Classified Texture Resizing for Mobile Devices

Jae-Ho Nah
LG Electronics
nahjaeho@gmail.com

Byeongjun Choi
LG Electronics
byeongjun.choi@lge.com

Yeongkyu Lim
LG Electronics
limyeongkyu@gmail.com

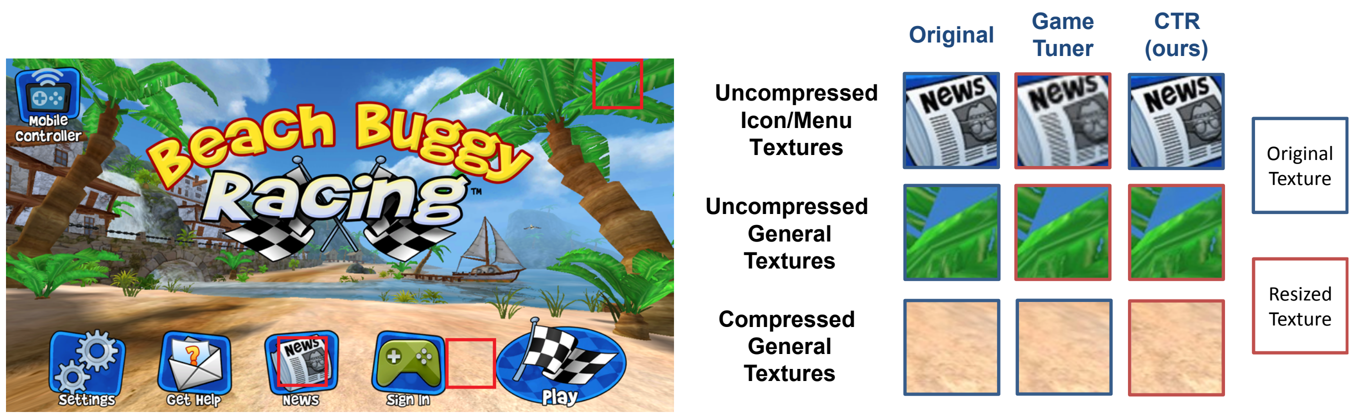

Figure 1: Comparison of Samsung Game Tuner with our classified texture resizing (CTR) method. In contrast to Samsung Game Tuner, our approach can distinguish 2D icon/menu textures and general textures and, as a result, can preserve visibility. Additionally, our approach can resize compressed textures which cannot be handled by Samsung Game Tuner. The screenshot was captured in the game Beach Buggy Racing (courtesy of Vector Unit).

ABSTRACT

Power consumption is one of the most important factors in mobile computing. Especially for high-quality games, it takes a lot of computing power to render visual effects. In order to reduce this, some rendering techniques (e.g., Samsung Game Tuner) adjust rendering parameters (screen resolution, frame rates, and texture sizes) to improve power efficiency or performance. Among them, the texture resizing reduces power consumption in some cases, but it sometimes results in poor rendering quality or no energy saving.

To improve the texture resizing, we present the classified texture resizing technique. Our main idea is to classify textures into certain types and to apply a different approach to each type. As a result, our approach minimizes degradation of rendering quality and can be applied to wider applications. Our experimental results show up to 16% power reduction of a GPU and DRAM.

CCS CONCEPTS

• **Human-centered computing → Mobile devices**; • **Computing methodologies → Texturing**; *Graphics processors*;

KEYWORDS

texture mapping, mobile GPUs, power reduction techniques

ACM Reference Format:
Jae-Ho Nah, Byeongjun Choi, and Yeongkyu Lim. 2018. Classified Texture Resizing for Mobile Devices. In *Proceedings of SIGGRAPH '18 Talks.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3214745.3214763

1 INTRODUCTION

Advanced graphics techniques are becoming increasingly common in high-quality mobile games. However, the use of these graphics techniques sometimes brings out performance, battery, or thermal issues. To relieve these issues, there are various techniques for improving GPU energy efficiency [Mittal and Vetter 2015].

A few smartphone manufacturers have tried to trade high graphics quality off against higher frame rates or less power consumption by providing software tools. LG Game Optimizer introduced in LG G4 decreases screen resolution of game apps at the Android framework level. This method only provides an on/off switch for predefined apps in the whitelist, so its flexibility is limited. As its enhanced version, LG Game Battery Saver in LG G6 offers the ability to control screen resolution and a frame rate of each app to users.

Samsung Game Tuner has similar functions to LG Game Battery Saver but includes one more feature: texture quality control (since v2.1 [2016]). By using this feature, a user can control the size of uncompressed static textures. Its process consists of four stages. First, an uncompressed texture is rendered into an off-screen frame buffer. Second, the result is uploaded into the CPU memory via the glReadPixels() function. Third, the texture is resized on the CPU. Finally, the resized texture is downloaded to the GPU again.

Texture quality control in Game Tuner can be useful when texturing is a performance bottleneck or requires a lot of memory traffic. However, this approach has several limitations. First, quality degradation can occur because game graphics designers usually separate compressed and uncompressed textures to prevent artifacts from compression. Second, the approach has no effect on textures either dynamically created by render-to-texture techniques or stored in a compressed format. Finally, the use of glReadPixels() increases app loading time.

2 CLASSIFIED TEXTURE RESIZING

To resolve the above limitations of Game Tuner, we present a different approach called classified texture resizing (CTR). As its name implies, our solution is based on texture classification. The classification criteria are generation time (static or dynamic), compression, mipmapping, and aspect ratios as depicted in Figure 2. Note that CTR is executed at the OpenGL ES driver level, so the classification is possible by analyzing OpenGL ES commands.

According to the classification, different approaches are applied to textures as follows. We first classify textures as static and dynamic textures. Static textures, which are already stored in an app, are classified into three categories again. First, non-mipmapped, compressed textures, which are usually background images (e.g., sky), are decompressed, resized, and re-compressed. For compression and decompression, we have ported the code of etcpak 0.5 [Taudul 2016], which is the fastest ETC compression library, into the driver. Our current implementation is less optimized than the original etcpak for x86 desktop platforms; only a single thread and no SIMD intrinsics are used. Also, only the ETC1/ETC2 RGB format is currently supported.

Second, mipmapped textures are resized with a simpler method. By decreasing the mipmap level of each texture by one in the command dispatcher, the maximum size of each texture can be decreased to one fourth. It is hard to distinguish mipmapped and non-mipmapped textures before a level-1 texture is loaded. Thus, our current implementation first performs ETC1 re-encoding when a

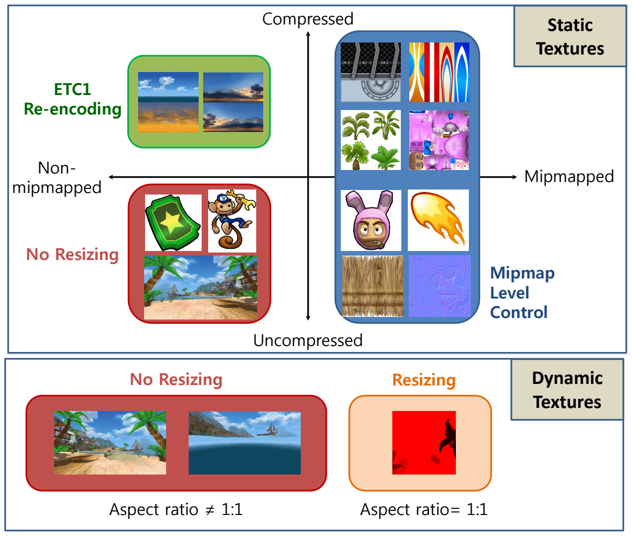

Figure 2: Our texture classification. A different approach is applied to textures in each category. The example textures are included in Beach Buggy Racing.

compressed level-0 texture is loaded. After that, the mipmap level control is performed from level 1 to the maximum level. In the case of uncompressed textures, CTR is much simpler than Game Tuner, so the resizing overhead is minimized.

Third, non-mipmapped, uncompressed textures are not handled because they are usually one-to-one mapped to a screen (e.g., icons, menus, screen-sized images, etc.). Because these textures are not compressed on purpose by graphics designers, CTR preserves image quality in contrast to Game Tuner.

In the case of dynamic textures which are generated on-the-fly with render-to-texture techniques, we use aspect ratios for our classification to resize only shadow maps. The reason for this is that resizing the main frame buffer or G-buffers has a side effect, redundant screen resolution resizing, when the screen resolution control is used together. To avoid that, we only resize textures with a 1:1 aspect ratio; shadow maps are usually classified as this type.

In summary, CTR provides the following advantages compared to Game Tuner: additional resizing compressed textures and shadow maps; efficient resizing mipmapped textures; and quality preservation of non-mipmapped, uncompressed textures.

3 EXPERIMENTAL RESULTS

For experiments, we executed three mobile games (Beach Buggy Racing, Implosion, and Xenowerk) on a MADK development board with an ARM 64-bit Cortex CPU, ARM Mali GPU, and LPDDR3 RAM. When executing the games, we used a National Instruments USB-6363 device to measure exactly the power consumption of the GPU and DRAM.

Compared to no texture resizing, the saving rates for power consumption of each game are 10.3% (Beach Buggy Racing), 4.2% (Implosion), and 16.3% (Xenowerk), respectively. As illustrated in Figure 1, in contrast to Game Tuner, we can preserve the visibility of icon and menu textures. Note that the resizing factors of both methods in the figure are commonly 25% and a Galaxy S7 was used for the comparison.

The loading time of each app is increased by 2.2 s (Beach Buggy Racing), 0.3 s (Implosion), and 2.8 s (Xenowerk), respectively. In Beach Buggy Racing and Xenowerk, ETC1 RGB texture re-encoding increases loading time, and these are comparable results to Game Tuner (~2 s). Implosion has only ETC2 RGBA textures, so this game does not lead to texture re-encoding and does show only a negligible loading time increase within the margin of error.

4 LIMITATIONS AND FUTURE WORK

We would like to decrease app loading time through cleverer mipmap detection and multi-threaded texture compression with ARM Neon intrinsics. Additionally, the current version only supports re-encoding ETC1/2 RGB textures. We would like to support various data types (e.g., RGBA, 3D, HDR, etc.) and texture compression formats (e.g., ASTC) as future work.

REFERENCES

Sparsh Mittal and Jeffrey S Vetter. 2015. A survey of methods for analyzing and improving GPU energy efficiency. *ACM Computing Surveys (CSUR)* 47, 2 (2015), 19.
Samsung Electronics. 2016. Game Tuner. https://play.google.com/store/apps/details?id=com.samsung.android.gametuner.thin. (2016).
Bartosz Taudul. 2016. etcpak 0.5. https://bitbucket.org/wolfpld/etcpak. (2016).