Novel approaches for GPU Performance Analysis

Karthik Hariharakrishnan*
ARM® Ltd, Cambridge, UK

Keywords: timeline, performance, deferred rendering, embedded GPU

1 Introduction

Most modern embedded GPU architectures use a concept called *deferred rendering* - a rendering job submitted to the GPU gets scheduled at a future point in time. When a graphics application issues a rendering API call(e.g. OpenGLES® call), the graphics driver running on the CPU, stores the state necessary for that call, but doesn't execute it on the GPU immediately. The CPU consumes subsequent API calls to build a rendering job for the GPU. When the application wants to display the result of rendering on a window(eg SwapBuffers), the CPU submits the constructed job to the GPU. This architecture is especially suited for an embedded GPU as it reduces communication and bandwidth between the CPU and GPU. Once the job has been submitted to the GPU, the CPU is free to work on preparing the next frame. It is important to ensure that different processing units(CPU and GPU) are kept busy running in parallel. An application that consumes a lot of time doing CPU computation, will starve the GPU and vice-versa. Understanding the relationship between the CPU and GPU is vital for developers who want to efficiently utilize the GPU. Timeline charts capture the amount of time a processing unit is busy. A timeline chart in the most basic form is a binary chart that indicates activity on a processing unit over time. This presentation discusses the state-of-the art approaches for capturing timeline and then discusses a different approach that moves both capture and visualization to the target device.

2 Exposition

For performance analysis of applications running on embedded GPUs, the most vital piece of information to find out is how loaded the GPU is over a period of time. As discussed above, this is captured by a *timeline chart* for each processing unit in the GPU.

The latest approaches for capturing timeline information from a target are discussed. The current suite of GPU performance analysis tools that run on desktops provide an integrated solution for profiling the complete system that includes both the CPU and GPU. We then discuss how some of the GPU timeline information can be directly captured and displayed on the target device. Hardware counters that describe the load on different parts of the GPU can also be acquired along with the timeline information. This provides very useful first level information for quickly identifying the bottleneck and load on the GPU's processing units.

The technique has been demostrated as an application developed in Android™. The Android application uses a service to communicate with the GPU device driver and retrieve the timeline information for the application that is being profiled. Once the profiling stops, the acquired timeline information is displayed on the target device.

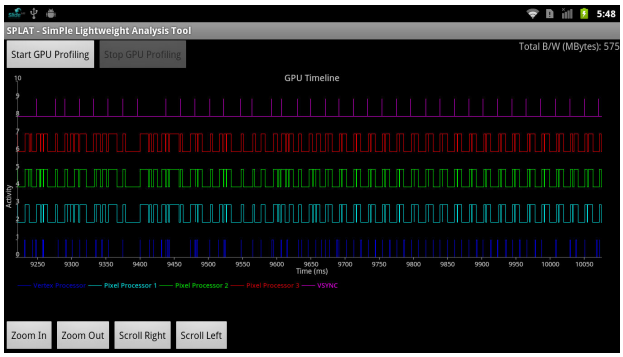

Figure 1: *Snapshot of Android application*

3 Elaboration

The acquisition of timeline data from the GPU consists of the following steps. The first step is to establish a way to retrieve hardware information from the GPU device driver(across the user-kernel interface). This can be done by the addition of necessary 'ioctl' (input/output control) calls. These ioctls are used to start/stop the profiling and to retrieve the profiling data. All of this is written using the Android NDK. The collected data is then plotted as a line chart showing the activity on the different cores in the GPU. The acquired hardware counters are also displayed to give more insight into the bottleneck. Another interesting use-case of such a technique, would be to use the GPU hardware information to take a different software path, thereby making a closed-loop software system with hardware performance feedback.

4 Conclusion

This presentation discussed a different approach for performance analysis where the acquisition and display of data was done on the target. A technique for this approach was developed in Android to profile 3D applications running on the same device.

*e-mail:karthik.hariharakrishnan@arm.com

SIGGRAPH 2012, Los Angeles, California, August 5 – 9, 2012.
ISBN 978-1-4503-1435-0/12/0008