Two-Level Pipelined Systolic Array Graphics Engine

J. A. K. S. Jayasinghe, *Member, IEEE*, F. Moelaert El-Hadidy, G. Karagiannis, Otto E. Herrmann, *Member, IEEE*, and J. Smit

Abstract—**Simultaneous improvement of interaction speed and image quality of raster graphics systems is an important topic in computer graphics research. Due to huge processing power requirements, realistic shading methods (like Phong shading) have been used only in noninteractive applications whereas less realistic shading methods (like Constant shading) have been used in current interactive applications. A design of a systolic array graphics engine is described which generates high-quality Phong-shaded images in real time. Under the area and speed limitations of the current IC technologies, the required speed is achieved using pipelined functional units. A prototype containing nine processing elements was fabricated in a 1.6-μm CMOS technology.**

I. Introduction

In THE past decade, raster graphics systems have become more popular than vector graphics systems because of their better image quality. The frame buffer in the raster graphics systems has been identified as the major bottleneck for real-time interaction [1] due to its insufficient bandwidth. A VLSI systolic array graphics (SAG) engine called Super Buffer (only capable of Constant shading) was first introduced in 1985 [2] to replace the frame buffer by a processor array. More powerful SAG engines capable of Gouraud shading were introduced later [3], [4] which use 16-b fixed-point arithmetic. The main advantage of the SAG engine is its smaller overall system size compared to other graphics systems [2], and its potential for better interaction speed [5]. 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 using faster functional units (like carry lookahead adders instead of simple carry-ripple adders) lead to larger silicon area. Since a large number of processing elements (PE's) is needed in an SAG engine, this solution is impractical. The purpose of this paper is to report a VLSI design of an advanced SAG engine built from pipelined functional units which can generate realistic images interactively for high-resolution displays.

This paper is organized as follows. In the next section, we introduce a structured frame store system as an environment for the advanced SAG engine. In Section III, we present the principles and architecture of the advanced SAG engine. We introduce pipelined functional units into this SAG engine to meet the performance requirements. This is done by the formal approach presented in Section IV. Next, two architectures built from pipelined functional units are described in Section V. In Section VI, some details of a prototype are presented. Finally, some conclusions are drawn.

Manuscript received July 27, 1990; revised November 1, 1990. This work is part of a project for developing a graphics workstation. The project is supported by the Dutch Research Foundation (STW) under Contract CWI77.1249, carried out by the University of Twente, Enschede, The Netherlands, and the Center for Mathematics and Computer Science, Amsterdam, The Netherlands.

J. A. K. S. Jayasinghe is with the Laboratory for Network Theory, University of Twente, 7500 AE Enschede, The Netherlands, on leave from the Department of Electronics Engineering, University of Moratuwa, Moratuwa, Sri Lanka.

F. M. El-Hadidy, G. Karagiannis, O. E. Herrmann, and J. Smit are with the Laboratory for Network Theory, University of Twente, 7500 AE Enschede, The Netherlands.

IEEE Log Number 9041479.

II. A Structured Frame Store System

For the sake of completeness, a brief description of the structured frame store system is presented that incorporates the advanced SAG engine. The low-level display file (LDF) in Fig. 1 contains nonoverlapping facets (patterns) describing the visible image. These patterns are shaded in real time while being displayed on the screen. Since a large number of small patterns can occur in a typical scene, high bandwidth is required to access the LDF. This has been achieved by parallel accesses of the LDF by multiple pattern loaders (PL's). The patterns that are contributing to the next pixel row are stored in the active pattern store (APS). There are two sets of systolic arrays that work independently. One takes care of the incremental calculations along the pixel-column direction, the systolic array preprocessing (SAP) engine, and the other along the pixel-row direction, the SAG engine. The SAP engine calculates the intersections of edges with the current pixel row, the color values at the leftmost edge, incremental color values along the pixel-row direction, etc. These data are sent into the scan-line command buffer (SCB) as instructions and data for the SAG engine. In the SAG engine, shading is done by incremental calculations. The output of the SAG engine is directly used to refresh the display.

This architecture offers attractive features such as expandability, linear performance improvement with increased hardware, efficient implementation in VLSI, and the ability to generate high-quality pictures with a good interactive behavior.

III. The Advanced SAG Engine

In this section, we introduce an advanced instruction set together with the corresponding architecture of an SAG engine. Before that, some notes on shading are made to

0018-9200/91/0300-0229$01.00 ©1991 IEEE

TABLE I
OUR ADVANCED INSTRUCTION SET

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

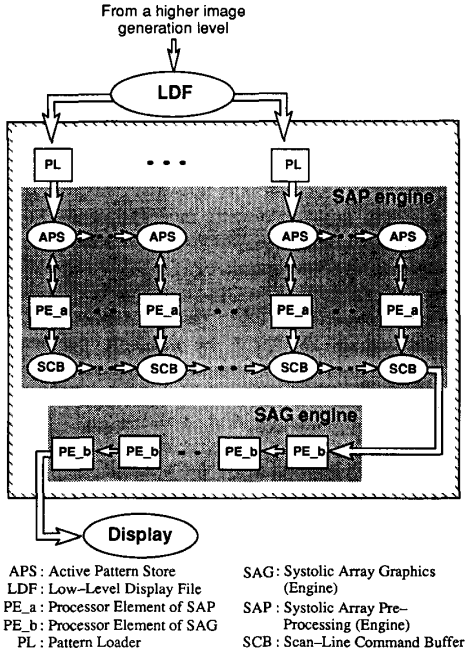


Fig. 1. Architecture of the structured frame store system.

highlight the power of the new instruction set and the architecture.

A. Some Notes on Shading

Shading enhances the realism of computer-generated images. There are several well-known shading techniques such as Constant shading, Gouraud shading, and Phong shading, mentioned in order of increasing complexity [6]. Constant shading calculates a single intensity value for an entire facet. Gouraud shading linearly interpolates the intensity along the edges and between the edges. Phong shading interpolates some vectors and calculates a unit vector dot product raised to a power. The image quality that Phong shading offers is superior to Gouraud shading but greatly increases the cost of computation. We conclude that shading methods providing realistic images need huge processing power. In order to achieve the required interaction speed, less realistic shading methods (like Constant shading) have been used in current interactive applications, whereas more realistic shading methods (like Phong shading) have been used only in noninteractive applications.

We have found that Phong shading can be approximated by second-order intensity interpolation with changes to the second derivative of the intensity at some strategic points during interpolation [5]. This approach dramatically reduces the computational power requirements. It introduces no visible degradation in image quality, because it calculates the intensity values to the same accuracy as the conventional Phong shading produces. A 10×10-pixel facet can be shaded using a factor of 10 less processing power, whereas for a 100×100-pixel facet the saving is a factor of 25. Though Phong shading was considered to be computationally too expensive for real-time applications, the simplicity of the second-order interpolation scheme makes real-time Phong shading not only feasible but also faster than Gouraud shading. When Phong shading is used, one can increase the sizes of facets without losing the quality of the image. This reduces the number of facets needed to describe an image. The fewer the number of facets, the smaller the amount of processing power required. Our estimate shows that by making the facets more than a factor of 4 larger, we can make Phong shading faster than Gouraud shading.

B. The Instruction Set

A generalized interpolation scheme, which performs 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 I shows our instruction set for this approach which can be executed on an SAG engine. The *EVAL*$*$($X, DX, \cdots$) instructions perform the intensity interpolation between pixel locations $X+1$ and $X+DX+1$, and the *SET*$*$($\cdots$) instructions are used to set or compensate for the discontinuities at the required locations. The *REF*( ) instruction is sent

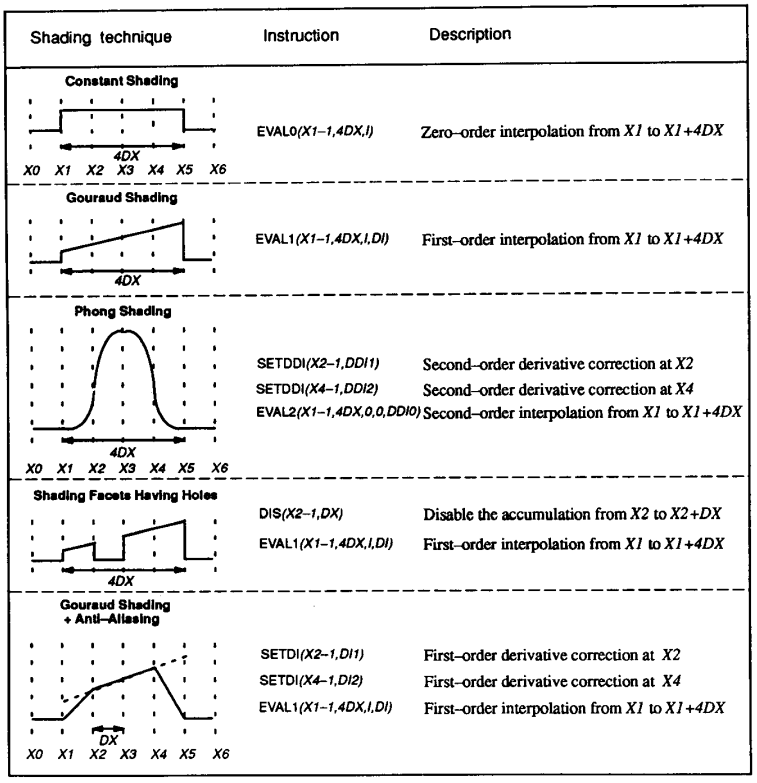


Fig. 2. Implementing some shading algorithms using our instruction set.

into the SAG engine in synchronism with the display refreshing. It sends the pixel value of the current PE it resides in to the display and resets each PE as it travels through the processor array. The instructions required to shade the current pixel row are sent into the SAG engine between the *REF*( ) instructions, due for the previous and current pixel rows. When fewer objects (than the capacity of the SAG engine) have to be displayed, the empty slots between the *REF*( ) instructions are filled by *NOP*( ) instructions. The *DIS*($\cdots$) instruction disables the accumulation of intensities. Facets having holes can be shaded efficiently by sending this instruction before an *EVAL* $*(\cdots)$ instruction. The *ACC_M*( ) instruction toggles the accumulation of negative intensities. Using this instruction, one can eliminate the back facing parts of a facet. Fig. 2 shows the implementation of some shading algorithms using our instruction set.

C. The Architecture

In an SAG engine, the pixel storage, which is limited to a pixel row, is distributed over the one-dimensional systolic array built from identical PE's. Processing and storage of pixels in each pixel column is done sequentially by a single PE, such that adjacent pixel columns are taken care of by adjacent PE's, minimizing communication requirements. During each processor cycle of our SAG engine, the PE containing the *REF*( ) instruction sends the pixel value in register *P* (see Fig. 3) to the display. As the *REF*( ) instruction is passed in synchronism with the display refreshing, the video stream is generated on the fly and the display resolution determines the clock frequency. Other PE's perform operations according to their current instructions. The controller decodes the instruction to enable proper functional units in the PE. At the end of the processor cycle all but the last PE pass instructions to their neighbors and the first PE receives a new instruction. Before the instructions and data are sent to the neighboring PE's, they are modified according to a known criterion. The registers *A*, *B*, and *C* store the intensity *I*, its first derivative *DI*, and its second derivative *DDI*. Data are represented by 36-b fixed-point numbers[1] for interpolations and discontinuity corrections. The processor addresses, *X* and *DX*, are represented by 12-b integers. The

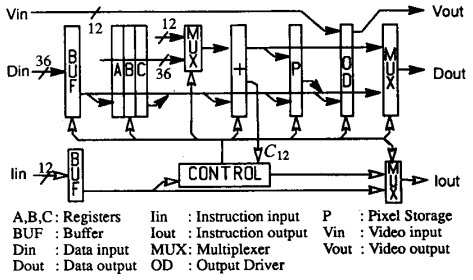


Fig. 3. Architecture of a PE in the advanced SAG engine.

[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.

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 $X+DX, X+2DX, X+3DX, \cdots$ can also be similarly identified, by substituting X by DX whenever X is zero. Faulty PE's are bypassed by disabling the decrementing of X and bypassing the instructions and data. As the data associated with instructions are sent in different time slots, the processor address decrementation, intensity interpolation, and intensity accumulation can be done on the same adder. The condition $X=0$ can be detected by monitoring the carry-out of the adder at the 12th bit (i.e., C_{12} in Fig. 3) when X is represented by the lowest significant 12 bits. The leftmost multiplexer provides the data for decrementing X. The rightmost multiplexers select the correct output data and op codes depending on the decisions related to the processor addresses.

IV. A Formal Approach for Two-Level Pipelining

Pipelined functional units are shown to be very attractive in achieving the necessary performance requirements under the speed and area limitations of the current IC technologies. Systolic arrays built from pipelined functional units have been referred to as two-level pipelined systolic arrays in the literature [7]. When the behavior of a systolic array is time invariant (i.e., each PE performs the same function in every cycle) and shift invariant (i.e., the output data of a PE are independent of the PE location in the array), it can be converted into a two-level pipelined design by using the formal methodology presented in [7]. Due to the time-variance and shift-variance behavior of the advanced SAG engine, this formal methodology fails to produce functionally correct designs [8]. Therefore, we use the following graph-theoretic approach, which can be used for two-level pipelining of systolic arrays irrespective of their behavior.

In our approach, the original systolic array is represented at bit level by a finite, vertex-weighted, edge-weighted, directed graph $G=\langle V, V', E, d_V, d_{V'}, w_E\rangle$ (from now on, for simplicity, we say graph) where $V \cup V'$ and E are the set of vertices and the set of edges, respectively. The functions d_V, $d_{V'}$, and w_E represent the weights defined on vertices in V, vertices in V', and edges in E, respectively. The graph G is constructed by replicating the graph of a PE, $G_{\mathrm{PE}} = \langle V_{\mathrm{PE}}, V'_{\mathrm{PE}}, E_{\mathrm{PE}}, d_{V_{\mathrm{PE}}}, d_{V'_{\mathrm{PE}}}, w_{E_{\mathrm{PE}}}\rangle$ and connecting vertices according to the communication between PE's as the entire systolic array is built from identical PE's.

In order to construct the graph G_{PE}, the bit-level functional units are represented by vertices V_{PE} and bit-level storage by vertices V'_{PE}. The communication between vertices is denoted by the edges and for each edge $e \in E_{\mathrm{PE}}$, $w(e)$ denotes the earliest communication time slot for all legal combinations of instructions. The edges which communicate in the ith ($i=0,1,2,\cdots$) time slot are weighted by i. As any vertex $v \in V_{\mathrm{PE}}$ represents a functional unit, the data sent on an output edge of v are dependent on the data supplied on several input edges of v. Though the propagation delay from each input edge to an output edge is different, for simplicity, we assume that the propagation delay for each vertex is equal to the worst propagation delay of it. Hence, we weigh each vertex $v \in V_{\mathrm{PE}}$ by $d(v)$, the maximum numerical propagation delay of the functional unit represented by the vertex v. In general, storage vertices represent multiport registers. Therefore, the data coming from an input edge are stored and passed to a selected output edge. The output edge is selected by another input edge representing a control signal. Therefore, we weigh each vertex $v' \in V'_{\mathrm{PE}}$ by $d'(v', v_1, v_2)$ for all vertices v_1 and v_2 such that vertex v_2 receives data from vertex v_1 through vertex v'. The quantity $d'(v', v_1, v_2)$ indicates the minimum latency (which is under the control of instructions) through the storage vertex. If the data supplied to vertex v_2 from vertex v' have no dependency on the data supplied to vertex v' from vertex v_1, then $d'(v', v_1, v_2)$ is undefined.

The maximum clock speed of the circuit is determined by the propagation delays of the critical path(s). A critical path in our graph G can be identified as a directed path activated in the same time slot such that the sum of the vertex weights on that path is maximum over the entire graph. If a critical path goes through a storage vertex, the critical path is terminated at the storage vertex whenever the vertex weight corresponding to that path is nonzero, as the vertex weights of storage vertices denote the latency whereas the weights of other vertices denote the propagation delay. In order to improve the clock speed, the graph is retimed. In retiming, pipeline registers are added to all critical paths to meet the given speed requirements, and then additional pipeline registers are added to other edges and/or latencies at storage vertices are changed such that the conditions of the following theorem are satisfied.

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 vertices of G, the logical behavior of the system will be kept intact if the differences in latencies through any pair of paths between any two vertices are equal in the original and retimed graphs when the weights of the storage vertices corresponding to these paths are not undefined.

Proof: Let m and n be two vertices and p_1 and p_2 be two paths from m to n. Assume path p_1 is activated on $i_{p1_1}, i_{p1_2}, \cdots, i_{p1_{N1}}$ time slots ($i_{p1_1} < i_{p1_2} \cdots < i_{p1_{N1}}$) and p_2 is activated on $i_{p2_1}, i_{p2_2}, \cdots, i_{p2_{N2}}$ time slots ($i_{p2_1} < i_{p2_2} \cdots < i_{p2_{N2}}$). We can make this assumption iff this path does not contain edges connected to a storage vertex such that the vertex weight is undefined. If we introduce a latency of k cycles in path p_1, by introducing pipeline stages and/or changing storage latencies, vertex m gets the data from vertex n through path p_1 in time slot $i_{p1_{N1}}+k$. In the original graph, vertex m gets the data from vertex n through paths p_1 and p_2 in time slots $i_{p1_{N1}}$ and $i_{p2_{N2}}$, respectively. In order to keep the logical behavior of the graph intact, we must get data through path p_2 in time slot $i_{p2_{N2}}+k$. Now, we can see that the difference in latency through path p_1 and p_2 is equal to $i_{p1_{N1}} - i_{p2_{N2}}$ cycles in both original and retimed graphs. This argument, when applied to all paths between any vertex pair, proves the theorem. □

As any circuit in a graph can be described in terms of a set of linearly independent circuits, it is enough to apply the two-level pipelining theorem to a set of linearly independent circuits of G. This produces a set of equations that contains more variables than the number of equations. Therefore, feasible solutions can be found by linear integer programming, for example, by minimizing the total register count.

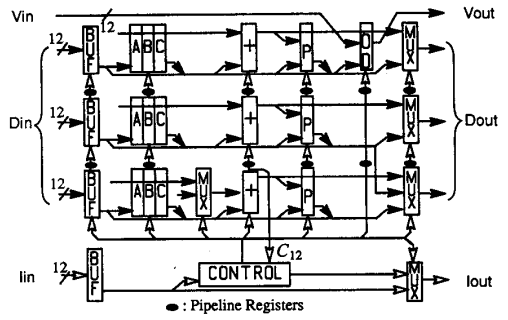


Fig. 4. A Group-X architecture where the pipelining depth is limited.

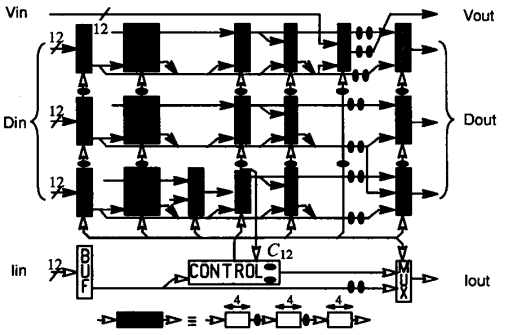


Fig. 5. A Group-Y architecture where the pipelining depth is unlimited.

V. Two-Level Pipelined Designs of the Advanced SAG Engine

Due to limited space, we cannot present the full details of the conversion of our SAG engine into two-level pipelined versions. The necessary steps to convert the advanced SAG engine into two-level pipelined designs are:

1) construct the graph G,
2) identify the critical paths and add pipeline registers into them to meet the given speed requirements,
3) 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 for our advanced SAG engine. We present these architectures in the following subsections.

A. The Group-X Architecture

If the speed requirements are such that only a few pipeline registers are necessary on the carry path, we can encounter a situation where no pipeline registers are required on the part of the adder in which the addresses are updated. We refer to this configuration as Group-X architecture. Fig. 4 shows an example where only two groups of pipeline registers have been placed on the 36-b data path. This divides the 36-b data path into three blocks of 12 b each. Let us assume that the instruction I_{in} and the first 12-b block of the input data D_{in} are supplied to the processor at the nth cycle. The instruction provides the proper control signals to the data path such that the correct inputs are provided to the adder. As the processor addresses are represented by 12-b numbers and sent into the processor on the first 12-b block of the data path, any decision related to a processor address can be taken in the same time slot as the address is decremented. We recall that the rightmost multiplexers select the proper output data depending on the decisions related to the processor addresses. Therefore, the output instruction I_{out} and the first 12-b block of the output data D_{out} can be produced at the nth cycle. In the case of the second 12-b block, the carry input to the adder is delayed by one clock cycle due to the first pipeline register on the carry path. Hence, the input and output activities of this adder must occur at the $(n+1)$th cycle. The following conditions are required for the second 12-b block:

1) the control signals which provide the inputs to the adder must be delayed by one cycle with respect to the relevant control signals of the first 12-b block;
2) the control signals which use the output of the adder must be delayed by one cycle with respect to the relevant control signals of the first 12-b block;
3) the input data D_{in} must be supplied at the $(n+1)$th cycle.

The set of pipeline registers between the first and second 12-b block has been inserted to meet the first and second requirements. As the second section of the adder is generating its output at the $(n+1)$th cycle, the second 12-b block of the output data D_{out} is generated on the $(n+1)$th cycle. Similarly, the pipeline registers between the second and third 12-b blocks of the data path are inserted. For proper operation, the last 12-b block of the input data D_{in} must be supplied in the $(n+2)$th cycle and the last 12-b block of the output data D_{out} will be generated in the $(n+2)$th cycle. Now we can see that the skews of different blocks of the input data D_{in} and output data D_{out} are compatible with each other such that two processors can be cascaded. In the case of the architecture in Fig. 3, the speed improvement of Group-X architectures is limited to a factor of 3.

B. The Group-Y Architecture

When the speed requirements are severe, more pipeline registers must be placed on the carry ripple path of the adder. As soon as pipeline registers are introduced into the section of the adder where the address is decremented, any decision related to a processor address cannot be taken in the same cycle as the least significant bit of the address is supplied. We refer to this configuration as Group-Y architecture. Fig. 5 depicts an example, where pipeline registers are placed 4 b apart. When the address is represented by a 12-b number, any decision related to the address has to be postponed by two cycles. Hence, the inputs to the rightmost multiplexers must be delayed by two cycles. The pipeline registers on the horizontal buses provide the necessary delays. Similar to the Group-X architecture, if there is a pipeline register between the kth bit and $(k+1)$th bit of the adder, pipeline registers are necessary on all the control signals which provide/use the data to/from the $(k+1)$th bit of the adder. Furthermore, the input data D_{in} must be skewed. We notice that the latency between any input port to the relevant output port has been increased by two cycles.

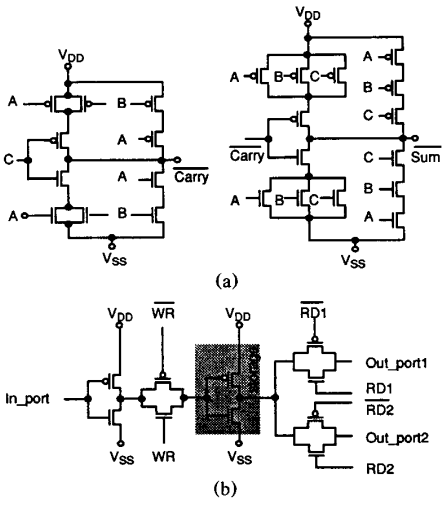


Fig. 6. Some circuits used in the prototype. (a) The adder. (b) The multiport register used for registers A, B, and C.

In general, if there are k pipeline registers on the section of the adder where the addresses are updated, k pipeline registers are necessary on the horizontal buses. As C_{12} is an input to the controller, some pipeline registers are also necessary in the controller. The speed improvement of this group is superior to the previous group, and the speed of a 1-b adder is the limiting factor.

VI. A Prototype Design

Due to the superior speed of Group-Y architectures, we have decided to implement the architecture given in Fig. 5 in silicon. The following steps were followed during the design:

1) hardware description of the SAG engine to verify its behavior with the specifications;
2) two-level pipelining with estimated propagation delays;
3) hierarchical decomposition and floor planning;
4) leaf cell design using symbolic layouts;
5) circuit extraction for functionality verification of leaf cells and better propagation delay estimation by SPICE;
6) refinement of the design by repeating steps 2–5;
7) design rule checking, electrical rule checking, and then refinement of the design;
8) circuit extraction to verify the functionality of the complete design by switch-level simulation and verification.

For the hardware description, the in-house developed hardware description language MoDL [9] was used. For the symbolic layout design, the symbolic layout design system CAMELEON [10] was used. The procedural design language GrapMG [11] was used to build the processor in a hierarchical form using the custom designed leaf cells. The design rule checking and electrical rule checking were done by the tools in the DRACULA design system.

Most leaf cells were designed such that they can be abutted. The adder is shown to be the critical cell which determines not only the speed of the processor but also the area of a PE to considerable extent. The adder shown in Fig. 6(a) was used due to its compactness, as all the transistors of the same type can be implemented in a single diffusion area. A two-phase serial shift register was used for the input buffer BUF. The gate capacitance of an inverter has been used as the storage element for the registers as the maximum storage time is a few microseconds. The registers have several I/O ports and transmission gates were used to select the proper ports. Fig. 6(b) depicts the circuit of the multiport register used for the registers A, B, and C. The multiplexers are also based on the transmission gates. In the refined design (step 6), the width of the first section of the data path was reduced to 3 b due to instruction decoding delays. For regularity, we decided to implement the data path as 3-, 5-, 4-, 3-, 5-, 4-, 3-, 5-, and 4-b-wide sections. Fig. 7 is a microphotograph of the prototype which consists of nine PE's. It contains the equivalent of 85K minimum feature size transistors in a 1.6-μm CMOS technology and the design was done in a university environment (see Table II for more details). The area overhead of the pipeline registers is approximately 25%. Due to relatively low processor count per chip, we have investigated an improved design. Using Domino logic and better circuits, the transistor count per PE can be reduced by approximately a factor of 2.3. When cells are abutted, some of them have to be stretched. Better layouts can minimize area overhead due to cell stretching. According to our estimate, the density can be improved by approximately a factor of 2.5 using better mask layouts. Therefore, given adequate resources, 50 ~ 60 PE's could be integrated in a single chip using the same 1.6-μm CMOS technology without any speed reductions. Submicrometer technologies could provide even better results.

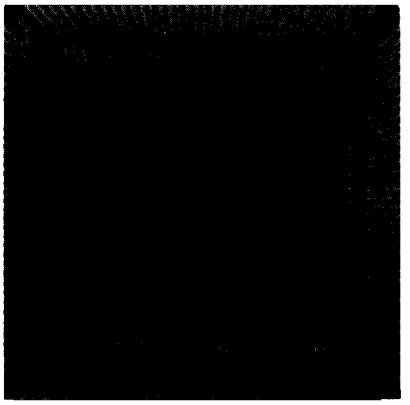


Fig. 7. Microphotograph of the prototype SAG engine.

TABLE II
The Characteristics of the Prototype (in Full CMOS) and the Expected Characteristics of an Optimized Design (in Domino Logic)

Characteristic	Prototype	Optimized Design
Pixel rate	66 ~ 83 MHz	66 ~ 83 MHz
Transistor count	85K	190 ~ 230K
PE count	9	50 ~ 60
Process	1.6-μm CMOS	1.6-μm CMOS
Chip size	10.2×11.4 mm^2	10.2×11.4 mm^2
Power supply	5 V	5 V
Power dissipation	< 5 W at 83 MHz	< 5 W at 83 MHz
Package	144 PGA	144 PGA

The testability of the design is an important aspect. The advanced SAG engine was designed such that each processor can be tested individually by bypassing the other processors. The signals of the test PE are observed via a serial shift register and are controlled via the ports I_{in} and D_{in}. The prototype chip has been tested and it is fully functional.

VII. Conclusions

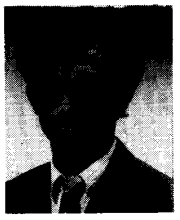
By converting the computationally intensive Phong shading method into second-order interpolation, it is now possible to generate images faster than Gouraud shaded images using the same amount of processing power. Therefore, real-time Phong shading becomes a reality. Due to the robustness of our approach, no visible errors are introduced. The simplicity of our approach enables significant speed improvements for Phong shading even with general-purpose hardware. The speed improvements can be further enhanced by ASIC's and hence we designed an advanced SAG engine. A silicon implementation of a prototype SAG engine supporting an advanced instruction set for real-time Phong shading is reported. Apart from Phong shading, it also supports Gouraud shading and Constant shading. The speed of the advanced SAG engine is improved by two-level pipelining. The two-level pipelined design is derived using the formal approach presented in Section IV which handles time-variant and shift-variant systolic arrays. The advantage of the two-level pipelining is its capability to provide complex functionalities at high pixel rates, which is difficult to achieve by other means using the same 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 supporting realistic shading techniques will be a breakthrough for real-time computer graphics.

Acknowledgment

Thanks to J. Huisken and other members of the VLSI Design Group at Philips Research Laboratories, Eindhoven, for their help and for allowing us to use their design system and fabrication process. The members of the Interactive Systems Group, CWI, Amsterdam, are also acknowledged for the discussions during the specification development phase of the advanced SAG engine. Thanks also to S. Gerez and K. Slump at the University of Twente, Enschede, for the critical remarks and proofreading.

References

[1] P. J. W. ten Hagen *et al.*, "Display architecture for VLSI-based graphics workstation," in *Advances in Graphics Hardware I*. Berlin: Springer-Verlag, 1987.
[2] N. Gharachorloo and C. Pottle, "SUPER BUFFER: A systolic VLSI graphics engine for real time raster image generation," in *Proc. 1985 Chapel Hill Conf.* VLSI, 1985, pp. 285–305.
[3] N. Gharachorloo *et al.*, "Subnanosecond pixel rendering with million transistor chips," in *Proc. SIGGRAPH*, Aug. 1988, pp. 41–49.
[4] T. Nishizawa *et al.* "A hidden surface processor for 3-dimension graphics," in *ISSCC Dig. Tech. Papers*, Feb. 1988, pp. 166–167.
[5] J. A. K. S. Jayasinghe *et al.*, "A display controller for a structured frame store system," in *Advances in Graphics Hardware III*. Berlin: Springer-Verlag, 1989.
[6] J. D. Foley and A. van Dam, *Fundamentals of Interactive Computer Graphics*. Reading, MA: Addison Wesley, 1984.
[7] H. T. Kung and M. S. Lam, "Fault-tolerance and two-level pipelining in VLSI systolic arrays," in *Proc. Conf. Advanced Res. VLSI*, Jan. 1984, pp. 74–83.
[8] J. A. K. S. Jayasinghe and O. E. Herrmann, "Two-level pipelining of systolic array graphics engines," in *Advances in Graphics Hardware IV*. Berlin: Springer-Verlag, 1990.
[9] J. Smit *et al.*, "The MoDL hardware design system," in *Proc. 8th Int. Conf. Computer Hardware Description Languages and Their Applications*, Apr. 1987, pp. 327–342.
[10] K. Croes *et al.*, "CAMELEON, A process tolerant symbolic layout system," in *Proc. European Solid-State Circuits Conf.*, Sept. 1987, pp. 193–196.
[11] H. Jansen *et. al.*, "GrapMG: Cost effective module generation," in *Proc. European Solid-State Circuits Conf.*, Sept. 1989, pp. 86–71.



J. A. K. S. Jayasinghe (S'86–M'88) was born in Colombo, Sri Lanka, in 1960. In 1984 he received the B.Sc. (engineering) degree in electronics and telecommunication engineering from the University of Moratuwa, Sri Lanka. In the same year, he joined the academic staff of the same university as an Assistant Lecturer. In 1987 he received the M.E.E. degree from the NUFFIC (Netherlands University Foundation for International Cooperation). Currently he is working towards the Ph.D. degree at the University of Twente, Enschede, The Netherlands. His main research interests are parallel processing systems for computer graphics and medical electronics.

F. Moelaert El-Hadidy was born in Prague, Czechoslovakia, in 1959. In 1982 she received the B.Sc. degree in communication and electronics engineering from the University of Cairo, Egypt. In December 1986 she received a Diploma from the Philips International Institute, Eindhoven, The Netherlands. In 1989 she received the M.Sc. degree from the University of Cairo. She is currently working towards the Ph.D. degree at the University of Twente, Enschede, The Netherlands.

She worked as a software engineer in Egypt from 1982 to September 1985. She then worked in the Philips CAD center in Eindhoven on an analog circuit analysis package until August 1987. Her interests include parallel architectures, systolic arrays, graphics, and ASIC's.

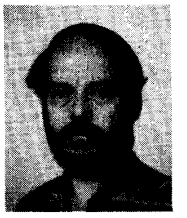
G. Karagiannis was born in Braila, Rumania in 1964. In 1987 he received the "Technologos Mihanikos" degree in electronics from the Technical Institute of Education (T.E.I.) of Athens, Greece, In 1988 he received the Ing. degree in electrical engineering from the Polytechnical High School (H.T.S.) of Enschede, The Netherlands.

In the same year he joined the staff of the University of Twente, Enschede, The Netherlands. Currently he is working in the field of IC design developing a graphics workstation. He is a part-time M.Sc. student in electronics at the same university.

Otto E. Herrmann (M'72) was born in Düsseldorf, West Germany, in 1933. He received the Dipl.-Ing. and Dr.-Ing. degrees in electrical engineering from the University of Technology Aachen, West Germany, in 1959 and 1965, respectively. In 1971 he received the "venia legendi" for telecommunication from the University of Erlangen, West Germany.

From 1959 to 1965 he was a Research Associate and Lecturer at the Universities of Aachen and Karlsruhe. From 1966 to 1972 he was Se-

nior Staff Member and from 1972 to 1975 "Abteilungsvorsteher" in the Department of Electrical Engineering of the University of Erlangen. During 1972 he was a Visiting Scientist on leave at Bell Laboratories, Murray Hill, NJ. Since November 1975, he has been heading the group for network theory, signal processing, and CACSD as a full Professor at the Faculty of Electrical Engineering at the University of Twente, Enschede, The Netherlands.



J. Smit was born in 1944 in Berkhout, The Netherlands. He received the Ing. degree in electrical engineering in 1966 from the Polytechnical High School (H.T.S.) in Enschede, The Netherlands, and the B.Sc. degree in electrical engineering in 1971 and the Ir. degree in electrical engineering in 1975 from the University of Twente, Enschede, The Netherlands.

Since 1971 he has been working at the University of Twente at several levels of responsibilities. Currently he is a senior staff member ("Universitair hoofd docent") in the Laboratory for Network Theory at the University of Twente, where he is responsible for research and education in the area of VLSI design.