A 33.2Mvertices/sec Programmable Geometry Engine for Multimedia Embedded Systems

Chang-Hyo Yu, Donghyun Kim and Lee-Sup Kim
Dept. of EECS, KAIST
Daejeon, Republic of Korea
{mosmov,dhkim}@mvlsi.kaist.ac.kr, lskim@ee.kaist.ac.kr

***Abstract*—This paper proposes a programmable geometry engine (GE) reducing the expensive internal buffers and register files of the conventional programmable GEs and sharing datapaths of a special function unit. The proposed GE is appropriate for the embedded 3D graphics environment where the reduction of hardware cost is a critical issue. The degraded performance caused by the hardware reduction is compensated by a variable write-back timing architecture with a dynamic hazard controller and a data forwarding method. The GE is implemented by a 0.13um CMOS technology and has the performance up to 33.2Mvertices/sec, which is 1.66 times improvement of the previous work. Its equivalent gate count is 206k and operation frequency is 166MHz.**

I. INTRODUCTION

Today's graphics hardware requires higher performance of the operating ability as well as more flexibility to support variety of the advanced 3D algorithm to achieve the more realistic rendering image. A conventional hard-wired geometry engine (GE) could get a higher performance by the increasing semiconductor technology, but the rapidly developing 3D rendering algorithm requires the hard-wired architecture change. Therefore, we need a programmable architecture to catch up with the up-to-date techniques, and we also need a higher performance GE processor while supporting the programmability without performance degradation.

The programmable GE architecture was first designed by Lindholm [1]. He suggested the programmable architecture for the flexibility of 3D geometry operations. It does basically light and transform operations by an assembly like language. The latest 3D graphics hardware commodities, produced by ATI or NVIDIA [2], provide powerful performance of their programmable GE up to 600Mvertices/sec by the various high techniques with the GDDR3 memory, six parallel GEs, and so on.

However, these graphics hardware are designed for the high-end desktop PC. The core of NVIDIA GeforceFX6800 ultra [2], one of the fastest PC-platform graphics chip, is manufactured by 222 million transistors, and has about 174Watts of power consumption. We need much less hardware cost and power consumption for various multimedia systems on chip (SoC) environments such as Digital TV, personal digital assistant (PDA), and the other home appliances. There are several studies for the embedded 3D graphics core of mobile environment. Kameyama [3] developed a 3D graphics LSI core for mobile system (Z3D). Z3D also integrates the programmable GE, which has the performance up to 185k vertices/sec. This core is suitable for the current mobile platform which has the display resolution below the QVGA (320x240). Imai [4] had shown a 3D graphics engine for the non-PC products. The engine has also a programmable geometry processor with 4.7Mvertices/sec of performance and 109.5mW of power consumption. This engine is focused on reducing the power consumption; the core uses a multiple clock domains in GE and latch based registers. Sohn [5] described a 3D graphics GE using fixed-point datapath for low power graphics system. The engine uses a fixed-point datapath for GE instead of the floating-point datapath with a precision degradation not shown in the low display resolution.

The non-PC platform, such as home application's SoC environment, grows rapidly at the point of the size of display resolution as well as the quality of 3D rendering image. The previous works had shown good results at the present time, but the incoming multimedia SoC environment requires higher performance to support a higher screen resolution (QVGA or more) and the flexibility to adopt the advanced 3D algorithm.

In this paper, we developed a 3D graphics GE for the multimedia SoC environment with the performance up to 33.2Mvertices/sec at 166 MHz and 71mW of power consumption.

The remainder of the paper is arranged as follows. Section II shows the architecture of programmable GE. Section III shows the implementation of this work and Section IV discusses the results while the last Section concludes this paper.

This research was sponsored by SAMSUNG electronics and the Consortium of Semiconductor Advanced Research through the SYSTEM IC 2010 project, Korea

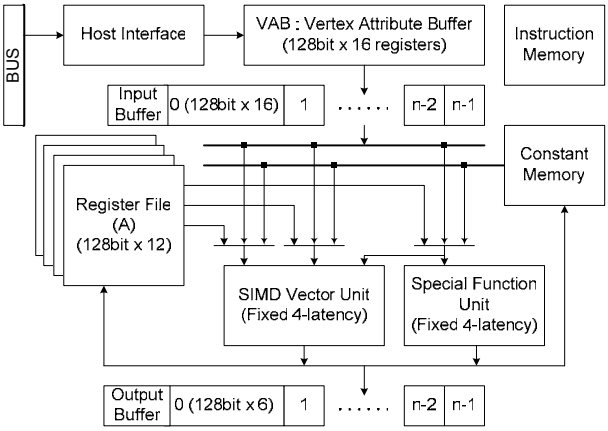

Figure 1. The conventional programmable GE architecture of the high-end desktop PC. It is focused on maximizing the performance.

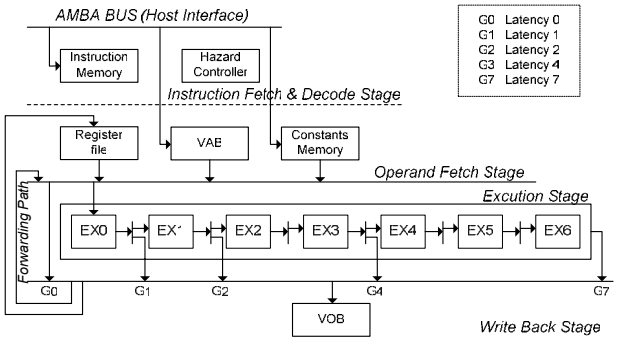

Figure 2. The proposed architecture; a variable write-back timing geometry pipeline with data forwarding and hazard controller.

II. PROGRAMMABLE GEOMETRY ENGINE

A. The conventional programmable Geometry Engine

The programmable GE is a first user-programmable stream architecture that exploits a vertex parallelism and streaming nature [1]. GE takes one vertex data as inputs and produce one transformed vertex data as outputs. No vertices affect the other adjacent vertices in the same frame.

Fig. 1 shows the GE architecture. The vertex attributes are converted to floating-point format before arriving at the vertex attributes buffer (VAB), which has room for the sixteen input attributes. In a round-robin fashion, the VAB drains into a number of input buffers that are used to feed the floating-point core. Vertex data is read from the input buffers and transformed to the output buffers. Only one vertex attribute may be read per program instruction. To hold constants such as matrices, light positions, and plane coefficients that are used in typical vertex programs, there is a memory bank of more than 128 quad-floats. Each temporal register files is 16 quad-floats in size and allows three reads and one write per instruction. Any vector read may be sourced as multiple operations and individually swizzled/negated each time. Since the latency of floating-point core is generally four [1], a vertex engine can handle parallel processing up to four vertices. This means four register files are required for vertex multi-threading, but a programmer understands as if only one register file exists. Finally, the transformed vertices are feed to the hardwired primitive engine for polygon culling and clipping.

B. The proposed Geometry Engine

However, the conventional high-end PC architecture provides excellent performance, the hardware cost and power consumption is not capable of integrating SoC environment. We must modify the conventional architecture to get sufficient performance with reduced hardware cost and less power consumption for the home plat-form and the other applications. The major commercial graphics hardware vendors reveal that their floating-point datapath has four-latency for the longest instruction. Although the short latency is better, the hardware cost is burden to integrate SoC system. In our design, we have the maximum latency to be seven for the special functions by sharing the datapath. Therefore, we need seven number of register files and input/output buffers for multi-threading. However, the number of register files and their control method makes the hardware complex. We try to reduce the number of register files while the performance does not degrading. To solve this problem, our architecture uses the single-threaded operations with different write-back latency method. Therefore, we only need one register file and do not need a number of input/output buffers.

The execution pipeline stage is divided into seven sub stages. Each sub stage can write-back to the vertex output buffer (VOB) or register file with its latency cycle time. This means the instructions, which have different latency, can write back at different cycle time. By the variable write-back method leads us to speed up the single-threaded geometry operations. Fig. 2 shows the proposed GE architecture. The engine composed of four way SIMD datapath, special function unit (SFU), register file, VAB, VOB and internal memory for instruction and constant variables. Since our GE is designed to share the hardware resources and to use the different write-back timing method, there can be a data dependency and a confliction of the write-back operation. To solve this problem, we use two techniques: hazard control method and forwarding. The hazard controller detects all the cases of data dependency and adjusts the issue timing of the instruction fetch. And the forwarding reduces the dependency by passing a result directly to the functional unit that requires it.

III. IMPLEMENTATION

The hardware implementation of GE is divided into two main blocks: the floating-point core and hazard control unit. The floating-point core processes the instruction set, and the hazard control unit adjusts an issue timing of the instruction caused by data dependency and lack of hardware resources.

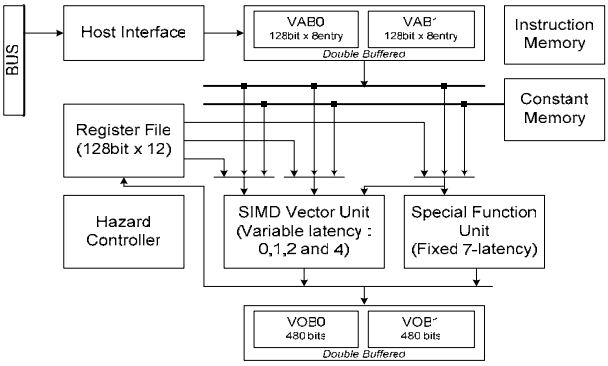

Figure 3. The proposed GE architecture. 4-way SIMD vector unit has variable latency instruction for fast write-back. SFU unit has fixed 7-latency for reducing the hardware cost. There is only one register file instead of the number of maximum latency (for multi-threading, we require 7). A number of Input buffers and Output buffers are substituted for double buffered VAB and VOB. It reduces the buffer size two seventh or less.

A. *Overview of the Geometry Engine*

The overall diagram of the GE is shown in Fig. 3. Vertex attributes, the inputs from the host, are transferred into VAB, which is composed of eight quad-float (128bit) entries with double buffering method to hide a transferring latency. The VOB is also implemented with double buffering method.

B. *The floating-point core*

The floating-point core is a vector processor operating on quad-float data. Vertex data is read from the VAB and transformed into the VOB. The latency of the vector and special function units are grouped into five: G0, G1, G2, G4 and G7.

The SIMD vector unit is responsible for the MOV, MUL, ADD, MAD, DP3, DP4, DST, MIN, MAX, SLT, and SGE operations. The special function unit is responsible for the RCP, RSQ, LOG, EXP and LIT operations. The SIMD datapath needs four cycle latency in the longest instruction (G4). We design a floating-point multiplier with one cycle and a two input adder with two cycles. Four input floating-point adder has three cycle latency. Therefore the longest latency group, G4, pass through the multiplier and the four-input adder. The group G7 is operated by the SFU unit, which has the latency of seven. The transcendental functions (logarithm, exponent and etc.) and division need several cycle times of iteration to support high precision. However, we want to design these functions fast while having a certain extent of precision. To achieve a fast latency with reasonable hardware cost, the SFU unit shares the datapath; floating-float multipliers, 4-input adder and other functional blocks, and uses a table lookup method by a 2KB of ROM. The logarithm and exponent has 10-bit precision of their mantissa and the reciprocal and square root functions have 22-bit precision of mantissa field. These amounts of precision support to a standard API of the vertex shader version 1.1 [7]

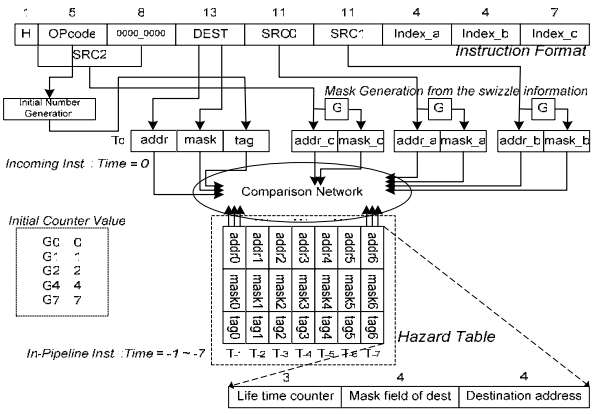

Figure 4. The description of the hazard control unit

TABLE I. THE PREFERRED FORMAT OF THE VAB AND VOB

VAB (field)	Description (128bit)
0	Vertex Position
1	Reserved
2	Vertex Normal
3	Vertex Color
4	Reserved
5	Reserved
6	Texture Coordinate
7	Reserved

VOB	Description
Vertex Position	128 bits (x,y,z,w)
Vertex Color	128 bits (r,g,b,a)
Texture Coordinate 0	96 bits (s0,t0,q0)
Texture Coordinate 1	96 bits (s1,t1,q1)
Vertex Fog	32 bits

C. *The hazard control unit*

Dependencies can be occurred due to the two reasons: first, the data dependency of the pipeline architecture with a single-threaded operation. Second, the limited hardware resources in the variable latency write-back architecture. Since the 3D graphics geometry operations have less data dependency than the operations of the general processor because of the vertex parallelism, we could manipulate the dependencies easily with two strategies: data forwarding and adjusting the instruction issue timing.

There are three types of dependencies: RAW, WAW hazards, and write-back conflicts. The RAW and WAW hazards are prevented by two ways: first, the data forwarding between write-back stage and the first execution stage for short interval of the consecutive instructions, second, the adjusting an instruction fetch timing for the other cases. The write-back conflicts are caused by the limited datapath, which has only one write port of the register file. It is solved by the issue timing control. The hazard controller has the write-back timing information of the current execution slots. When the incoming instruction would be conflicted with an in-pipeline instruction, the hazard controller pushes a NOP instruction to the pipeline.

The hazard control unit is shown in Fig. 4. It contains the internal registers for the information of the in-pipeline instructions, which is the history of issued instructions. The registers are composed of three fields; the first field is a 4-bit tag for the lift-time counter which is decremented each cycle. The tag is used for determining the write-back timing. The second field is the destination address of the register file. It

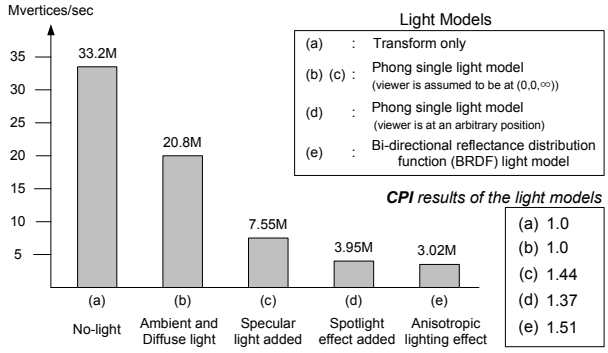

Figure 5. The performance of the GE. The various light models are used for testing the vertex processing speed. The average clock cycles per instruction (CPI) of our GE at the light models are shown in the bottom box.

TABLE II. THE GATE COUNTS AND POWER CONSUMPTIONS

0.13um CMOS process 1.2V 166Mhz	Equivalent Gate Count	Power Consumption at the light model (c) of Fig. 5
Geometry Engine Core	189517	58790uW
SFU Lookup Table(ROM)	16514	5911uW
Instruction Memory(SRAM)	1KB SRAM (52317)	6474uW
Constants Memory(SRAM)	1.5KB SRAM (4827)	304uW
Geometry Engine Total	206031 + 2.5KB SRAM	71479uW

is used for testing RAW hazard comparing the incoming source addresses with and the in-pipeline destination addresses. The last field is the write mask information. Since the data field of 3D system is usually composed of quad-float vector (128-bit), each floating-point field of a vector can take independent value other fields. Therefore, the hazard controller discriminates not only the addresses but also the mask information. The additional information makes the detection of the hazards sophisticated.

IV. DISCUSSTIONS

This work is a part of the embedded 3D graphics SoC system. Table II shows the equivalent gate count of our GE core, lookup table, and SRAM memory. Table II also reports their power consumption during the runtime of the full 3D applications (with the light model (c) of Fig. 5). A 206k logic gates and 2.5kB SRAM are integrated. The total GE consumes about 71mW at 166 MHz of operating frequency.

The performance of our GE is presented in Fig. 5. The average clock cycles per instruction (CPI) reveals the efficiency of the architecture. The high-end GE cores take an effort to be one of CPI (or less than one by using multiple cores). Fig. 5 also shows the CPI results of our GE at the various light models. Although our GE is developed by sharing the datapath in the SFU and reducing the expensive internal buffers and register files, the CPI can be achieved to be 1.0 at the light model (a) and (b) as shown in Fig. 5. Even though the complex light model, which has longer program code, has larger CPI due to the number of hazards in the

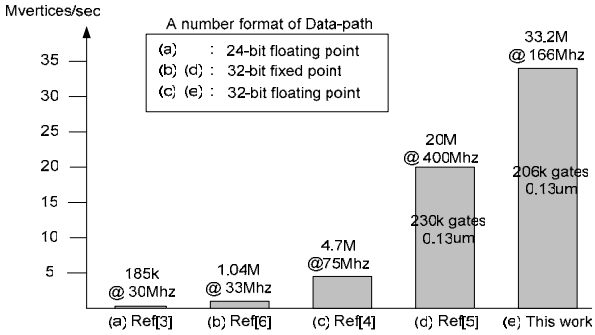

Figure 6. The performance comparison between our GE and the privious works.

instructions, the CPI is also not large. It means our GE architecture is very efficient. We compare our hardware with the previous results [3-6]. Fig. 6 shows the comparison. Even though the logic gate count is smaller than that of the engine described in [5], the maximum vertex processing speed of our GE is increased up to 1.66 times without any precision degradation by using the 32-bit floating-point datapaths in the full pipelines.

V. CONCLUSIONS

In this paper, we presented a 3D graphics geometry hardware, and implemented it into a real application for multimedia embedded systems. In spite of the hardware reduction, the hardware provides high performance by the variable write-back timing method with the well-designed dynamic hazard controller. The proposed programmable GE processes up to 33.2Mvertices per second, and it has a 71mW of power consumption at 166 MHz of operating frequency. The proposed GE has 1.66 times performance improvement of the previous system [5] even supporting the single precision floating-point. The incoming 3D graphics SoC environment, which has larger display resolution (QVGA or higher), can be supported by our GE.

REFERENCES

[1] Erik Lindholm, Mark J. Kilgard, and Henry Moreton, "A User-Programmable Vertex Engine", SIGGRAPH 2001, pp.149-158

[2] NVIDIA, http://www.nvidia.com/page/geforce_6800.html

[3] Masatoshi Kameyama, et al., "3D Graphics LSI Core for Mobile Phone Z3D," Proceeding of Graphics Hardware 2003, pp.60-67, Jul 2003

[4] Masatishi Imai, et al., "A 109.5mW 1.2V 600M texels/s 3-D Graphics Engine," ISSCC Dig. Tech. Papers, pp.332-333, Feb 2004

[5] Ju-Ho Shon, et al, "A Programmable Vertex Shader with Fixed-Point SIMD Datapath for Low Power Wireless Applications," Proceeding of Graphics Hardware 2004, pp107-114, Aug 2004

[6] Ramchan Woo, et al., "A 210mW Graphics LSI Implementing Full 3D Pipeline with 264Mtexels/s Texturing for Mobile Multimedia Applications," ISSCC Dig. Tech. Papers, pp.44-45, Feb 2003

[7] Microsoft, DirectX vertex shader stanard, http://msdn.microsoft.com /archive/default.asp?url=/archive/en-us/dx81c/directxcpp/Graphics /Reference/Shader/Vertex/Instructions/Instructions.asp