Hardware implementation of a tessellation accelerator for the OpenVG standard

Seung Hun Kim, Yunho Oh, Karam Park, and Won Woo Ro^{a)}

School of Electrical and Electronic Engineering, Yonsei University,
134 Shinchon-Dong, Seodaemun-Gu, SEOUL, 120-749, KOREA

a) wro@yonsei.ac.kr

Abstract: The OpenVG standard has been introduced as an efficient vector graphics API for embedded systems. There have been several OpenVG implementations that are based on the software rendering of image. However, the software rendering needs more execution time and power consumption than hardware accelerated rendering. For the efficient hardware implementation, we merge eight pipeline stages in the original specification to four pipeline stages. The first hardware acceleration stage is the tessellation part which is one of the pipeline stages that calculates the edge of vector graphics. In this paper, we provide an efficient hardware design for the tessellation stage and claim this would eventually reduce the execution time and hardware complexity.

Keywords: OpenVG, vector graphics, tessellation, hardware accelerator

Classification: Electron devices, circuits, and systems

References

- [1] K. Pulli, "New APIs for mobile graphics," *Proc. SPIE - The International Society for Optical Engineering*, vol. 6074, pp. 1–13, 2006.
- [2] Khronos Group Inc., "OpenVG specification Version 1.0.1"
[Online] <http://www.khronos.org/openvg/>
- [3] S.-Y. Lee, S. Kim, J. Chung, and B.-U. Choi, "Salable Vector Graphics (OpenVG) for Creating Animation Image in Embedded Systems," *Lecture Notes in Computer Science*, vol. 4693, pp. 99–108, 2007.
- [4] G. He, B. Bai, Z. Pan, and X. Cheng, "Accelerated Rendering of Vector Graphics on Mobile Devices," *Lecture Notes in Computer Science*, vol. 4551, pp. 298–305, 2007.
- [5] J. D. Hobby, "Practical segment intersection with finite precision output," *Computational Geometry*, vol. 13, no. 4, pp. 199–214, Nov. 1999.
- [6] P. Shirley, "Transformation Matrices," in *Fundamentals of Computer Graphics*, 2nd edition, pp. 135–154, AK peters, 2005.

1 Introduction

The performance of modern embedded systems has been significantly im-

proved in recent years. Due to this advance, the user's demands have been increased and many innovative applications have been developed [1]. For examples, GUI (Graphic User Interface), navigation systems, and SVG (Scalable Vector Graphics) player become widely used in various electrical appliances that need high quality graphic environment. As a result, the vector graphics implementation on the mobile device has been an important topic.

The vector graphics maintain its quality regardless of the modification on the size while the raster graphics could not. This quality consistency is a suitable property for the embedded systems that have different display sizes. From this point of view, the Khronos group has constituted the OpenVG standard which is optimized vector graphics API for the embedded systems [2].

The Khronos group provides the sample implementation of the OpenVG which is based on software rendering. In addition, Lee *et al*, He *et al* also have introduced another software implementation [3, 4]. However, the software rendering is not a suitable approach for the embedded systems since the vector graphics need more mathematical calculations than raster graphics to show the high quality graphics image. Therefore, demands on a new implementation as a form of hardware accelerator exist. In fact, an existing 3D GPU (Graphics Processing Unit) can be used as the OpenVG hardware accelerator by adopting the tessellation technique [5]. However, there might be some problems since the 3D graphics hardware is not tailored for the 2D acceleration. First, tessellating the arbitrary paths to triangles causes significant pre-processing which affects uneven frame rates. In addition, it would require an additional software driver for complex polygon tessellation.

To address the problems mentioned above, we propose a new hardware design for the fast execution of the tessellation operation in the OpenVG specification. Our final goal is to design a complete hardware accelerator. The tessellation is the first step for designing a full 2D hardware accelerator of specific vector graphics API. We optimize the necessary calculation for rendering to provide an efficient hardware design.

2 OpenVG paths data process

One of the main factors for vector graphics is drawing geometry. All geometries of OpenVG to be drawn must be described with one or more paths that are determined in the OpenVG specification. Paths consist of a sequence of segment commands; a segment command in the standard format may specify a move, a straight line, a Bezier curve, or an elliptical arc [2].

The rendering pipeline of OpenVG is defined as eight stages in the specification. However, implementations are not required to match the given pipeline stage-for-stage. We merge the second and third stage as one which named tessellation. This adjustment simplifies the hardware architecture. In the tessellation, the coordinates are calculated in accordance with the segment commands and the edges are generated by using the calculated coordinates. The edge is the vector information that should be converted into

a raster format before being displayed on the device. In the process of tessellation, operands for the mathematical calculations are all floating point numbers. Indeed, we aim to reduce the execution time and power consumption by using hardware floating point arithmetic units.

3 Design algorithm

The proposed tessellation hardware includes three sub-modules: the vertex data generation module, the transformation module, and the edge data generation module. Each of three modules is comprehensively described in this section. In addition, there are totally eight multipliers, eight adders, one square root arithmetic unit, and two dividers; all these functional units are shared by three sub-modules.

3.1 Vertex data generation module

All geometric plots are composed of line segments and each line segment requires the vertex information. The vertex information includes a vertex position, unit tangent vectors, and vertex flags. The necessary data can be obtained by the vertex data generation module.

In the case of the curve commands, the vertex position is calculated with the cubic Bezier curve equation (Eq. (1)) or the quadratic Bezier curve equation (Eq. (2)). In the equations, $\mathbf{P}_n$ is a control point of the Bezier curve, which is the two dimensional vector with two components x and y .

$$\mathbf{P}(t) = (1-t)^3\mathbf{P}_0 + 3t(1-t)^2\mathbf{P}_1 + 3t^2(1-t)\mathbf{P}_2 + t^3\mathbf{P}_3, \quad t \in [0, 1]. \quad (1)$$

$$\mathbf{P}(t) = (1-t)^2\mathbf{P}_0 + 2t(1-t)\mathbf{P}_1 + t^2\mathbf{P}_2, \quad t \in [0, 1]. \quad (2)$$

We have factored the cubic Bezier curve equation as Eq. (3) to minimize the number of floating point multiplier and adder. The vertex data generation module calculates Eq. (3) as depicted in Fig. 1. The line between the arithmetic units means the flow of operands and results instead of hardwired

Fig. 1. Solving process of cubic Bezier curve

structure.

$$\mathbf{P}(t) = (1-t)^3\mathbf{P}_0 + 3t(1-t)\{(1-t)\mathbf{P}_1 + t\mathbf{P}_2\} + t^3\mathbf{P}_3 \quad (3)$$

The operands and results of the arithmetic units are saved in the specific registers and the controller of the vertex generator module allocates these registers to the inputs of the arithmetic unit. Therefore, the same unit can be utilized again for each step in Fig. 1. For example, operands of the second multiplier are $1-t$ and x_1 at step 1; x_1 is a x component of vector $\mathbf{P}_1$. After then, the controller changes the operands as constant 3 and t at step 2.

As mentioned above, $\mathbf{P}$ has two components. Therefore, eight multipliers and two adders are needed to calculate the equation. One additional floating point adder is needed to calculate the parameter t .

Eq. (2) is less complex than Eq. (1) so the module calculates the Eq. (2) in four steps without factoring. $(1-t)^2$, $2x_1$, $2y_1$, $t(1-t)$, t^2 are calculated at first by using five multipliers and then, results are multiplied with components of vector $\mathbf{P}_0$, $\mathbf{P}_2$ or other results.

3.2 Transformation module

The coordinate data has to be converted into the screen coordinates system to rasterize the vector graphics. The transformation of the coordinates system is calculated by multiplying a three-dimensional matrix with a coordinate vector [6]. However, the meaningful calculations of that can be summarized as Eq. (4) to (7).

$$x_s = (x_u - \text{Min}_x) \times \text{width_ratio} \quad (4)$$

$$\text{width_ratio} = \frac{\text{screen width}}{\text{Max}_x - \text{Min}_x} \quad (5)$$

$$y_s = (y_u - \text{Min}_y) \times \text{height_ratio} \quad (6)$$

$$\text{height_ratio} = \frac{\text{screen height}}{\text{Max}_y - \text{Min}_y} \quad (7)$$

The suffix s denotes the screen coordinates system and the suffix u denotes the user coordinates system. Max_x , Max_y , Min_x , and Min_y mean the maximum and minimum coordinates of the user coordinates system. The screen width and the screen height represent the display size.

The transformation module calculates the width ratio and height ratio only once since these values are fixed for one vector graphics image. Therefore, transformation of the coordinate system can be performed simply by multiplying the coordinates with the aspect ratio. In addition, if Min_x and Min_y are all zeros, the controller of the module changes the state which excludes the subtracting operations. Two dividers for floating point division are needed in the transformation module.

3.3 Edge data generation module

The edge data consists of x and y coordinates for two points: the starting point and ending point. It also includes the reciprocal of slope that are produced by the vertex data generation module.

The controller of the edge data generation module operates differently depending on the line “stroking” option; *whether the line has to be stroked or not*. Stroking is drawing a line with a given thickness. If the geometric plot does not need stroking option, the module reads two sets of vertex information which have the same unit tangent vectors. After that, the module defines the starting point as the first vertex position and the ending point as the second vertex position. The reciprocal of slope is calculated from the unit tangent vectors.

When the geometric plot is drawn with the stroking option, we first need to find four vertices that are required to generate the edge data as shown in Fig. 2.

Fig. 2. Edge generation using vertex

The first line segment consists of four vertices that are *al*, *ar*, *bl*, and *br*. In our proposed design, the module calculates the coordinates of *al* and *ar* with Eq. (8) to (11).

$$al_x = \sqrt{\frac{w}{2}} \times (-ua_y) + a_x \quad (8)$$

$$al_y = \sqrt{\frac{w}{2}} \times (ua_x) + a_y \quad (9)$$

$$ar_x = \sqrt{\frac{w}{2}} \times (ua_y) + a_x \quad (10)$$

$$ar_y = \sqrt{\frac{w}{2}} \times (-ua_x) + a_y \quad (11)$$

Suffix *x* and *y* denotes the coordinates for each axis and the prefix *u* means unit tangent vectors. *w* in the equations is a stroke width. For example, *ua_y* means that *y*-coordinates of unit tangent vector at vertex *a*. Other coordinates of edges can be calculated in the same way. The module calculates the reciprocal of slope from the unit tangent vectors of the vertex *a*. We have utilized eight multipliers and eight adders to calculate the coordinates of four vertices at same time.

4 Implementation results

Proposed algorithm is designed in Verilog HDL and implemented on a Xilinx Virtex-4 xc4vlx80. Xilinx ISE (version 10.1) has been used for the place and route procedure. In order to compare the performance of the hardware acceleration, we have designed the software which performs the same calculation

with the proposed tessellation hardware. In fact, the execution time on the second and third stages of the original OpenVG pipeline is measured and compared.

The software is designed in the C language and simulated on ARM Developer Suite (version 1.2). The target processor is ARM926EJ-S and the simulation results show 1110.365 *ms* execution time for 71,912 vertices and 69,066 edges at 200 MHz operation frequency. On the other hand, the designed tessellation hardware simulated on Xilinx ISE (version 10.1) using Model Sim (version 6.0) and the execution time is measured as 95.557 *ms* for the same number of vertices and edges at 100 MHz operation frequency. Our design reduces the execution time by 91.4% compared to the software implementation and this is equivalent to a speedup of 11.6.

In addition, we have designed the proto-type of rasterizer and per-pixel operation module to verify the hardware design and its operation. The rasterizer module is implemented using fill scan line algorithm and it calculates coverage values of every pixel in the display device. The per-pixel operation module calculates the color information using the coverage value and includes the display controller in the module. The demo graphics with our implementation is shown in Fig. 3.

Fig. 3. Tessellation verification in LCD of FPGA kit

5 Conclusion

This paper proposes a design of our tessellation accelerator which can be used in the OpenVG standard for geometric plots. The hardware calculates the vertex positions, unit tangent vectors, and edge data of geometries that are described in the path segment commands of OpenVG. There are totally eight multipliers, eight adders, one square root arithmetic unit, and two dividers; those functional units are shared by three sub-modules. We have achieved fast execution of the tessellation operation and consequently 11.6 speedup compared to the software implementation. Also the functional correctness of the design is verified on the FPGA kit.

As future work, our proposed hardware will be compared to other hard-

ware accelerators. In that evaluation, we will also demonstrate the practical feasibility with considering effectiveness of the proposed design.

Acknowledgement

This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2009-0070364) and by LG Electronics.