A Vector Graphic Accelerator for Embedded Systems

Y. Choi[1], E.-K. Hong[1], G.-H. Lee[1], Y.-L. Shen[1], T.-K. Kim[1], H.-G. Kim[2], and H.-C. Oh[3]
[1] Parallel Computation Lab., #231B Bio-Sci Bldg(Green), Korea Univ., Seoul 136-713, Korea
[2] R&D center, Advanced Digital Chips, #1009-5 DaeChi-Dong, Seoul 135-280, Korea
[3] Dept. of Info. Eng., Korea Univ., Chung-Nam 339-700, Korea
E-mail: [3]ohyeong@korea.ac.kr

Abstract: This paper presents a prototype hardware accelerator for two-dimensional vector graphics applications based on the OpenVG standard. Since our design mainly targets embedded applications, we focus on efficient uses of limited resources, especially the memory bandwidth. The designed accelerator can process images of 640x240 pixels with moderate complexity at the rate of 30 frames per second. Our current design costs approximately .40 million equivalent gates when it is implemented using a 0.18um CMOS standard cell library.

1. Introduction

Unlike the bitmap (or raster) graphics, the vector graphics models images using mathematical expressions which requires relatively small amount of data. Using those mathematical expressions, an image can be easily converted to an image of an arbitrary size while not losing its quality that much. Due to these advantages, the vector graphics is drawing interests from the industries producing various systems.

As vector graphics applications earns growing popularity in various systems, efficient acceleration schemes for vector graphics need to be developed at various perform- ance levels[1-3]. This paper introduces a prototype hard-ware accelerator for for the vector graphics applications based on the OpenVG standard[4]. The standard provides a low-level two-dimensional vector graphics Application Programming Interface (API) for hardware acceleration [4]. Even though it is also possible to accelerate the OpenVG graphics using the resources prepared in the graphic processors for three-dimensional graphics, special-purpose hardware for two-dimensional graphics is still an attractive solution for embedded systems in which the implementation cost and power consumption are crucial.

The accelerator is designed to be used in an SoC (System on a Chip) in which ADChips' AE32KC processor is the host processor. Since the SoC targets the applications in resource-limited areas, the main objective of our design is to utilize the memory bandwidth efficiently.

Closely following the reference implementation (RI) [5] of OpenVG, we divide the OpenVG rendering process into four stages: Application, Tessellation, Rasterization and Scissoring, and PixelPipe stages. From our analysis using the RI and a variety of (simple) applications running on the AE32KC processor, we found that most of the execution time is spent in executing the stages after the tessellation stage. Thus, our current design implements in hardware the stages after the tessellation stage, as shown in Figure 1. Up to the tessellation stage, we currently rely on the software implementation modified for the host processor.

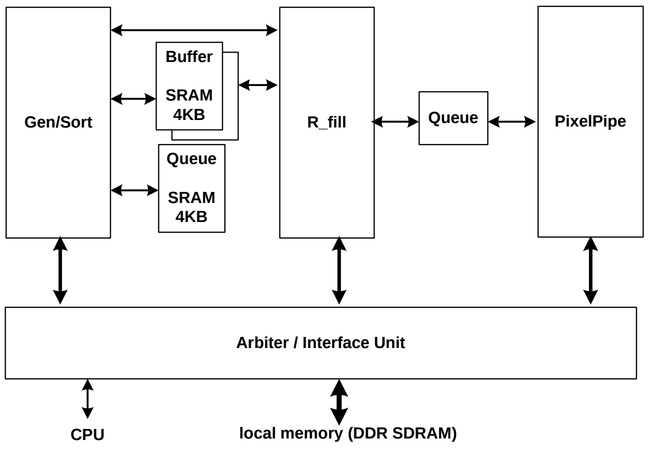


Figure 1. Block diagram of the accelerator

The accelerator is designed to be able to render 30 frames of 640x480 images per second, while it is able to render larger images such as 1024x768 images at a lower rate. Since the local memory often constitute the bottleneck of the system, we adopt one 4KB SRAM (write) queue and two 4KB SRAM buffers to reduce the amount of memory accesses. Using the double buffers, one scan line can be processed by the Gen/Sort stage, while the previous one is being rasterized by the R_fill stage.

The designed accelerator has been modeled in VerilogHDL and simulated using Cadence NC-verilog. Then it has been synthesized on approximately .40 million equivalent gates using a 0.18um CMOS standard cell library. Synopsys DesignWare library has also been used in our simulation and synthesis.

The rest of this paper is organized as follows: Section 2 explains the architecture and basic modules of the proposed accelerator. Section 3 describes our evaluation of the accelerator. Finally, in Section 4 we conclude this paper and discuss directions for future work.

2. The Accelerator

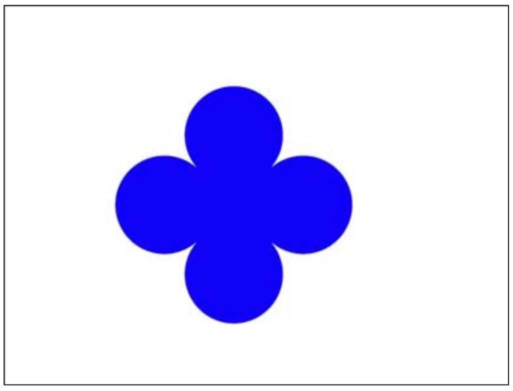
As shown in Figure 1, the accelerator consists of three macro stages, denoted as Gen/Sort, R_Fill, and PixelPipe. The Rasterization and Scissoring stage consists of two stages: Gen/Sort and R_fill. Each macro stage is also designed in the pipelined fashion.

The Gen/Sort stage generates the active-edge table and sorts the active edges which will be used in the R-fill stage. We have sought a sorting scheme that makes the utilization of the data accessed from memory efficient, in order to reduce the execution time in the rasterization stage. It is also crucial to supply data in time from memory, which is the main function of the Arb/IF unit.

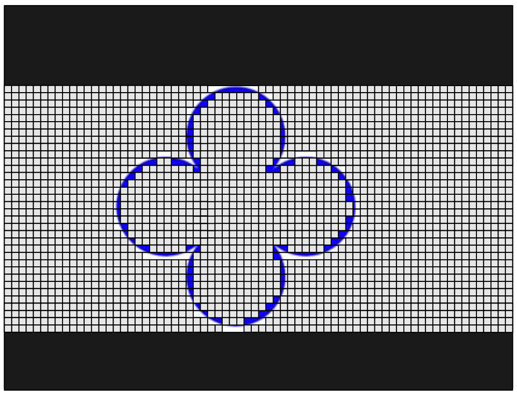
2. 1 The Gen/Sort Module

The Gen/Sort module supplies the active-edge data to the R_fill module. It generates the data from the edges that are suppied by the Tessellation stage, a software stage in this work. Then it sorts the active-edge data by pixels for each scan line. This sorted active-edge data is used for the edge fill operation[2] in the R_fill module.

The edges associated with the same pixel are not sorted. So, the edges are sorted in a fashion similar to the bucket sort. Since an acitvie edge data can belong to serveral pixels, this information is stored in the data of the rightmost pixel.



(a)



(b)

Figure 2. A moving flower. When the image in (a) is being processed, the black and gray parts in (b) have no edge to be sorted.

Some frame includes lots of scan lines and/or pixels that have nothing to do with active edges so that they can be excluded from the sorting process. When the image in Figure 2(a) is being processed, for example, the black and gray parts in Figure 2(b) have no edge to be sorted. The Gen/Sort module, therefore, uses the maximum and the minimum of the coordinates over all the edges in the frame being processed to sort out those scan lines and pixels which do not have any active edges involved. In order to reduce the memory bus traffic, as shown in Figure 1, two SRAM buffers are used during this process and for transfering the sorted results to the next stage, the R_Fill module.

In order to reduce the memory bandwidth, the edge data generated by the Tessellation stage is represented in a 16-bit fixed-point format. Some of the active-edge data provided to the R_fill module are represented in the same 16-bit fixed-point formant. Throughout our design, however, we use 24-bit precision arithmetic units for most floating point operations. Figure 3 summarizes the operation of the Gen/Sort module. The performance of the R_fill module depend on the number of edges created by the Tesselation stage.

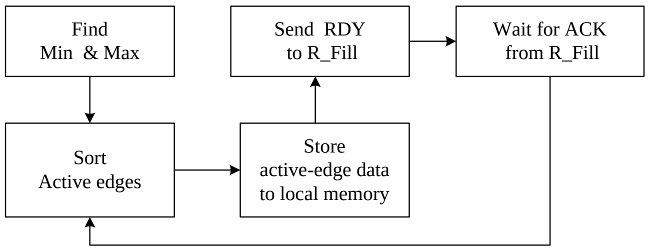


Figure 3. Operation of Gen/Sort

2. 2 The R_fill Module

The R_fill module decides whether to paint each pixel or not and sends the coverage values to the PixelPipe module. The active-edge data from the Gen/Sort module are analyzed for making the decision.

From the active-edge data, the R_fill module obtains the winding value which is used to generate coverage value. We use the 8-queens algorithm for better image qulity and generate eight sample points. After judging whether the active edges are located to the left of the sample point, the winding value is calculated from the direction values of the active edges. Then, the coverage value is determined from the calculated winding value and the fill rule set by the user.

When the winding value is calculated, not all the active edges existing to the left of the current pixel should be considered. Only those which are across the current pixel are considered. In order to implement this scheme, the R_fill module use the falg information set by the Gen/Sort module, which indicates the last pixel of the active edge. The module also keeps the winding value of the previous pixel.

All the results generated by the R_fill module, the position (x and y) and the coverage values, are pushed into the queue between the R_fill and PixelPipe modules. The performance of the R_fill module also depend on the number of edges created in the Tesselation stage.

2. 3 The PixelPipe Module

The PixelPipe module determins an actual color value for each pixel by using the coverage value provide by the R_fill module. Masking, Painting, Blending, and Anti-aliasing are main functions of this module.

In order to implement the Masking effect, the module reads the alpha value from the mask image and multiplies the alpha value by the coverage value. When the alpha value is 0, the designated part of the shape will not appear. The Blending effect is implemented using the color values previously stored in the frame buffer and the new color value to be painted. These two color values have their own

RGB and Alpha values. For Painting, the color value set by the user is applied. After Masking, Blending, and Painting, the final color value is determined in Antialiasing.

Since the PixelPipe module performs lots of arithmetic operations, one of the main objectives in our design is to develop efficient schemes for sharing the functional units among these operations. Figure 4 illustrates the operation of the PixelPipe module. The performance of the PixelPipe module is affected by the size of the display device and the size of the shapes to be drawn.

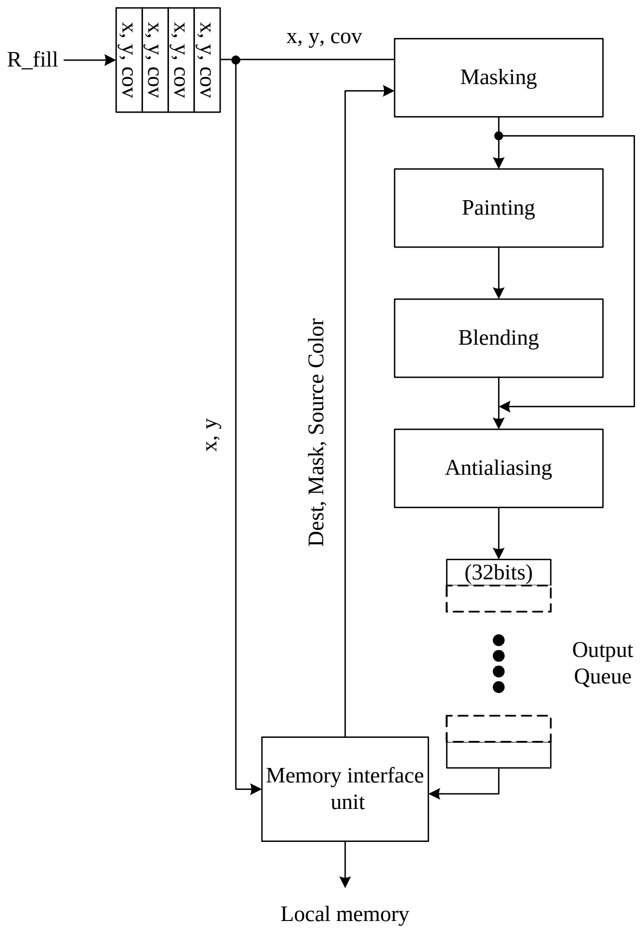


Figure 4. Operation of PixelPipe

2. 4 The Arb/IF Module

The Arb/IF module arbitrates the read and write requests from other three modules as depicted in Figure 1. The centralized arbitration scheme is used. The module also provides the interface to the bus, which is the AXI bus currently, and functions as a bus bridge. This module depends on the rest of the system.

3. Evaluation

This section presents our test and evaluation methods and the results that we observe.

3. 1 Evaluation Environment

Figure 5 describes how we test and evaluate our design. We have tested and verified the functionality of our design using Cadence NC-verilog simulator with the test vector collected from the RI. The frame buffer data generated by the accelerator has also been displayed by software.

The designed accelerator has been synthesized using Synopsys DesignCompiler with a 0.18um CMOS standard cell library. Synopsys DesignWare library has also been used in our simulation and synthesis.

Post-synthesis simulation has also been performed using Cadence NC-verilog simulator. It has been observed that our current design operates correctly with 62.5MHz clock. .

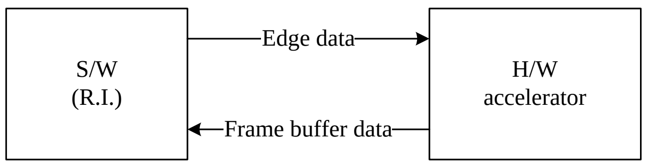


Figure 5. Evaluation Environment

3. 2 Results

The designed accelerator occupies approximately 0.40 million gates, excluding three SRAM modules. Each of the 4KB SRAMs costs about 17,500 gates. The cost of Arb/IF unit is significant and depends on the rest of the system.

The performance of the accelerator depends heavily on the complexity of the image. Figure 6 shows an example motion image considered in the test. In Figure 6, only one flower is moving. The designed accelerator can process the images of size 640x480 pixels with moderate complexity (with blending, and with less than 100 active edges per scan line, etc) at the rate of 30 frames per second.

The proposed accelerator can also process the pictures of size 1024x768 pixels with the same (moderate) complexity at the rate of 18 frames per second.

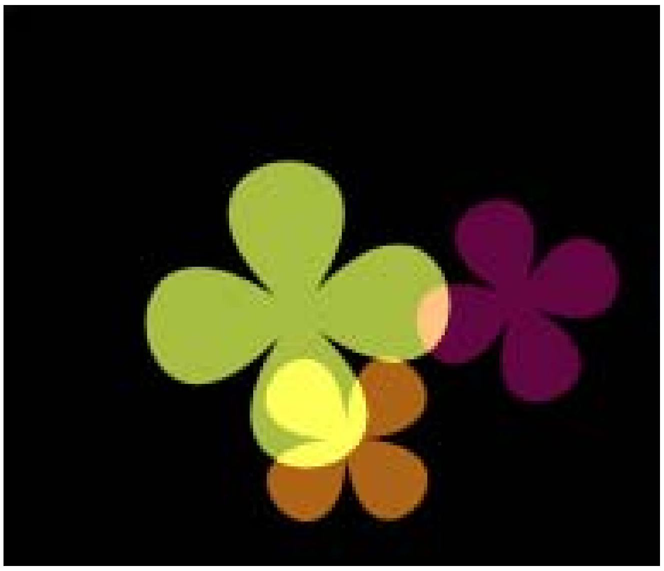


Figure 6. An example motion image used (640x480)

4. Conclusion

We have designed a prototype hardware accelerator for 2D vector graphics applications based on the OpenVG standard. One of our main design objectives is to develop schemes for exploiting efficiently limited resources such as memory bandwidth. The designed accelerator can process the images of 640x240 pixels with moderate complexity at the rate of 30 frames per second.

Our current design has been designed as an initial reference design and is planned to be tested in a multiprocessor SoC. Now we are working on including the Tessellation stage into the hardware and improving our current design. In this work, we have learned that there are a wide design space for optimization. The planned optimizations include optimizations in the algorithmic, arithmetic, and architectural levels.

Acknowledgement

Authors wish to acknowledge the financial support of ADChips Inc., Korea and the CAD tool support of IDEC (IC Design Education Center), Korea.

References

[1] S.-Y. Lee, S. Kim, J. Chung, and B.-U. Choi. “Scalable Vector Graphics (OpenVG) for Creating Animation Image in Embedded Systems,” *Proc. KES2007*, pp. 99-108, 2007.

[2] D. Johansson and M. Socha. *OpenVG for Mobile Equipment*, MS Thesis, Dept. of Computer Science, Lund Institute of Technology, 2006

[3] R. Huang and S.-I. Chae. “Designing an OpenVG accelerator: algorithms and guidelines,” *Proc. ICCCE’06*, pp. 555-560, 2006.

[4] Khronous Group std., *OpenVG*, Khronous Group Standard for Vector Graphics Accelerations, www. khronous.org., 2005.

[5] http://www.khronos.org/openvg/