TWO-LEVEL PIPELINED SYSTOLIC ARRAY GRAPHICS ENGINE*

J.A.K.S. Jayasinghe, G. Karagiannis, F. Moelaert El-Hadidy,
O.E. Herrmann and J. Smit

University of Twente, Laboratory for Network Theory
P.O. Box 217, 7500 AE Enschede, The Netherlands

Abstract

Improvement of the interaction speed of raster graphics systems is an important topic in computer graphics research. A two-level pipelined systolic array graphics engine will be described, which can improve the interaction speed and image quality for high resolution displays.

1 Introduction

In the past decade, raster graphics systems became popular over vector graphics systems due to their high image quality. The frame buffer in the raster graphics systems has been identified as the major bottle-neck for real-time interaction [5]. A VLSI *systolic array graphics* (SAG) engine called *Super Buffer* (only capable of constant shading) was first introduced in 1985 [1] to replace the frame buffer by a processor array. More powerful SAG engines capable of Gouraud shading were later introduced [2],[3] which use 16 bit fixed-point arithmetic. The main advantage of the SAG engine is its smaller overall system size compared to other graphics systems [1], and its potential for better interaction speed [4]. As the maximum operating speed of a systolic array is determined by the delay of the most complex operation, the maximum operating speed of an SAG engine tends to reduce as more complex functions are introduced to improve the image quality. Attempts to improve the speed by fast functional units (like carry look-ahead adders instead of simple carry-ripple adders) become unpractical due to their large silicon area and large number of processing elements (PEs) in an SAG engine. The purpose of this paper is to report a VLSI architecture of an SAG engine built from pipelined functional units which can generate realistic images interactively for high resolution displays.

In the next section we introduce an advanced instruction set and an architecture of an SAG engine. We derive two architectures built from pipelined functional units to improve the operating speed. Some details of a prototype are also presented. Finally, some conclusions are drawn.

*The work described in this paper is part of a project for developing a graphics workstation. The project is sponsored by the Dutch Research Foundation (STW), under contract CWI77.1249, carried out by University of Twente, Enschede, The Netherlands, and the Center for Mathematics and Computer Science, Amsterdam, The Netherlands.

2 An Advanced SAG engine

2.1 The Instruction Set

Shading determines the realism of computer generated images. Conventional shading methods providing realistic images need huge processing power. Therefore, less realistic shading methods (like Constant shading) have been used in interactive applications. We have found that Phong shading, which can generate quite realistic images, can be approximated by second order interpolation. This approach dramatically reduces the computational power requirements, hence it becomes feasible for interactive computer graphics. A generalized interpolation scheme where one can perform zero, first or second order interpolation with discontinuities in intensity and/or derivatives of the intensity provides a unified approach to support several shading methods. Table 1 shows an instruction set for this approach which can be executed on an SAG engine. The *EVAL*(...) instructions* perform the interpolation and the *SET*(...) instructions* support the discontinuities. The *REF() instruction* sends a new pixel to the display. When less number of objects have to be scan-converted, the empty slots between the *REF() instructions* are filled by *NOP() instructions*. The *DIS(...) instruction* disables the accumulation of intensities. The *ACC_M() instruction* toggles the accumulation of negative intensities. Figure 1 shows two shading examples.

2.2 The Architecture

The SAG engine consists of a one-dimensional array of identical PEs. Processing and storage of each pixel column is done by a single PE, such that adjacent pixel columns are taken care of by adjacent PEs, minimizing the communication bottle-necks. During each cycle of the SAG engine, the PE containing the *REF() instruction* sends the pixel value in register P (see Figure 2) to the display. Other PEs perform operations according to the instructions in them (the controller decodes the instruction to enable proper functional units in the PE). At the end of the cycle all but the last PE pass instructions to their neighbors and the first PE receives a new instruction. Before the instructions are sent to the neighboring PEs, the data and op-code associated with them are modified. The registers A,B,C store the intensity I, its first derivative DI and its second derivative DDI. Data is

represented by 36 bit fixed-point numbers[1] for the interpolation and discontinuity corrections. The processor addresses X and DX are represented by 12 bit integers. Due to bandwidth limitations, the processor addresses (i.e., X and DX), DDI, DI and I are sent into the array in consecutive time slots in the given order. For fault tolerance reasons, the processor location X is identified by decrementing the address X at each PE and detecting whether its value is zero or not. Processor locations $X+DX, X+2DX, X+3DX, ...$ can also be similarly identified, by substituting DX to X whenever X is zero. Faulty PEs are bypassed by disabling the decrementation of X and bypassing the instructions. As the data associated with instructions is sent in different time-slots, the processor address decrementation, intensity interpolation and intensity accumulation can be done on the same adder. Then, the condition $(X = 0)$ can be detected by monitoring the carry out of the adder at the 12^{th} bit (i.e., C_{12} in Figure 2) when X is represented by the low-significant 12 bits. The leftmost multiplexer provides a -1 (in 12-bit format) on one input of the adder to decrement X. The rightmost multiplexers select the correct output data and op-codes.

For proper operation, only one *REF() instruction* can reside in the SAG engine at any time. As only a single row of pixels can be stored in an SAG engine, pixel values of each pixel row are calculated in real-time by sending the instructions in between the refresh instructions. As the video stream is generated on the fly, the speed of the clock is determined by the display resolution.

3 Two-level Pipelining of the Advanced SAG engine

36 bit arithmetic primarily determines the maximum operating speed and hence the maximum display resolution in real-time applications. For an optimized carry-ripple adder in a $1.6\mu m$ CMOS technology, the estimated cycle time is about 100ns, which is far below the needs of high resolution displays[2]. Due to the large area of faster adders (like carry look-ahead), it is not feasible to use them in this engine as the number of PEs needed is equal to the number of pixel columns. Pipelined carry-ripple adders are shown to be practical. Systolic arrays built from pipelined functional units have been referred to as *two-level pipelined systolic arrays* in the literature and a formal approach to derive such arrays from word-level systolic arrays has been reported [7]. Unfortunately, this formal approach fails to produce correct SAG engine architectures due to time-varying behavior of the PEs (i.e., PEs perform different functions in different time slots.) and shift-variance of the data passing through the array (i.e., output data of a PE depend on its location in the array.). We have developed a formal approach for two-level pipelining of systolic arrays subject to time-varying and shift-variance behaviors which is reported in [6].

[1] At least $m+2n$ bit fixed-point representation is needed to prevent visible quantization errors, where 2^m and 2^n are intensity and horizontal display resolutions.

[2] 1024 x 1024 display refreshed at 50Hz needs a 12ns cycle time.

3.1 Our Formal Approach

For the completeness of this paper, we describe the formal approach we reported in [6] briefly without any proofs. The original systolic array is represented at bit-level by a finite, vertex-weighted, edge-weighted, directed graph G which is constructed by replicating the graph of a PE, $G_{PE} = \langle V_{PE}, V'_{PE}, E_{PE}, d_{V_{PE}}, d_{V'_{PE}}, w_{PE} \rangle$ and connecting corresponding vertices by weighted edges indicating the earliest communicating time slot $i (i = 0, 1, 2, ...)$ between PEs as the entire array is built from identical PEs. In G_{PE}, vertices V_{PE} and V'_{PE} denote the bit-level functional units and storage. Each edge $e \in E_{PE}$ and weight of it $w_{PE}(e)$ denotes the communication between nodes and earliest communication time slot for all legal combinations of instructions respectively. Vertex weight $d(v), v \in V_{PE}$ denotes the numerical propagation delay. Each vertex $v' \in V'_{PE}$ is multiple weighted by $d(v'_{a \to b})$ for each input edge a and output edge b connected to node v', when there is a direct information flow or else undefined. The weight $d(v'_{a \to b})$ indicates the minimum latency (which is under control of instructions) between the information flow from edge a to b through the storage node v'.

In order to improve the clock speed, pipeline registers are added to critical paths in G to meet the given speed requirements, and then some more additional pipeline registers are added to some other edges and/or latencies at some storage vertices are changed to keep the logical behavior of the system intact.

Two-level Pipelining Theorem: If a two-level pipelined design is obtained by adding pipeline registers to some edges and/or changing the latencies of storage nodes in G, the logical behavior of the system will be kept intact if the differences in latencies through all pairs of paths between any two nodes are equal in the original and re-timed (i.e., two-level pipelined) graphs.

3.2 Two-level Pipelined Designs

The architecture in Figure 2 can be converted into two-level pipelined designs as outlined below:

- Construct the graph.
- Identify the critical paths and add pipeline registers into them to meet the given speed requirements.
- Apply the two-level pipelining theorem and get a functionally correct and feasible design.

Due to the difference in bit requirements for processor addresses (i.e., X, DX) and intensity data (i.e., I, DI, DDI) we get two groups of two-level pipelined architectures:

Group-X There are no pipeline registers in the functional units where the addresses are updated. The latency from the input-ports to output-ports of each PE is unchanged.

Group-Y Pipeline registers are inserted into the functional units where the address are updated. As the address related decisions are delayed, the latency from the input-

ports to output-ports of each PE is increased by the same value.

In the case of the architecture in Figure 2, the speed improvements of Group-X architectures are limited to 3 folds, since the address and intensity data are represented by 12 and 36 bits respectively. Figure 3 shows a Group-X architecture. The adder is divided into 3 blocks of 12 bits by pipeline registers. As the input data to the adder must be in skewed format, pipeline registers are placed in the control signal paths of registers. Due to the skewed output data from the adder, pipeline registers are placed at the rightmost multiplexer.

Figure 4 shows a Group-Y architecture where the longest carry-ripple path is limited to 5 bits, hence a factor $\frac{36}{5}$ speed up. In this architecture, 8 pipeline registers are placed into the 36 bit adder carry-ripple path to reduce the propagation delays. The first 12 bits, on which the address X is updated, are divided into 3,5 and 4-bit blocks due to the instruction decoding delays. The remaining 24 bits are divided into 3,5,4,3,5 and 4-bit blocks to get a regular design. As the control signals for the rightmost multiplexers are generated in conjunction with the carry signal C_{12} from the adder, the (data) inputs to these multiplexers are delayed by pipeline registers hence the latency from the input-ports to the output-ports of each PE is increased. Pipeline registers on the control signals of registers A,B,C,P and leftmost multiplexer are due to the skewed input data required at the adder inputs. Furthermore, pipeline registers are introduced on the rightmost multiplexer control signals due to the skewed output data from the adder. Pipeline registers on the pixel stream are placed due to the increased latency between the input-ports and output-ports of each PE.

Due to the superior speed of Group-Y architectures, we decided to implement the architecture given in Figure 4. In a university environment, a 9-PE prototype consisting of 85,000 minimum feature size transistors in a 1.6 μm CMOS process has been realized. The area overhead of the pipelined registers is approximately 25%. The hardware description language MoDL [8] has been used for the functional verification, and symbolic layout design system CAMELEON has been used for the cell design. According to extensive simulation results, it is capable of operating with a 12 $\sim$ 15ns cycle time, which has to be verified when the chips are back from the foundry at the end of January 1990. More details are given in Table 2. We estimate that 50 $\sim$ 60-PEs can be implemented on the same die by improving the pitch-matching of leaf-cells and using Domino Logic without any speed reductions.

4 Conclusions

A silicon implementation of a Two-level pipelined SAG engine supporting an advanced instructions set is reported. The advantage of the two-level pipelining is that it can provide a complex functionality at high pixel rates, which is difficult to achieve by other means using less silicon area. As computer graphics users have a great desire for high image quality, high interaction speed and high resolution, we think that two-level pipelined SAG engines will be a break-through for real-time computer graphics.

Acknowledgements

Thanks to Jos Huisken and other members of the "VLSI Design Group" at Philips Research Laboratories for their numerous help and for allowing us to use their design system and fabrication process. The members of "Interactive Systems Group", CWI, Amsterdam are also acknowledged for the discussions during the specification development phase of the advanced SAG engine.

References

[1] N. Gharachorloo and C. Pottle, *SUPER BUFFER: A Systolic VLSI Graphics Engine for Real Time Raster Image Generation*, Proceedings of 1985 Chapel Hill Conference on VLSI, Computer Science Press, 1985, pp. 285-305.

[2] N. Gharachorloo, S. Gupta, E. Hokenek, P. Balasubramanian, B. Bogholtz, C. Mathieu and C. Zoulas, *Subnanosecond Pixel Rendering with Million Transistor Chips,* Proceedings of SIGGRAPH 88, August 1988, pp. 41-49.

[3] T. Nishizawa, T. Ohgi, K. Nagatomi, H. Kamiyama and K. Maenobu, *A Hidden Surface Processor for 3-Dimension Graphics*, ISSCC 88, February 1988, pp. 166-167.

[4] J.A.K.S. Jayasinghe, A.A.M. Kuijk and L. Spaanenburg, *A Display Controller for a Structured Frame Store System*, Advances in Graphics Hardware III, Springer-Verlag, 1989.

[5] P.J.W. ten Hagen, A.A.M. Kuijk and T. Triekens, *Display Architecture for VLSI-based Graphics Workstation*, Advances in Graphics Hardware I, Springer-Verlag, 1987.

[6] J.A.K.S. Jayasinghe and O.E. Herrmann, *Two-level Pipelining of Systolic Array Graphics Engines*, To be published in Advances in Graphics Hardware IV, Springer-Verlag, 1990.

[7] H.T. Kung and M.S. Lam, *Fault-Tolerance and Two-level Pipelining in VLSI Systolic Arrays*, 1984 Conference on Advanced Research in VLSI, M.I.T., 1984, pp. 74-83.

[8] O. Herrmann, et.al., *MoDL : Modeling and Design Language*, University of Twente, 1988.

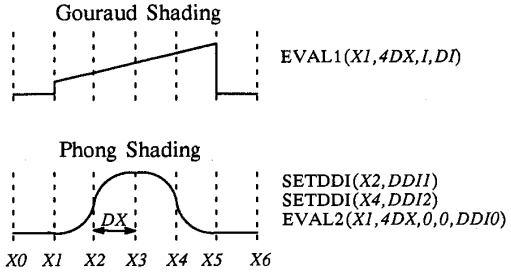


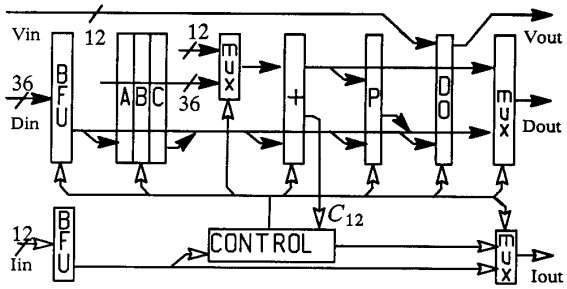
Figure 1: Two Shading Examples

REF() Send the local pixel storage to the display and reset the processor.
EVAL0(X, DX, I) *EVAL1(X, DX, DI, I)* *EVAL2(X, DX, DDI, DI, I)* Interpolate and accumulate the intensity between processors $X+1, X+DX+1$ until the next *REF()*. Zero, first and second order interpolation is done by the *EVAL0(...),EVAL1(...)* and *EVAL2(...)* instructions respectively.
SETPI(X, DX, I) *SETPDI(X, DX, DI)* *SETPDDI(X, DX, DDI)* Correct the periodic discontinuities during the next interpolation at processors *DX* apart, starting from $X+1$. The intensity, its first derivative and its second derivative is corrected by the *SETPI(...),SETPDI(...)* and *SETPDDI(...)* instructions respectively.
SETI(X, I) *SETDI(X, DI)* *SETDDI(X, DDI)* Correct the discontinuity during the next interpolation at processor $X+1$. The intensity, its first derivative and its second derivative is corrected by the *SETI(...),SETDI(...)* and *SETDDI(...)* instructions respectively.
DIS(X, DX) Disable the accumulation between processors X+1, X+DX+1.
NOP() Do nothing.
ACC_M() Toggle the accumulation of negative intensities.

Table 1: Our Advanced Instruction Set

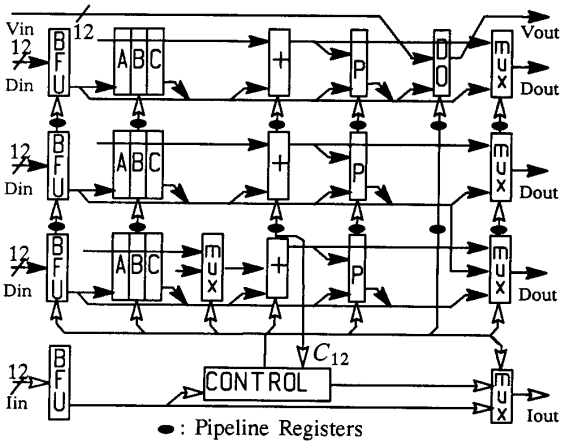
Feature	Prototype	Optimized Design
Pixel rate	66 ~ 83 MHz	66 ~ 83 MHz
Transistor count	85 K	190 K ~ 230 K
PE count	9	50 ~ 60
Process	1.6μm CMOS	1.6μm CMOS
Chip size	10.2 x 11.4mm^2	10.2 x 11.4mm^2
Supply	5V	5V
Dissipation	< 5W at 83 MHz	< 5W at 83 MHz
Package	144 PGA	144 PGA

Table 2: Main Features of the Prototype (in full CMOS) and Expected Features of an Optimized Design (in Domino Logic)



A,B,C:Registers MUX:Multiplexer P :Pixel Storage
BUF :Buffer OD :Output Driver

Figure 2: Architecture of a PE



●: Pipeline Registers

Figure 3: A Group-X Architecture

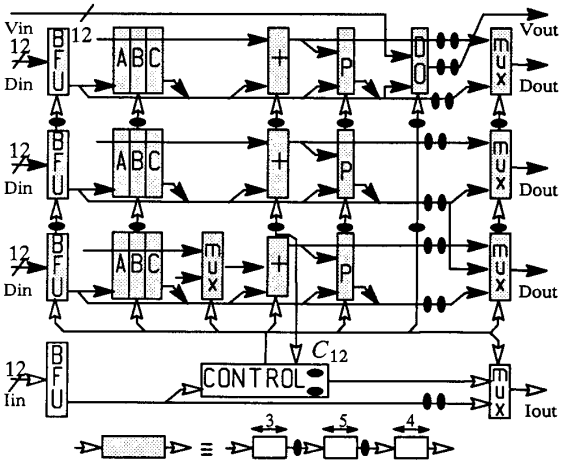


Figure 4: A Group-Y Architecture