3D Graphics System with VLIW Processor for Geometry Acceleration

Young-Wook Jeon*, Young-Su Kwon*, Yeon-Ho Im*, Jun-Hee Lee*,
Sang-Joon Nam**, Byung-Woon Kim*, and Chong-Min Kyung*
**MCU design team 4, System IC, Hyundai Electronics Industries
*Dept. of Electrical Engineering and Computer Science, KAIST, Taejon 120-749,Korea
E-mail: woogy@vslab.kaist.ac.kr

Abstract

To process enormous 3D data, we have designed a VLIW (Very Long Instruction Word) processor called FLOVA (Floating-Point VLIW Architecture) exploiting the ILP (Instruction-Level Parallelism) in 3D programs. This paper presents FGA (FLOVA Geometry Accelerator) that is the 3D graphics system and it almost removes the time required to process the geometry stage. We have developed the 3D graphics library, FGA-GL, to supports the FGA system. The deferred primitive rendering algorithm of FGA-GL enables the geometry processing of the primitive data to be done concurrently with the host job such as primitive data management or game play. FGA improves the average performance of 3D graphics system by 2.5-3.0.

I. INTRODUCTION

The 3D polygonal graphics pipeline is composed of three main stages: primitive data management, geometry stage, and rendering stage. High-end 3D graphics accelerators provide dedicated hardware to accelerate the geometry stage but low-end cards generally delegate them to the software. Several technologies were developed to accelerate the geometry stage because current CPU cannot keep pace with the latest low-end 3D graphics accelerators[1][2][3]. The parallel processors or SMP computers are used to process the geometry stage in [4][5]. The PMesa uses shared memory parallel processors to process the geometry stage. The frequent change of the global context and the thread synchronization decreases the performance of PMesa. SIMD instructions are added to the host processor to accelerate the geometry stage[6][7]. Although the geometry processing is accelerated by SIMD instructions, the host processor must do not only the geometry processing but also the host job such as primitive data management or the execution of 3D data generation algorithm.

We have used the independent geometry processor, FLOVA (FLOating-point Vliw Architecture) that has a 4-way VLIW architecture to accelerate the 3D graphics system of PC. The 3D geometry data have inherent parallelism as pointed out in [8] and thus the algorithms of geometry processing can exploit the instruction-level parallelism efficiently in FLOVA. The FGA (FLOVA Geometry Accelerator) system is the geometry accelerated graphics system that uses FLOVA as the geometry engine and Voodoo™ as the rendering engine. FGA-GL (FGA Graphics Library) that has API's similar to that of OpenGL is the 3D graphics library for FGA. FGA-GL can almost remove the geometry processing time using *deferred primitive rendering*. The deferred primitive rendering enables the concurrent execution of the geometry processing and the host job such as primitive data management or the game play.

This paper is organized as follows. In Section II, we describe the deferred primitive rendering algorithm. The architecture of VLIW geometry processor is explained in Section III and the experimental results are shown in Section IV. Section V concludes the paper.

II. FGA-GL WITH DEFERRED PRMITIVE RENDERING

The graphics library for our geometry acceleration system uses a standard OpenGL™ API's and there is no need to re-write 3D applications to use our library. We have modified Mesa that is a public domain implementation of OpenGL [9] [10]. The deferred primitive rendering that almost removes the geometry processing time is implemented in our graphics library, FGA-GL.

The traditional OpenGL graphics pipeline in PC is shown in Fig. 1. P_i means the i-th primitive array that is a set of primitives to be drawn. C_i means the "state" of the OpenGL and it is the geometry and the rendering context. Before any primitive array is sent to the rendering engine, the state must be set to represent the suitable context. The state is set by the sequence of API's and the behavior of one API command is affected by the previous API commands. Most of the OpenGL API's are to change the state and only small part is to specify primitives. "Set C_i" in Fig. 1. means the execution of the API's that change the state of the i-th primitive array. $G(C_i, P_i)$ is the geometry processing of P_i using the context C_i. At the geometry stage, the primitives are transformed and lit by the light sources. The time required for $G(C_i, P_i)$ is over 70-80% of the total graphics pipeline according to 3D application benchmarks. $R(C_i, GP_i)$ is the rendering of the primitives whose geometry processing is ended. $R(C_i, GP_i)$ is usually done on the dedicated rendering hardware like Voodoo™ and the time required for the rendering is under 10%.

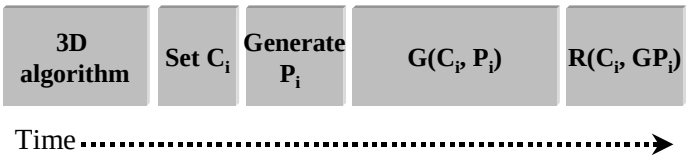

Fig. 1. The traditional OpenGL graphics pipeline in PC.

The state is saved as a structure of C language in Mesa. We have classified the state C_i into the geometry state, GC_i and rendering state, RC_i. The API's can also be classified into API's related to geometry state, API's related to rendering state, and API's that specify the primitives as shown in Fig. 2. The examples of API's related to the geometry are glMultMatrixf(), glTranslate(), glRotate(), glLightfv(), etc. Some of these API's are to change the transformation matrix and some are to set the attributes of light sources. The examples of API's related to rendering are glTexImage(), glAlphaFunc(), etc. There are API's that are related to both the geometry and the rendering. The API's that specify primitives are to set the coordinates in 3D space and the normal vectors of the pri-

mitives. The examples of such API's are glVertex3f(), glNormal3f(), etc.

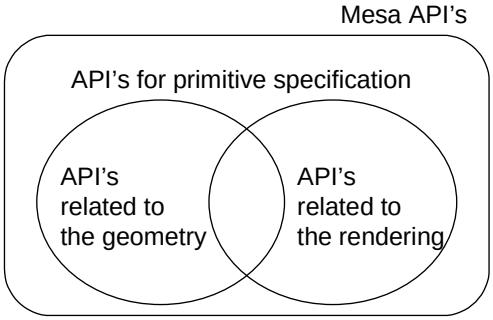

Fig. 2. Classification of OpenGL API's.

The algorithm of deferred primitive rendering is shown in Fig. 3. The dotted box is the basic granule that includes the rendering of (i-1)-th primitive array, change of i-th rendering state, host job, change of (i+1)-th geometry state and saving of (i+1)-th rendering state, and generation of (i+1)-th primitive. The geometry stage is done in VLIW geometry processor. The i-th primitive array, P_i is rendered after P_{i+1} is generated. The geometry processing of P_i, i.e., $G(GC_i, P_i)$ is executed concurrently with the host's job.

The state is set by API's related to geometry or rendering. These API's are called by the 3D application that does not discriminate the geometry state from the rendering state. The FGA-GL only set the geometry state, GC_{i+1} while it saves the rendering state, RC_{i+1}. The RC_{i+1} must be saved as in Fig. 3 because it should be set after the primitive P_i is rendered. The rendering state is saved as a list and the execution time for saving the rendering state is not considerable. If the geometry processing is not ended even after P_{i+1} generation, the host waits for FGA to end the geometry processing.

Fig. 4 shows the comparison of traditional graphics system (TGS) which is the 3D graphics system without FGA and FGA-GL. The geometry processing in TGS occupies most of the total execution time of 3D graphics pipeline, whereas that in FGA-GL is hidden due to parallel execution with host jobs.

III. VLIW GEOMETRY PROCESSOR

The geometry processor on the FGA board has a 4-way VLIW architecture called FLOVA with 13 functional units [11][12]. It consists of a branch/program control unit, three integer ALU's, an integer multiplier, a shift unit, two load/store units, three floating-point units, media ALU and multiplier. FLOVA has an instruction cache of 8KB, two data memories of 16KB and several peripherals like DMA, PCI, TIMER, and so on. The overall block diagram of FLOVA is shown in Fig. 5.

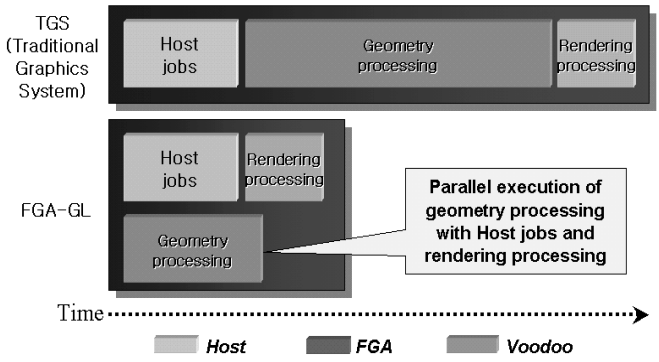

Fig. 4. Comparison of the total execution time per frame in the TGS and FGA-GL

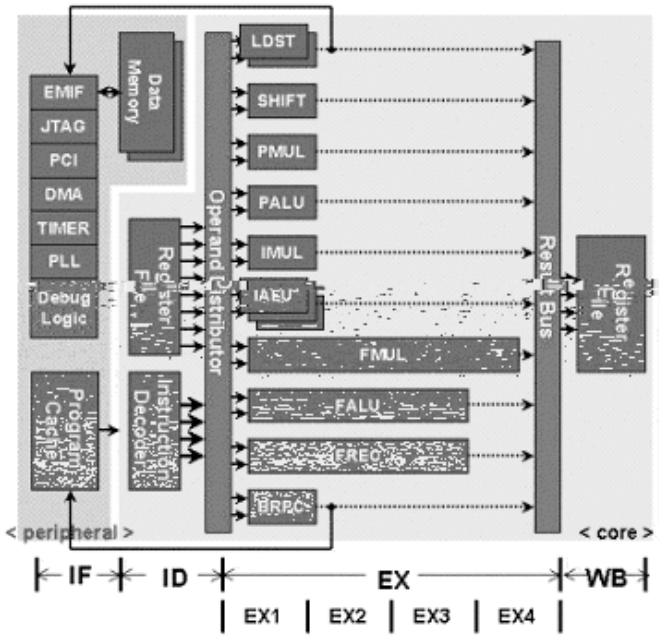

Fig. 5. The block diagram of FLOVA.

An instruction cache contains the compressed FLOVA instruction words without padding NOP's. The register file with 64 entries has eight read-ports and four write-ports and can be accessed as 32-bit or 64-bit register. When two single-precision floating-point operations or integer operations are executed, the register file is accessed as 32-bit, while in double-precision floating-point operations or media operations, it is accessed as 64-bit.

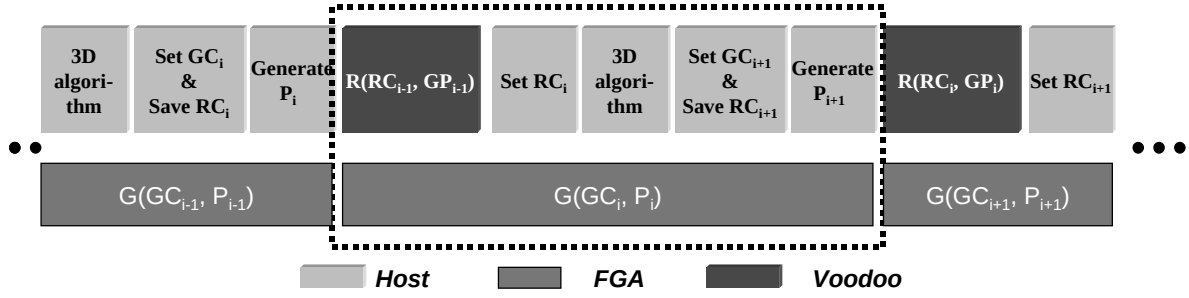

Fig. 3. Algorithm of deferred primitive rendering.

To accommodate a wide variety of multimedia applications, FLOVA supports rich packed 64-bit data types such as 8 packed bytes, 4 packed words, 2 packed doublewords and 2 single precision floating-point. Each element within a packed data type is a fixed-point integer. To enable a wide variety of image algorithms, FLOVA supports packed bytes. Basic support for packed doubleword data types is needed for operations that need higher precision than 16 bits and a variety of 3D graphics algorithms. A major feature of FLOVA instructions using rich data types is saturation arithmetic operations. These operations are very important, for example, in algorithms dealing with visual data such as 3D game implementing a Gouraud-shading technique. If the saturation operations are not supported, the complex saturation check routines need to be implemented with the branch instructions using carry or overflow flag, which significantly increases the execution cycle count.

FLOVA has three floating-point units that consist of an floating-points ALU, an floating-point multiplier, and floating-point reciprocal unit.

The floating-point multiplier (FMUL) has a special logic that computes the power operation for the lighting of the vertex based on piecewise linear approximation and multiplication. This special logic is called "Fastpow" which consists of two 32-bit carry-save adders and is easily merged in the floating-point multiplier to use internal 24x24 multiplier for fixed-point multiplication. The detailed operation of the "Fastpow" is described in [11][13]. It computes the floating-point power operation with only 4 cycles while it takes over 150 cycles or requires a large ROM table in other processors while the difference of power value generated by the "Fastpow" and SPARC is under 5%. The overall block diagram of the FMUL with "Fastpow" logic is shown in Fig. 6.

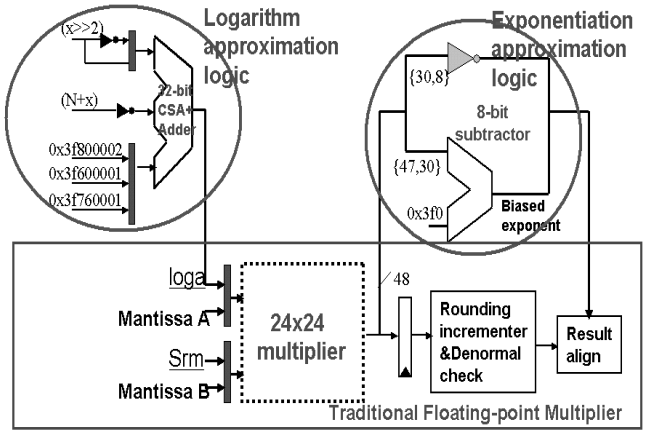

Fig. 6. The block diagram of FMUL.

The floating-point reciprocal unit can compute the reciprocal operation of the floating-point number with 8-bit precision. The 8-bit precision is sufficient for 3D graphics application. If the programmer wants more precision, the Newton-Raphson iteration method can be used to increase the accuracy.

The geometry code composed of FLOVA instructions is highly parallelized into four execution slots. To improve the parallelism of the lighting stage, we have used "paired lights" which compute the lighting of the vertex for two light sources concurrently.

IV. PERFORMANCE

T_G and T_{NG} is the geometry processing time and other time except geometry processing of the normal graphics system. T_C is the communication overhead by the interaction between host and FGA and T_{FG} is the geometry processing time consumed in the geometry processor. The theoretical performance improvement of FGA is as follows.

$$PI = \begin{cases} \dfrac{T_G + T_{NG}}{T_{NG} + T_C} & T_{NG} \geq T_{FG} \\ \dfrac{T_G + T_{NG}}{T_{FG} + T_C} & T_{NG} < T_{FG} \end{cases}$$

The experiments show that T_{FG} is usually smaller than T_{NG} and T_C is $0.1\ T_G$. - $0.3\ T_G$. Fig. 7 shows the experimental results of the performance of FGA for several benchmarks. The average of performance improvement against the normal graphics system without FGA is 2.5.

Fig. 8 shows the performance improvement as the light of the scene is increased. The lighting stage stands for 70% of the geometry processing time and we have optimized the lighting stage using the parallel execution of the FLOVA. The performance of FGA is better when more lights are used.

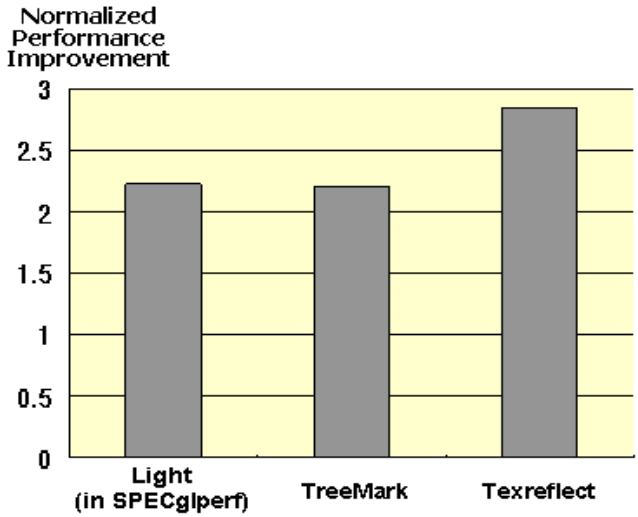

Fig. 7. The performance improvement of FGA.

Fig. 9 shows the layout of FLOVA. The estimated chip size of FLOVA processor consisting of about 650,000 gates, is 10mm × 10mm under 0.35μ.m TLM CBIC process, and performance is 2100 MOPS (Million Operations Per Second) / 500 MFLOPS (Million Floating-point Operations Per Second) at 100 MHz clock frequency.

FGA system board is shown in Fig. 10. It consists of SRAM, Board controller, and FLOVA. We have developed FGA system board as a PC add-on card and have used PCI interface for the communication between FGA and Host.

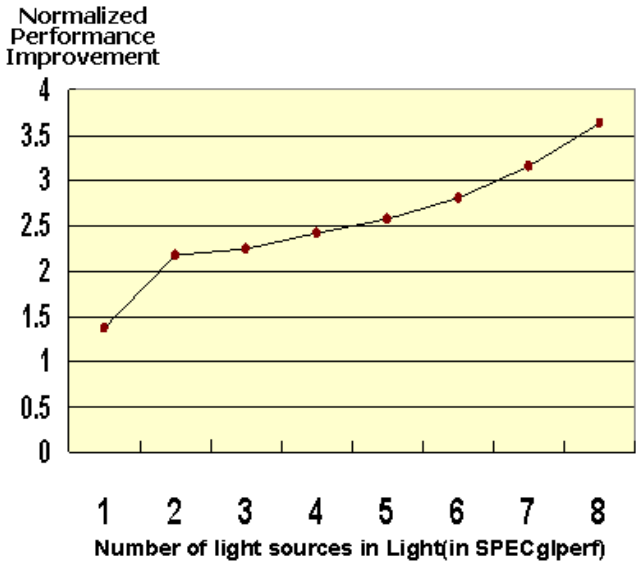

Fig. 8. The performance improvement of FGA as the number of lights is increased.

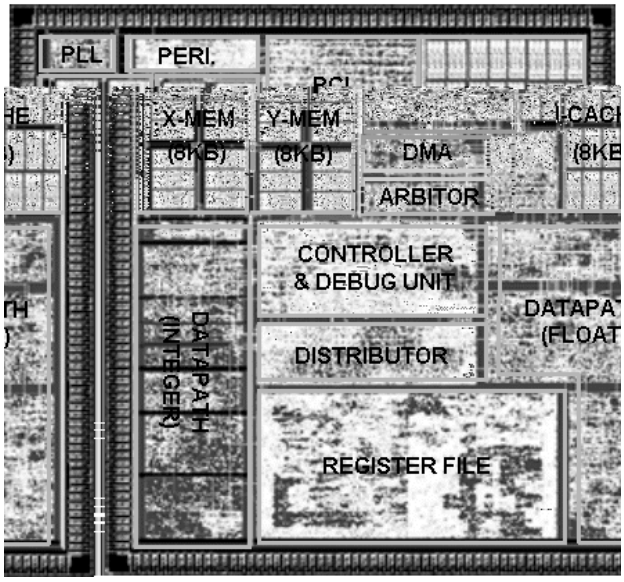

Fig. 9. The layout of FLOVA.

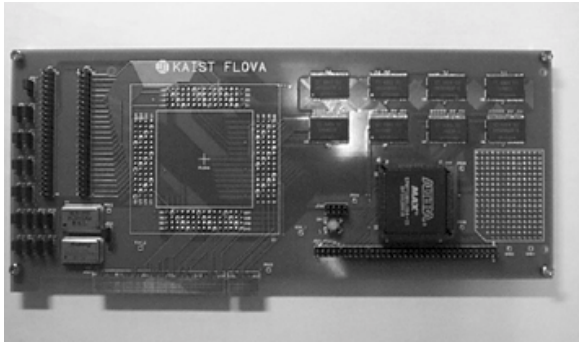

Fig. 10. The FGA system board that consists of SRAM, Board controller, and FLOVA.

V. CONCLUSIONS

We have designed the FGA that is the geometry accelerated graphics system using VLIW geometry processor called FLOVA and FGA-GL, which is the 3D graphics library for FGA. The deferred primitive rendering of FGA-GL enables the concurrent processing of the host job and the geometry processing and thus it almost removes the time required for the geometry processing in 3D graphics pipeline. The FLOVA can execute the geometry stage very efficiently with the fast lighting unit and two operation modes for the transformation in the geometry stage. FGA improves the average performance of 3D graphics system by 2.5-3.0.

REFERENCES

[1] Yulun Wang, Amante Mangaser, and Partha Srinivasan, “A Processor Architecture for 3D Graphics,” IEEE Computer Graphics and Applications, Vol. 11, No .5, pp. 96-105, Sep. 1992.

[2] John G. Torborg, “A Parallel Processor Architecture for Graphics Arithmetic Operation,” Proceedings of SIGGRAPH, pp. 197-204, 1987.

[3] J. H. Clark, “The Geometry Engine: A VLSI Geometry System for Graphics,” Computer Graphics, Vol. 16, No. 3, pp.127-133, July 1982.

[4] John S. Montrym, Daniel R.Baum, David L. Dignam, and Christopher J. Migdal, “InfiniteReality : A Real-Time Graphics System,” Proceedings of SIGGRAPH, pp. 293-301, 1997.

[5] J. Sebot Julien, A. Vartanian, J-L Bechennec and N. Drach-Temam, “A Parallel Algorithm for 3D Geometry Transformations in OpenGL,” Europar, pp. 669-652, 1999.

[6] ``3DNow! Technology, Delivering Leading-Edge 3D Graphics and Multimedia Performance for the New Era of Realistic Computing,", Advanced Micro Devices, INC., May 1998.

[7] Shreekant Thakkar and Tom Huff, “The Internet Streaming SIMD Extensions,” Intel Technology Journal Q2, 1999.

[8] Homan Igehy, Gordon Stoll, and Pat Hanrahan, “The Design of Parallel Graphics Interface,” Proceedings of SIG- GRAPH, pp. 141-150, 1998.

[9] Mark Segal and Kurt Akeley, “The OpenGL Graphics System : A Specification, “ Mar. 23, 1998.

[10] Mesa library. http://www.mesa3d.org.

[11] S.J.Nam, Y.S.Kwon, B.W.Kim, Y.H.Im, K.G.Kang, and C.M.Kyung, “FLOVA: A Four-issue VLIW Processor with 3D Graphics Acceleration Units,” 10th International Conference on Signal Processing Applications and Technology, Nov. 1999.

[12] S.J.Nam, B.W.Kim, Y.H.Im, Y.S.Kwon, K.G.Kang, J.H.Lee and C.M.Kyung, “VLIW Geometry Processor for 3D Graphics Acceleration,” COOLChips II, pp.107-120, Apr. 1999.

[13] Young-Su Kwon, In-Cheol Park, and Chong-Min Kyung "A Hardware Accelerator for the Specular Intensity of Phong Illumination Model in 3-Dimensional Graphics", to be presented in ASP-DAC 2000.