Half Frame Forwarding:Frame-rate Up Conversion for Tiled Rendering GPU

Jinhong Park, Minkyu Kim, Sunho Ki, Youngduke Seo, Chulho Shin
LG Electronics, Korea *

(a) (b) (c)

Figure 1: (a) GFXBench3.0 Manhattan golden image, (b) our result, (c) difference image between a golden and ours

1. Introduction

Although the mobile industry has recently begun trending towards high quality graphics content, it is still difficult to satisfy this trend due to performance, power and thermal issue of GPU/CPU in mobile application processor.

The power consumption of mobile GPU increases almost linearly with workload, depending on the graphics content. Thus, lowering the frame rate can reduce the power consumption. Some applications force the skipping of some frames during rendering in order to decrease the frame rate. Although this feature can be exploited at runtime in order to reduce the power consumption, it can make noticeable artifacts such as flickering and lagging. One common technique to help remedy these artifacts is a frame rate up conversion (FRUC) algorithm[Bowles 2012], which typically consist of two processes to determine the dynamic objects in consecutive frames: motion estimation and motion-compensated interpolation. But, those have high computational costs and, naturally, result in high power consumption. Thus, the conventional approaches are not suitable for mobile devices.

In this work, we propose a novel FRUC algorithm based on a half frame forwarding approach with a low cost solution to detect dynamic objects for tile-based GPU rendering. The proposed algorithm makes intermediate frames using the tiles that cover a region with dynamic objects. Using an LG G3 Screen, our experimental result demonstrates that the proposed algorithm can reduce the system power consumption by up to 20.4%.

2. Our Approach

The most mobile GPUs, such as Imagination Technologies PowerVR, are based on the tile-based rendering algorithm, which decreases off-chip memory traffic. The tile-based rendering algorithm divides the frame buffer into smaller tiles (e.g. 16x16 pixels per tile) and renders each of them separately using high-speed on-chip memory. Once per-tile rendering is complete, the final pixels are written to a frame buffer in off-chip memory.

This idea comes from the observation that the tile-based rendering is performed in per-tile order; one after the other and stored in the frame buffer. Our algorithm generates the intermediate frame by composing the tiles from a region with dynamic objects and the previous frame when a half of the current frame is rendered.

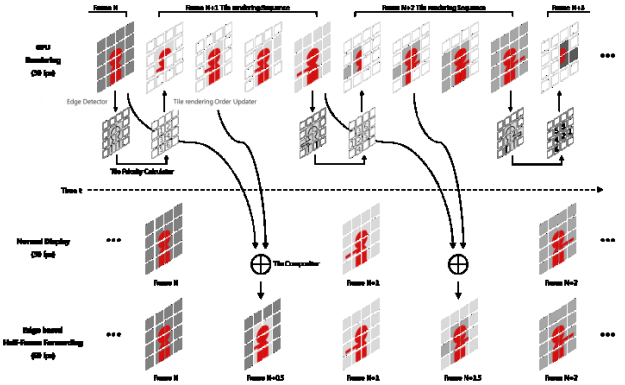

Figure 2: Our Algorithm

Figure 2 shows the proposed algorithm. Supposing that N^{th} frame was rendered, we first extract a region of the dynamic objects. Second, the tiles of this region should be assigned priorly to GPU for the rendering of $N+1^{th}$ frame. Then, these tiles are rendered one by one and overwritten to the previous N^{th} frame, which will be the intermediate $N+0.5^{th}$ frame and displayed. In order to extract a region of dynamic objects, we exploit the per-tile-based edge detection. We compute the edge detection algorithm in the tiles of the same location from the consecutive two frames, and the algorithm derives the number of pixels as well as color information from an edge region of the tile. As the pixel number of the edge region is higher, a higher priority for the next frame rendering is given to this tile; if not, the algorithm determines that this tile does not contain dynamic objects. But, in the case of a region that the static objects with the dynamic light sources, this region is also regarded as having dynamic object changes. Thus, the color changes of the pixels should be additionally considered.

For our experiments, we use three test vectors: GFXBench 3.0 Manhattan, T-Rex, and Android Lollipop UI. The simulation was performed using the LG G3 Screen platform which has Imagination's RGX GPU. Figure 1 shows the golden frame, the intermediate frame from our experimental results and the difference between them. Further, we gain 8.5%, 19.2%, and 20.4% of power saving for each given application while maintaining the same frame rate compared to the previous results.

References

Bowles, H., Mitchell, K., Sumner, R. W., Moore, J., & Gross, M. 2012. Iterative image warping. *Computer graphics forum* 31,2pt1, 237-246.

*e-mail: {jinhong1.park, mink.kim, sunho.ki, youngduke.seo, chulho.shin} @lge.com

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author.
Copyright is held by the owner/author(s).
SIGGRAPH 2015 Posters, August 09 – 13, 2015, Los Angeles, CA.
ACM 978-1-4503-3632-1/15/08.
http://dx.doi.org/10.1145/2787626.2787634