Hybrid Curve Rendering Scheme for Mobiles
JeongJoon Yoo
Samsung Electronics
jeongjoon.yoo@samsung.com

Abstract—In this paper, we present an efficient curve rendering method useful in smartphone and mobile device that have dynamic workload changes between CPU and GPU. To do so, we propose a Hybrid scheme that adaptively tessellates a curve into a set of small triangles and sub-curves to render them effectively on GPU. Experimental comparisons show that our scheme not only outperforms others but also reduces the power consumptions. As the result, we can effectively enjoy our scheme in any smartphones and mobile devices regardless of any workload environment between CPU and GPU.

Keywords—Curve rendering, Workload, Mobiles, CPU, GPU

I. INTRODUCTION

With increasing the resolution size of display, resolution independent graphics or path rendering is much more recognized as an important rendering scheme than ever [1, 2, 3, 4]. In the resolution independent graphics, curve rendering is the most of compute intensive work in smartphone usually found in applications including games, map app, and font rendering et al. There have been several researches and implementations of curve rendering. In the era of beginning of smartphone, the curve rendering was mainly executed on CPU rather than on GPU. Later Loop and Blinn proposed a resolution independent curve rendering algorithm that can be effectively executed on GPU [1].

The problem of such curve rendering schemes in smartphone or mobile device is that they are device dedicated algorithms. It means that they are designed to run on dedicated device (CPU or GPU) regardless of the workload of CPU or GPU. Because they did not consider the current workload of CPU and GPU, they suffer from performance bottleneck. For example, even though the Loop-Blinn scheme was designed to be executed on GPU, it shows performance degradation when the GPU is busy with other jobs.

To solve this problem, we propose a new curve rendering scheme that can be used in smartphone or mobile device environment where its CPU and GPU workload are dynamically and frequently updated.

The remainder of this paper is organized as follows. In section II, we will explain previous schemes for curve rendering. In section III, we introduce our Hybrid scheme that is effectively executed on smartphone or mobile device that the workload of CPU and GPU is ever-changing. In section IV, the performance and the power consumption of our scheme will be compared with the previous schemes. Finally, we will conclude the paper in section V.

II. RELATED WORK

A. CPU-based curve rendering
De_casteljau [5] with polygon scan conversion [6] was generally used to render a curve on CPU. It tessellates a curve into many numbers of contiguous lines. After then it fills with a scanline from the start point on a left line to the end point on another right line as shown in Fig 1 (a). This scheme is called SW_Tess scheme executed on CPU. Such scheme was used in skia [7] and cairo [8] 2D graphic libraries used in smartphone. Because a software approach for a curve tessellation occupies many resources of CPU, it is not suitable to use this scheme when CPU is busy. To overcome this problem, some ‘CPU + GPU’ schemes were proposed.

B. CPU + GPU-based curve rendering
To relieve a burden of CPU of SW_Tess, multi-stage tessellation scheme [9] was proposed as shown in Fig 1 (c). In the 1st stage, a curve is roughly tessellated into triangles in CPU. In the 2nd stage they are again finely tessellated in GPU. Although it reduces the burden of CPU, an additional hardware tessellator is required in GPU. So, we cannot use this scheme in the current commercial GPU.

Loop-Blinn [1] proposed a resolution independent curve rendering algorithm that can be effectively executed on GPU. This scheme regards a curve as one or two triangle(s) (one/two triangle(s) for a quadratic/cubic curve respectively) as explained in Fig 1 (d). Each triangle is submitted to GPU and each part of a curve is rendered separately. That is, all fragments in each triangle execute curve inside test, and then inside pixels are survived with a color. The final image is display as shown in Fig 1 (b).

Because this scheme mainly rely on GPU to render a curve, the performance is decreased when the GPU is busy.

Whenever smartphone technologies are increasingly innovated like virtual reality, augmented reality and machine learning, workload between CPU and GPU is ever-changing. However, the previous curve rendering schemes are designed to run on dedicated device like CPU or GPU regardless of the workload of CPU or GPU. Because they did not consider the workload of CPU and GPU, they suffer from performance bottleneck.

To remove such problem, we propose a new curve rendering scheme that adaptively changes the amount of work
Fig 2. Hybrid Curve Rendering Scheme

between CPU and GPU depending on the relative size of workload between CPU and GPU in smartphones.

III. HYBRID CURVE RENDERING

A rational for our algorithm comes from the fact that the rendering cost for one pixel inside a curve is expensive than the rendering cost for one pixel inside a triangle. It is because checking a point being inside a curve (which is 2 or 3-dimensional equation) is expensive than checking a point being in a triangle (which is 1-dimensional equation). So, when the GPU shader core is busy, we try to reduce overall workload in GPU shader core to render a curve. This is the motivation and a basic idea of our scheme.

More specifically, depending on the workload between CPU and GPU in the smartphone, we change the tessellation level that determines the level of tessellation for a curve. Unlike SW_Tess, a curve is tessellated into a set of sub-curves and a set of triangles in our scheme (Hybrid; tessellates a curve into hybrid primitives) as shown in Fig 2.

Initially, the default value of Tess_level is zero. When the utilization of GPU is relatively larger than the utilization of CPU, the value of Tess_level is increased. When Tess_level = 1, the curve is tessellated into two sub-curves (C1 and C2) and one triangle (T1). When the Tess_level = 2, the curve is tessellated into four sub-curves (C3, C4, C5, and C6) and three triangles (T1, T2, and T3). This kind of triangulation reduces overall workload in GPU shader core. On the other hand, it increases overall workload of CPU for tessellation.

When the utilization of CPU is larger than GPU, the value of Tess_level is decreased. The set of curves are rendered with the help of GPU shader core. On the other hand, the set of triangles are rendered with the help of GPU fixed pipeline.

The experimental comparisons between our scheme and others will be done in the next section.

IV. EXPERIMENTAL COMPARISONS

In this section, we compare the performance (fps) and power consumption of our Hybrid scheme with previous ones. Because the hardware accelerator for a curve is not implemented yet in the commercial GPU, the multi-stage tessellation scheme is not compared with ours.

We rendered a scalable vector graphics file having thousands of curves on Galaxy S8 with Exynos8895.

When the utilization of CPU is larger than the utilization of GPU (CPU bound), Loop-Blinn and our schemes (Hybrid) has almost same performance. On the other hand, when the utilization of GPU is larger than the utilization of CPU (GPU bound), our scheme outperforms Loop-Blinn as shown in Fig 3. We also compared the power consumption of curve rendering schemes. Hybrid scheme outperforms others and its effectiveness was increased when the GPU is much bottleneck.

The result of performance and power consumption is because we reduced the number of cost expensive computations for a curve and replace them into cost inexpensive fragment shaders for both triangles and sub-curves. Although Loop-Blinn and SW_Tess are not executed adaptively on dynamic workload environment, our Hybrid scheme meets the adaptability of smartphone environment that the workload of CPU and GPU is ever-changing.

V. CONCLUSIONS

Our contribution in this paper is proposing Hybrid curve rendering scheme that can be used in dynamic workload smartphone environment. Because we reduced the number of cost-expensive computations and replaced them with the cost in-expensive triangle rendering, we can get higher performance and reduced the power consumptions when the smartphone is GPU bound. As a result, we got 78% increase of the performance and 39% decrease of the power consumption. Also because our scheme works not only on GPU bound but also on CPU bound environment adaptively, we can reduce the performance optimization period for new release of smartphone. Our scheme immediately used in any current commercial GPU without any additional hardware extension. As a conclusion, we can use our Hybrid scheme in any smartphone environment and with any applications workload on it.

REFERENCES