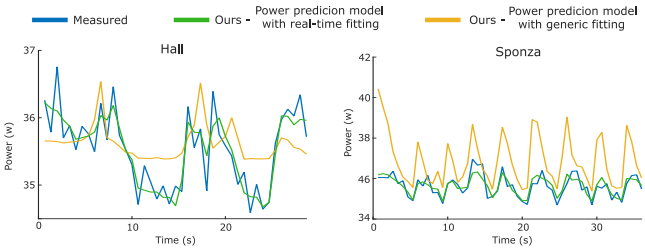

Figure 13: *Power cosumption of the Hall and Sponza scenes with ground truth measured data, predicted with our power model with real-time fitting and predicted with our power model with generic fitting.*

it could analyse different options to avoid frame-rate drops in video games before they happen. It could also be applied to estimate the total energy consumption of a new rendering task, given a limited set of initial data. Since b, t, and f in Equation 4 can be fetched with native OpenGL queries, incorporating our power prediction model to any project is straightforward, and requires no modifications of any existing shader code.

As we have shown, real-time fitting of our power prediction model provides very high accuracy. However, it is also possible to fit the model with a generic dataset to obtain valid coefficients before running any specific scene. To do that, we collect rendering samples from dummy scenes with a varying number of batches, vertices, and fragments, covering the whole parameter space from P_m to P_M. With these data, we fit k_{bi}, k_{vi}, and k_{fi} in Equation 4 using linear regression on the power consumption and number of primitives. This offline process takes around 4 minutes for one rendering configuration, and provides a reasonable approximation of the actual power consumption (see yellow curve in Figure 13).

To perform our runtime quality error computations, we have set a frequency of 10 frames, which we have selected to be as small as possible without minimizing the impact of background rendering on the frame rate. Alternatively, this frequency could be automatically adjusted during runtime according to the current frame, to guarantee a given target frame rate.

Throughout the whole paper, we have defined the power optimal configuration as the one with lowest quality error that meets our power budget (Equation 2). Similar to Wang's work [WYM*16], solving the analogous problem of obtaining the rendering configuration with the lowest error consumption within an error budget is straightforward:

$$\mathbf{s} = \arg\min_{\mathbf{s}} P(\mathbf{s},\mathbf{c}) \qquad \text{subject to} \qquad e(\mathbf{s},\mathbf{c}) < e_{\text{bgt}} \tag{10}$$

In this case, we would start our selection of the optimal configuration by estimating the error for all configurations and discarding the ones above the budget, then predicting power for the remaining configurations, and choosing the one with lowest consumption.

Limitations and future work: Our framework still has some limitations that could be addressed in future work. Our power prediction model seamlessly supports dynamic scenes, by using the in-

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

formation of the current frame to predict power consumption. However, our error computation mechanism needs to explicitly store the positions of moving objects and restore them for background rendering. For scenes with a large number of moving objects, this could become too computationally expensive.

Our power model is based on the typical rendering pipeline with basic processing of batches, vertices, and fragments. It does not accurately model other GPU stages that could be integrated into the pipeline, such as geometry shaders or tesselation, which should be included as additional contributors to our formula. Apart from that, our model is already able to seamlessly represent the additional fragments generated by a geometry shader.

We have demonstrated the viability of our framework using a reasonable number of different shaders, under the strict constraint of real-time execution. We have not, however, exhausted all the possibilities; testing our proposal in a complex rendering engine is a very interesting direction for future work.

Our framework may produce inaccurate predictions when the rendering samples used to fit the model do not include information related to a certain pass (e.g., the *Reflections* pass if no reflective surfaces were being rendered at the time). However, these inaccuracies tend to last only a few frames, and the system eventually self-corrects; we have found that this does not have a relevant impact on performance in the long run.

Acknowledgements

We would like to thank all reviewers for their insightful comments. We also thank Bowen Yu for his contribution in the initial phase of this project, and Julio Marco for helping with figures and proofreading the paper. This research has been partially funded by National Key R&D Program of China (No. 2017YFB1002605), NSFC (No. 61472350), Zhejiang Provincial NSFC (No. LR18F020002), the Fundamental Research Funds for the Central Universities (No. 2017FZA5012), European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (CHAMELEON project, grant agreement No 682080), and the Spanish Ministerio de Economía y Competitividad (projects TIN2016-78753-P and TIN2016-79710-P).

References

[AMS08] AKENINE-MÖLLER T., STROM J.: Graphics processing units for handhelds. *Proceedings of the IEEE 96*, 5 (May 2008), 779–789. 2

[APX14] ARNAU J.-M., PARCERISA J.-M., XEKALAKIS P.: Eliminating redundant fragment shader executions on a mobile GPU via hardware memoization. In *ISCA* (2014). 2

[CCC*16] CHEN W., CHEN W., CHEN H., ZHANG Z., QU H.: An energy-saving color scheme for direct volume rendering. *Computers & Graphics 54* (2016), 57 – 64. Special Issue on CAD/Graphics 2015. 2

[CWC*14] CHEN H., WANG J., CHEN W., QU H., CHEN W.: An image-space energy-saving visualization scheme for OLED displays. *Computers & Graphics 38* (2014), 61 – 68. 2

[DCZ09] DONG M., CHOI Y.-S. K., ZHONG L.: Power modeling of graphical user interfaces on OLED displays. In *Proceedings of the 46th Annual Design Automation Conference* (2009), ACM, pp. 652–657. 2

[GSC*15] GHARBI M., SHIH Y., CHAURASIA G., RAGAN-KELLEY J., PARIS S., DURAND F.: Transform recipes for efficient cloud photo enhancement. *ACM Trans. Graph. 34*, 6 (Oct. 2015), 228:1–228:12. 2

[Hen] HENNESSY D. A. P. . J. L.: *Computer organization and design : the hardware/software interface. Appendix C: Graphics and Computing GPUs.*, 5th ed. ed. The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufmann, Elsevier,, Boston :. 4

[HK10] HONG S., KIM H.: An integrated GPU power and performance model. In *Proceedings of the 37th Annual International Symposium on Computer Architecture* (New York, NY, USA, 2010), ISCA '10, ACM, pp. 280–289. 2

[ILMR03] IYER S., LUO L., MAYO R., RANGANATHAN P.: Energy-adaptive display system designs for future mobile environments. In *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services* (2003), ACM, pp. 245–258. 2

[JGDAM12] JOHNSSON B., GANESTAM P., DOGGETT M., AKENINE-MÖLLER T.: Power efficiency for software algorithms running on graphics processors. In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics* (2012), pp. 67–75. 2

[JGY*11] JIMENEZ J., GUTIERREZ D., YANG J., RESHETOV A., DEMOREUILLE P., BERGHOFF T., PERTHUIS C., YU H., MCGUIRE M., LOTTES T., MALAN H., PERSSON E., ANDREEV D., SOUSA T.: Filtering approaches for real-time anti-aliasing. In *ACM SIGGRAPH Courses* (2011). 9

[KY14] KYUNG C.-M., YOO S.: *Energy-Aware System Design: Algorithms and Architectures*. Springer Publishing Company, Incorporated, 2014. 2

[Lot09] LOTTES T.: FXAA, 2009. URL: https://developer.download.nvidia.com/. 9

[MWDG13] MASIA B., WETZSTEIN G., DIDYK P., GUTIERREZ D.: A survey on computational displays: Pushing the boundaries of optics, computation, and perception. *Computers & Graphics 37*, 8 (2013), 1012 – 1038. 2

[NVM15] NVML: Nvidia management library, 2015. 8

[PLS11] POOL J., LASTRA A., SINGH M.: Precision selection for energy-efficient pixel shaders. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics* (2011), pp. 159–168. 2

[SPP*15] STAVRAKIS E., POLYCHRONIS M., PELEKANOS N., ARTUSI A., HADJICHRISTODOULOU P., CHRYSANTHOU Y.: Toward energy-aware balancing of mobile graphics. In *IS&T/SPIE Electronic Imaging, International Society for Optics and Photonics* (2015), vol. 9411, pp. 94110D–10. 2

[Sta15] STACHOWIAK T.: Stochastic screen-space reflections. In *ACM SIGGRAPH 2015 Courses Advances in Real-Time Rendering in Games* (2015). 9

[VAKH13] VATJUS-ANTTILA J. M., KOSKELA T., HICKEY S.: Power consumption model of a mobile GPU based on rendering complexity. In *2013 Seventh International Conference on Next Generation Mobile Apps, Services and Technologies* (Sept 2013), pp. 210–215. 2, 4

[WBSS04] WANG Z., BOVIK A. C., SHEIKH H. R., SIMONCELLI E. P.: Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing 13*, 4 (April 2004), 600–612. 6

[WYM*16] WANG R., YU B., MARCO J., HU T., GUTIERREZ D., BAO H.: Real-time rendering on a power budget. *ACM Trans. Graph. 35*, 4 (July 2016), 111:1–111:11. 2, 3, 4, 6, 7, 8, 11

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.