Graphics Hardware (2004)
T. Akenine-Möller, M. McCool (Editors)

A Programmable Vertex Shader with Fixed-Point SIMD Datapath for Low Power Wireless Applications

Ju-Ho Sohn[†] Ramchan Woo[†] Hoi-Jun Yoo[‡]

Semiconductor System Laboratory, Department of Electrical Engineering and Computer Science,
Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea

Abstract

The real time 3D graphics becomes one of the attractive applications for 3G wireless terminals although their battery lifetime and memory bandwidth limit the system resources for graphics processing. Instead of using the dedicated hardware engine with complex functions, we propose an efficient hardware architecture of low power vertex shader with programmability. Our architecture includes the following three features: I) *a fixed-point SIMD datapath to exploit parallelism in vertex processing while keeping the power consumption low,* II) *a multithreaded coprocessor interface to decrease unwanted stalls between the main processor and the vertex shader, reducing power consumption by instruction-level power management,* III) *a programmable vertex engine to increases the datapath throughput by concurrent operations with main processor. Simulation results show that full 3D geometry pipeline can be performed at 7.2M vertices/sec with 115mW power consumption for polygons using the OpenGL lighting model. The improvement is about 10 times greater than that of the latest graphics core with floating-point datapath for wireless applications in terms of processing speed normalized by power consumption, Kvertices/sec per milliwatt.*

Category and Subject Descriptors: I.3.1 [Computer Graphics]: Hardware Architecture—Graphics Processor C.1.2 [Processor Architecture]: Multiple Data Stream Architectures—Single–instruction–stream, multiple–data–stream processor (SIMD)

1. Introduction

As the mobile electronics market increases rapidly, 3G wireless terminals such as PDAs or mobile phones get more popular. Since these applications have the display devices, the rendering capabilities from simple two-dimensional graphics even to the real time three-dimensional graphics will be necessary to provide more realistic functionalities.

For the wireless applications, the low power consumption is the most important issue because they are driven by batteries. The Advanced RISC Machines (ARM) processor family that has the reduced instruction set computer (RISC) architecture is most widely used as a main platform for the wireless applications because of its high MIPS/Watt [Cla02]. Since there are very limited system resources in terms of computation power and memory bandwidth, the graphics architecture should be designed to consume as little energy as possible on the ARM platform. Moreover, since the users hold the small screens of wireless terminals close to their eyes, the average eye-to-pixel angle is wider than that of a PC system. Therefore we must provide more realistic graphics images with low power consumption in wireless terminals.

Recently, several researches have tried to increase the mobile graphics capabilities in wireless applications. Since the rasterization and texture mapping require more processing complexities than the rest of operations in the 3D graphics pipeline [ODK*00], most of graphics architectures have mainly focused on the rendering pipeline and achieved the efficient graphics performances [AMS03][WCS*03]. However, since relatively little attention has been given to 3D geometry operations, now they become the performance barriers in 3D graphics pipeline. The geometry units of graphics systems in PC and workstations [LKM01][MBD*97] show high performance while consuming much power. For wireless applications, the general-purpose RISC processors with simple integer datapath [WCS*03] or conventional floating-point datapath [KKF*03] were used to process vertex shading operations. However, the simple integer

[†] {sohnjuho, ural}@eeinfo.kaist.ac.kr
[‡] hjyoo@ee.kaist.ac.kr

datapath could not provide the required performance of vertex shading. In conventional floating-point datapath, the performance is also limited due to low operating frequency within allowed budget of limited power consumption.

In this paper, we propose a programmable vertex shader for the low power wireless applications. It can fill the gap between the flexible high performance 3D geometry systems and the low power wireless platforms with limited system resources. The proposed hardware architecture for low power vertex shader has three major features:

1) a fixed-point SIMD datapath that uses an energy-efficient fixed-point arithmetic to exploit data level parallelism in vertex processing while keeping the power consumption low,

2) a multithreaded ARM coprocessor interface - an instruction extension mechanism of ARM platform, which decreases the unwanted stalls between the main processor and the vertex shader, and reduces power consumption by instruction-level power management, and

3) a programmable vertex engine that increases the datapath throughput for streaming vertex data input by concurrent operations with the main processor while saving the external memory bandwidth.

This paper is organized as follows. First, the hardware architecture of the proposed vertex shader is described, and followed by a discussion about the program model. Then the implementation details are presented. After that, simulation results and evaluations are provided. Finally conclusions are made.

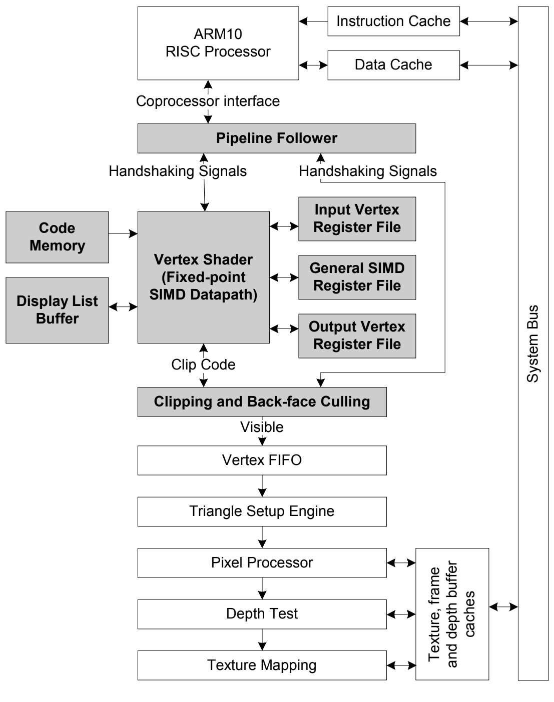

Figure 1: System configuration and block diagram of proposed vertex shader

2. Architecture

Figure 1 shows the system configuration of our graphics architecture and the gray blocks indicate the proposed vertex shader. All hardware blocks including vertex shader are connected to the ARM10 RISC processor through the coprocessor interface, and use system bus interfaces for external memory accesses. In this section, the hardware architecture of the vertex shader is described focusing the three previously mentioned features.

2.1. Fixed-point SIMD Datapath

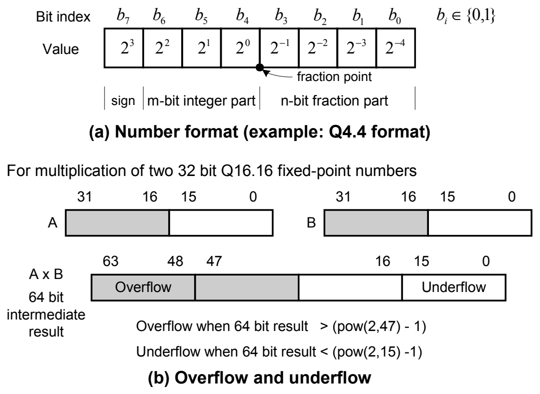
Bit index b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0 $b_i \in \{0,1\}$
Value 2^3 2^2 2^1 2^0 2^{-1} 2^{-2} 2^{-3} 2^{-4}
fraction point
sign
m-bit integer part
n-bit fraction part
(a) Number format (example: Q4.4 format)
For multiplication of two 32 bit Q16.16 fixed-point numbers
31 16 15 0
A
B
A x B 64 bit intermediate result
63 48 47 16 15 0
Overflow
Underflow
Overflow when 64 bit result > (pow(2,47) - 1)
Underflow when 64 bit result < (pow(2,15) -1)
(b) Overflow and underflow

Figure 2: Fixed-point representation

Most 3D graphics systems require real number representation to support various 3D rendering algorithms. We use fixed-point number format for real number representation as shown in Figure 2.(a) [Kol02] instead of using floating-point number. There are two reasons for this. First, the hardware architecture of fixed-point arithmetic is much simpler than that of floating-point arithmetic because fixed-point arithmetic uses only integer datapath. Therefore the fixed-point unit can occupy less silicon area, and it can consume less power. Second, because fixed-point unit can operate at higher clock speed, it can process the same task faster than floating-point unit. Therefore, we can use less energy when applying fixed-point arithmetic in the graphics operations. For matrix multiplication, 32-bit fixed-point hardware consumes 17% less power than single precision floating-point multiplier. It can also operate at 30% higher clock frequency.

To evaluate the accuracy of the fixed-point arithmetic in the 3D geometry operations, the following equations can be used to decide the number of bits for fractional part of Qm.n fixed-point number [HV01], where the 'm' is the number of bits representing integer part and 'n' is the number of bits representing fractional part.

for transformation,

$$n_f = n_a + 3 + \left\lceil \log_2\left(1 + \frac{\text{distance of far plane from eye}}{\text{distance of scene vertex to eye}}\right) \right\rceil$$

for lighting,

$$n_f = n_a + 8 \quad \text{or} \quad n_a + 9$$

where 'n_f' is the number of fractional bits to get n_a bits of accuracy after transformation and lighting calculations.

The screen resolution and color depth of wireless terminals are relatively small, for example the common displays of today's wireless application such as QVGA has 320 x 240 resolution with 16-bit color depth. In this case, 14 or 15 bits are enough for the fractional part to get pixel level accuracy

Our architecture utilizes 128-bit wide 4-way SIMD instructions, enabling to concurrently process up to four 32 bit fixed-point data elements in a single cycle. To improve the usefulness in fixed-point arithmetic, status registers are applied to indicate the overflows and underflows occurred in the multiplications of two fixed-point numbers as shown in Figure 2.(b). These status registers can be used to check errors in fixed-point arithmetic without extra cycle penalties.

In order to enhance the dynamic range further in real number representation, we add controlled ADD/SUB (CAS) instruction and controlled logical shift (CLS) instruction (Figure 3) for software emulation of floating-point arithmetic to fixed-point SIMD instructions. After updating negative flag in arithmetic status register by previous instructions such as SUB, we can make CAS instruction be a single ADD instruction or SUB instruction. We can also make CLS instruction be a single left shift instruction or right shift instruction. These instructions can reduce the unnecessary comparison operations in exponent alignment and normalization of floating-point arithmetic. With floating-point emulation, our architecture shows 160Mflops performance at 400MHz clock speed.

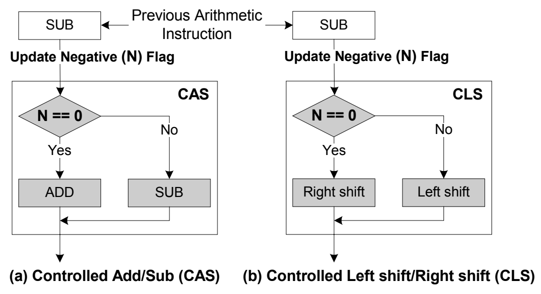

Figure 3: Special instructions for floating-point emulation

2.2. Multithreaded ARM Coprocessor

The ARM architecture supports an extension of the instruction set architecture by adding coprocessors [Fur02]. The proposed architecture is integrated with ARM10 RISC processor. The coprocessor interface is used to interconnect the proposed architecture with main processor because of the following reasons

▪ Since the coprocessor operates in lock step with core pipeline of the main processor, it can avoid a complex synchronization and provide a single context of control.

▪ The coprocessors have the direct signal paths from the main processor. They don't need the bus arbitrations for hardware accesses contrary to conventional hardware accelerators, which use the shared system bus. Therefore, coprocessor interface can reduce the unwanted stalls between main processor and hardware accelerators.

▪ Since the data cache of main processor can be shared to store graphics primitives as well, additional memories are not necessary to store vertex data.

To achieve high graphics performance at low power consumption, we add two additional features to the conventional coprocessor. First, a multithread architecture is proposed to enable the stream processing of vertex data. For this, three independently accessible register files are used for storing input vertex data, output vertex data and temporary results. The vertex shader processes polygon data stored in input vertex register file, and generates the shaded vertex output to the output vertex register file. The general purpose SIMD register file is used to hold the temporary results. Second, the coprocessor can be activated only when it is required on an instruction-by-instruction basis. If the current instruction is not a valid

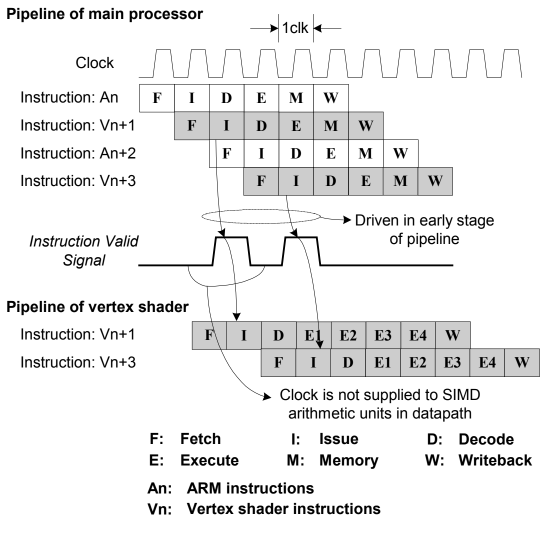

Figure 4: Instruction pipeline and clock gating

coprocessor instruction, clock signal is not supplied to the fixed-point SIMD arithmetic units in datapath. As shown in Figure 4, the instruction valid signal driven in the early stage of the main processor pipeline disables the unnecessary hardware blocks of the coprocessor datapath to reduce power consumption.

2.3. Programmable Vertex Engine

In general, the ARM coprocessor cannot be operated without instruction issuing from the main processor. Therefore, the datapath of the main processor is stalled while the coprocessor executes instructions. Because it leads to the performance degradation of main processor when processing the streaming vertex data, we improve the throughput of main processor and vertex shader by allowing coprocessor to execute the vertex program independently.

Figure 5 shows the internal architecture of the vertex engine. After downloading the vertex program into the internal code memory via the coprocessor interface, the vertex shader starts executing each instruction running free from main processor. As previously mentioned, the vertex shader is a multithreaded coprocessor with fixed-point SIMD datapath. The input vertex register file that is used to hold the vertex attributes such as position and normal vector is fed into the fixed-point SIMD datapath. The vector arithmetic unit is responsible for all arithmetic operations such as addition and multiplication, and the special function unit is responsible for reciprocal (RCP) and reciprocal square root (RSQ). Most of the operations are performed as the 32 bit fixed-point numbers, and achieve a single cycle pipelined throughput. The constants such as transformation matrix, lighting parameters and lookup table entries are stored in the display list buffer and temporary results are stored in the general purpose SIMD register file. The shaded vertex output is transformed into one of the ouput vertex register files. There are three output vertex reigter files for caching of vertex data in the primitive assembly and only one of them is accessible in the vertex program. To save the silicon area, only one input vector operand can be swizzled aribitrarily in the SIMD datapath and all the output writes can be controlled by component-wise write mask bits

In the vertex shader, we have the display list buffer to store graphics primitives, reducing the traffic on external memory I/O. Also, the display list buffer can be shared to hold vertex constants at the same time for design simplicity of hardware. To enhance the efficiency of addressing, we have two integer address registers for indexed display list buffer reads. In addtion, the display list buffer has the following two features.

▪ Auto increment and decrement addressing modes: the address register can be updated automatically after indexed display list buffer reads, which is useful to manage the vertex streams.

▪ 8 bit / 16 bit unpack with shuffling of vector components: For geometry compression, the 8 bit or 16 bit read data from the display list buffer can be automatically sign-extended to the full 32 bit fixed-point numbers, which can be used as the delta difference between one vertex and the next vertex [Dee95].

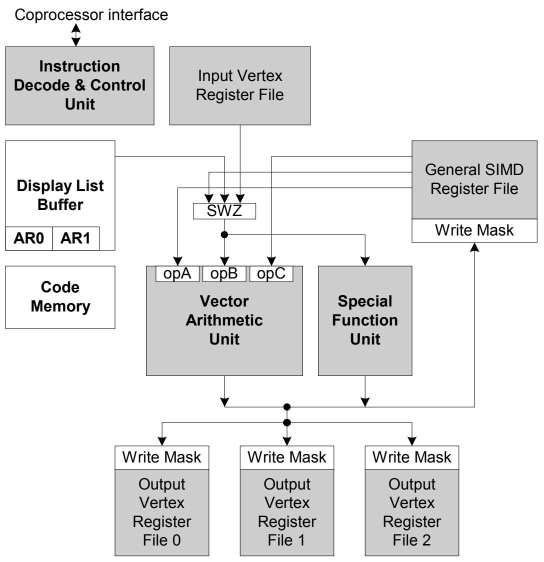
Coprocessor interface
Instruction Decode & Control Unit
Input Vertex Register File
Display List Buffer
AR0
AR1
General SIMD Register File
Write Mask
SWZ
Code Memory
opA
opB
opC
Vector Arithmetic Unit
Special Function Unit
Write Mask
Output Vertex Register File 0
Write Mask
Output Vertex Register File 1
Write Mask
Output Vertex Register File 2

Figure 5: Internal architecture of vertex engine

3. Program Model

In this section, we will describe the program model of the proposed vertex shader including the processor operating states and the instruction set architecture.

3.1. Dual Personality

The main purpose of our design is to use the simple program model through ARM coprocessor interface, while improving the datapath performance with low power consumption for vertex data streams. From the programmer's point of view, the vertex shader can be in one of two states as shown in Figure 6.

1) Tightly coupled coprocessor (TCC): All instruc-

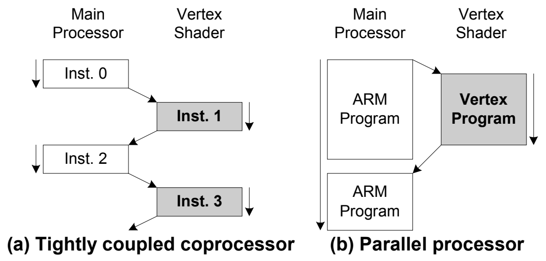

Figure 6: Processor operating states

tions of the vertex shader are issued in the instruction stream of the main processor as extended instructions and, they are executed in lock step with main pipeline.

2) Parallel processor (PP): In this state, vertex shader can process the vertex program stored in the internal code memory without instruction issues from main processor. The vertex shader can achieve the maximum throughput of datapath because it can operate concurrently with main processor [Gan98]. The main processor reads the vertex attributes in the main memory and feeds them into the vertex shader while vertex program is executed. To maintain the communication protocol of the ARM coprocessor interface, the vertex shader drives the coprocessor busy signals to the main processor, leading next coprocessor instruction to busy-wait loop for system synchronization.

3.2. Instruction Set

We use two kinds of instruction set architectures for each processor operating state. The instruction set in TCC state contains all data processing and transfer instructions for vertex shader. In PP state the instruction set consists of 20 operations, which are the modified subset of the previous programmable vertex engine [LKM01]. Table 1 shows the instruction set of vertex program that can be executed in PP state.

Opcode	Full name	Description	Latency	Throughput
MUL	Multiply	Vector→Vector	4	1
MAD	Multiply and add	Vector→Vector	4	1
DP3	3 term dot product	Vector→Replicated scalar	5	2
DP4	4 term dot product	Vector→Replicated scalar	5	2
TRFM	Transform	Vector→Vector	7	4
ADD	Addition	Vector→Vector	1	1
SUB	Subtraction	Vector→Vector	1	1
MOV	Move	Vector→Vector	1	1
RCP	Reciprocal	Scalar→Replicated scalar	6	3
RSQ	Reciprocal square root	Scalar→Replicated scalar	8	5
MIN	Minimum	Vector→Vector	1	1
MAX	Maximum	Vector→Vector	1	1
SLT	Set on less than	Vector→Vector	1	1
SGE	Set on grater or equal	Vector→Vector	1	1
SEQ	Set on equal	Vector→Vector	1	1
LSL	Logical shift left	Vector→Integer Vector	1	1
ASR	Arithmetic shift right	Vector→Integer Vector	1	1
ZERO	Set zero	Vector	1	1
ARL	Address register load	Vector→Scalar integer	2	2
END	Vertex program end	Miscellaneous	1	1

Table 1: Instruction set of vertex program

3.2.1 Control flow instructions

In TCC state, the control flow instructions such as branch and return are managed by the main processor and the vertex shader executes only the extended SIMD arithmetic instructions. However all vertex shader instructions are conditionally executed to maximize execution throughput. They affect on the memory and registers only if the arithmetic flags (negative, zero, carry out and overflow) satisfy a condition specified in the instruction. When implementing the fixed function pipeline of graphics library such as OpenGL which is controlled by global states, the state checking, vertex shading path selection, homogeneous clip space operations and back face culling are handled in TCC state. The remaining code segments for actual vertex shading operations can be executed without state checking. These operations are carried out in the vertex program of PP state. It supports the vertex transform paths without branching for simplicity and efficiency of hardware architecture. Even if the control flow instructions are not supported in PP state, simple if/then/else statement is still possible through SLT, SGE and SEQ instructions.

3.2.2 Fixed-point SIMD arithmetic instructions

To save the system resources, we make the datapath simple and efficient without complex hardware blocks. All arithmetic operations including RCP and RSQ are executed on the fixed-point numbers that can have any precisions, using only low power interger arithmetic units. They are also fully pipelined and bypassed to forward the data to the different stage of pipeline of correct instructions.

Since the multiplication equivalent instructions consume most of time in vertex shading operations, we make the throughput of MUL and MAD be a single cycle, and support dot products for coding convenience. Moreover we add TRFM instruction to enhance the performance of the vertex transformation in

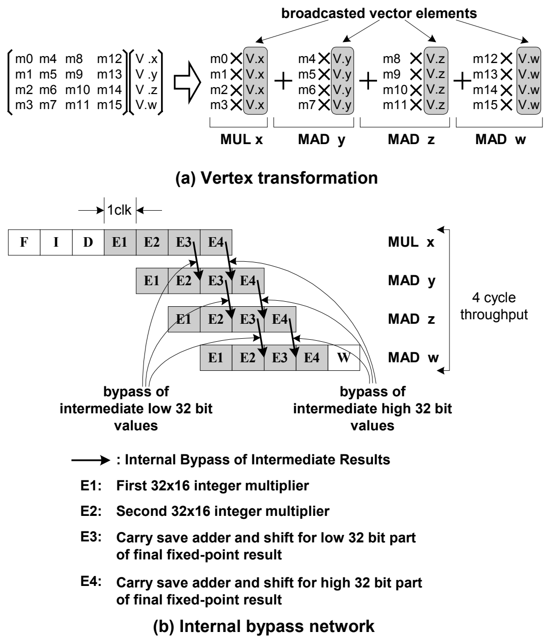
broadcasted vector elements
m0 m4 m8 m12
m1 m5 m9 m13
m2 m6 m10 m14
m3 m7 m11 m15
V.x V.y V.z V.w
m0×V.x m1×V.x m2×V.x m3×V.x
m4×V.y m5×V.y m6×V.y m7×V.y
m8×V.z m9×V.z m10×V.z m11×V.z
m12×V.w m13×V.w m14×V.w m15×V.w
MUL x
MAD y
MAD z
MAD w
(a) Vertex transformation
1clk
F I D E1 E2 E3 E4
E1 E2 E3 E4
E1 E2 E3 E4
E1 E2 E3 E4 W
MUL x
MAD y
MAD z
MAD w
4 cycle throughput
bypass of intermediate low 32 bit values
bypass of intermediate high 32 bit values
: Internal Bypass of Intermediate Results
E1: First 32x16 integer multiplier
E2: Second 32x16 integer multiplier
E3: Carry save adder and shift for low 32 bit part of final fixed-point result
E4: Carry save adder and shift for high 32 bit part of final fixed-point result
(b) Internal bypass network

Figure 7: TRFM instruction

homogeneous coordinates. Although the latency of MUL and MAD are four cycles, TRFM instruction can calculate the vertex transform-ation at every four cycles with the help of broadcasting of vector elements and internal bypass network of intermediate values as shown in Figure 7.

For design simplicity, we remove the complex functions such as the logarithmic, exponential and specular power functions, and rather the table look-up is used for these functions. The integer shift instructions of fixed-point numbers are added in order to extract bit fields for index calculations in the lookup table. After shift operations of vertex specific index, the ARL instruction can allow an offset into the lookup table.

The following vertex program implements the vertex transformation and full Phong shading. It uses OpenGL lighting equations assuming infinite light and viewpoint positions. To calculate the specular power function, we use the lookup table of 64 entries which store the specular coefficients for given shininess value. After calculating dot product of normal vector and the Blinn halfway vector, we use the integer shift instructions for offset values. After the rearrangements of the instructions for compiler optimization, the proposed hardware running at 400MHz can process these vertices at a rate of 7.2M vertex/sec including view frustum clip check, perspective division and viewport transform.

```
# Vertex Transformation and OpenGL Lighting
#
# c[0-3]     = modelview matrix (column-wise)
# c[4-7]     = modelview inverse transpose (column-wise)
# c[8-11]    = modelview projection matrix (column-wise)
# c[16]      = light position
# c[17]      = blinn halfway vector
# c[18]      = precomputed specular light * specular mat.
# c[19]      = precomputed diffuse light  * diffuse mat.
# c[20]      = precomputed ambient light  * ambient mat.
# c[32-47]   = 64 entries lookup table for specular power (column-wise)
# c[48]      = 16th, 32th, 48th and 64th entries of lookup table
# c[49].x    = fraction bit length of fixed-point format
# c[49].y    = fraction bit length - 2
# c[49].z    = fraction bit length - 6
# c[49].w    = fraction bit length + 4
# c[50]      = (0, 1, 2, 3) in integer format

# Vertex transformation to eye space
TRFM VGR0.xyz, VIR[OPOS], c[0];

# Normal vector transform to eye space
TRFM VGR1.xyz, VIR[NRML], c[4];

# Vertex transformation to clip space
TRFM VOR[HPOS], VIR[OPOS], c[8];

# Compute normalized light direction
SUB  VGR0.xyz, VGR0, c[16];
DP3  VGR0.w, VGR0, VGR0;
RSQ  VGR0.w, VGR0.w;
MUL  VGR0.xyz, VGR0, VGR0.w;

# Compute N.L and N.H
DP3  VGR2, VGR1, VGR0;
DP3  VGR3, VGR1, c[17];

# Index calculation of lookup table for specular power function
ASR  VGR4, VGR3, c[49].y;
ASR  VGR5, VGR3, c[49].z;
LSL  VGR6, VGR5, c[49].z;
SUB  VGR6, VGR3, VGR6;
LSL  VGR5, VGR5, c[49].x;
LSL  VGR7, VGR4, c[49].w;
SUB  VGR5, VGR5, VGR7;

# table look-up
ARL  A0.x, VGR5.x;
SEQ  VGR7.x, VGR4, c[50].x;
SEQ  VGR7.y, VGR4, c[50].y;
SEQ  VGR7.z, VGR4, c[50].z;
SEQ  VGR7.w, VGR4, c[50].w;
DP4  VGR4, VGR7, c[A0.x+32];
DP4  VGR5, VGR7, c[A0.x+33];

# Compute specular power using interpolation
SUB  VGR5, VGR5, VGR4
MAD  VGR3, VGR5, VGR6, VGR4;

# Compute light color values
MUL  VGR5.xyz, VGR3, c[18];
MUL  VGR4.xyz, VGR2, c[19];
ADD  VGR5.xyz, VGR4, VGR5
ADD  VOR[COL0].xyz, VGR5, c[20];

# texture coordinate
MOV  VOR[TEX0], VIR[TEX0];
```

4. Implementation

We have developed a fully synthesizable Verilog-HDL model of the proposed graphics architecture including vertex shader and a hardware rasterizer. The vertex shader consists of 230K logic gates and 280Kbit SRAM using 0.13um process. The capacity of display list buffer is 32KByte. The low power rendering engine integrated in our previous work [WCS*03] was applied as a rasterizer. For system evaluations, we designed a software graphics library, a subset of OpenGL, which utilizes the fixed-point arithmetic operations to optimally use the features of implemented hardware architecture. To manage the vertex shader such as downloads of vertex programs and vertex attributes, simple API calls are provided as OpenGL extension. After implementing software test bench using synthesized gate level circuits, we measured the vertex processing speed and power consumptions.

5. Results and Evaluation

In this section, we show hardware simulation results focusing the processing speed and power consumption.

In order to measure the accuracy of fixed-point arithmetic in vertex shading operations, we compare the rendered images of 3D objects using software only floating-point graphics library with our hardware using fixed-point graphics library. In Figure 8, lit, smooth-shaded spheres with different material properties and an animated 3D character are rendered to show reliability of the proposed vertex shader. From the results, we can show that under vertex-level accuracy, the maximum transformed distance between floating-point and fixed-point systems is less than 0.000025 for Q12.20 fixed-point format and 0.0002 for Q16.16 fixed-point format.

Figure 9 shows vertex processing speed and power consumption in case of unlit, diffuse lit and diffuse-specular lit polygons. For diffuse-specular lit polygons, full 3D geometry pipeline can be performed at a rate of 7.2M vertices/sec with 115mW power consumption

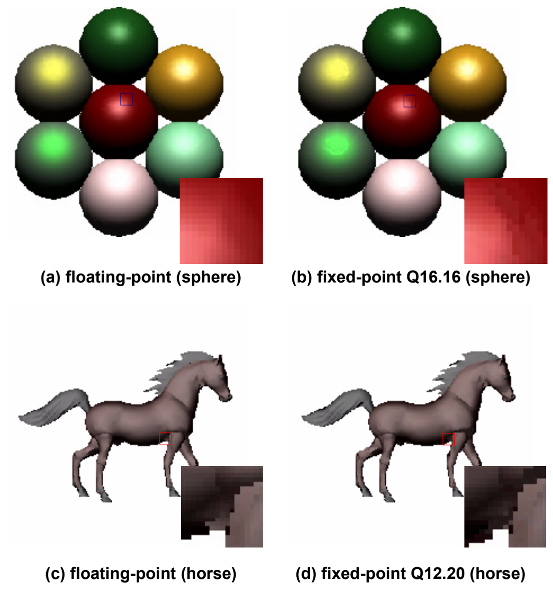
(a) floating-point (sphere)
(b) fixed-point Q16.16 (sphere)
(c) floating-point (horse)
(d) fixed-point Q12.20 (horse)

Figure 8: Fixed-point vertex shading of 3D objects (spheres: 5068 vertices, horse: 6798 vertices)

including host processor, vertex shader and display list buffer. In other cases, the performance was improved with less power consumption because unnecessary hardware blocks such as special function unit and lookup table can be disabled.

Figure 10 shows the relationship between rendered vertex counts and processing speed with various sizes of the vertex data format. The vertex processing speed was measured with and without using the display list buffer for vertex caching. To reduce the external memory bandwidth, vertex data in the display list buffer were stored as a 8 bit or 16 bit data type. In typical wireless applications, a 32-bit SDRAM memory running at 100 MHz is used as the main memory, and gives a usable bandwdith of 200MB/s with approximately 50% of efficiency. We restriced the peak bandwidth requirement of the vertex shader to 100MB/s in order to allocate the remaining bandwidth for other devices. As shown in this figure, the vertex shader shows twice higher performance when using the display list buffer than only data cache of the main processor. Moreover, the performance degradation is merely 10% as vertex counts are increased. Because we can feed vertex data stream continuously during execution of vertex program with the help of concurrent operations of the main processor, we can achieve high sustaining performance.

With high vertex processing speed, the hardware vertex shader in the PC and workstation platforms provide many advanced shading functions. However they consume a great deal of power, more than several tens of watts. Therefore we should use the following performance indices to measure the performance of vertex shader in wireless applications taking into account the power consumption [WYK*02].

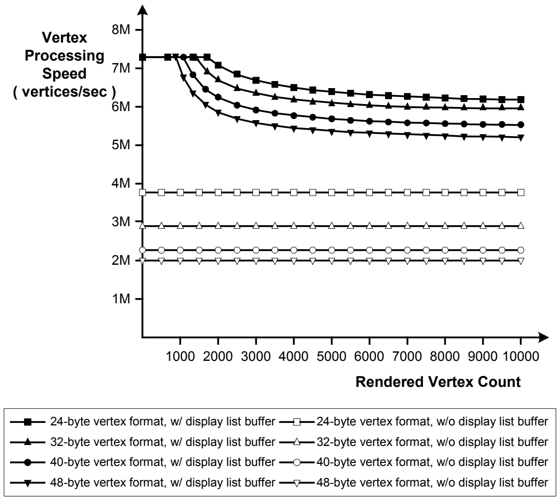

Figure 10: Relationship between rendered vertex counts and processing speed

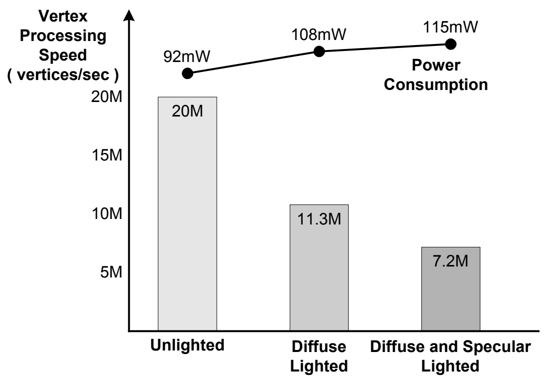

Figure 9: Processing speed and power consumption

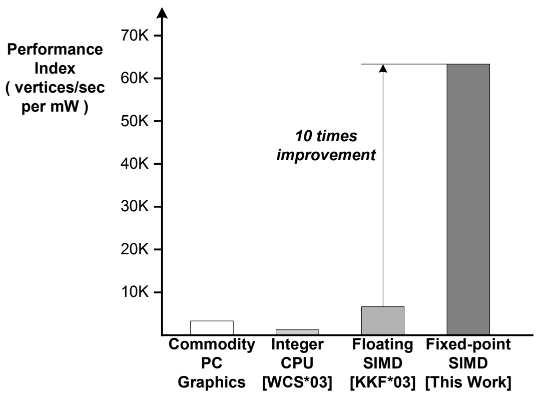

Figure 11: Performance comparison

$$\text{Vertex Shading Performance in Wireless Applications} = \frac{\text{Vertex Processing Speed (vertices/sec)}}{\text{Power Consumption in Vertex Shader(mW)}}$$

Figure 11 shows the performance comparison of proposed vertex shader with different graphics architecture in terms of Kvertices/sec per milliwatt. The improvement is about 10 times greater than that of the latest graphics core with floating-point datapath for wireless applications [KKF*03]. In comparison with simple integer datapath of the conventional low power embedded CPU [WCS*03], it shows far higher performance improvement.

6. Conclusions

We have presentated the design and implementation of a programmable vertex shader with fixed-point SIMD datapth for low power wireless applications. Most graphics architectures for wireless applications have mainly focused on rasterization and texture mapping due to high processing requirements. In order to balance 3D graphics pipeline within the limited system resources, we used simple and efficient programmable architecture for vertex shading instead of using dedicated hardware engine with complex functions. Since main purpose of the proposed design is to provide high performance with low power consumption, we used a energy-efficient fixed-point SIMD datapath to exploit parallelism in vertex shading operations. It shows comparable image quality to floating-point system, while operating at higher clock speed with low power consumption. For simple program model, the proposed vertex shader is connected to main processor through dedicated ARM coprocessor interface. Moreover, using multithredaded architecture, it can handle vertex data stream more efficiently. In addition to conventional ARM coprocessor, our architecture can be operated as a programmable vertex engine and execute vertex program concurrently with the main processor. All instructions are optimized to fixed-point arithmetic.

The architecture has been implemented in fully syntheiszable hardware model. Simulation results show that full 3D geometry pipeline can be performed at a rate of 7.2M vertices/sec with 115mW power consumption for polygons using the OpenGL lighting model. The improvement is about 10 times greater than that of the latest graphics core with floating-point datapath for wireless applications in terms of processing speed normalized by power consumption, or vertices/sec per milliwatt.

In the future, we would like to enhance the fixed-point artihmetic in vertex shading operations, and combine our architecture with low power programmable pixel shader to provide more higher quality of rendering images for low power wireless applications.

References

[AMS03] Akenine-Möller T., Ström J.: Graphics for the masses: A hardware rasterization architecture for mobile phones. *In Proc. SIGGRAPH 2003, pp. 801-808, July, 2003*

[Cla02] Clark D.: Mobile processors begin to grow up. *IEEE Computer, vol. 35, Issue 3, pp. 22-24, March, 2002*

[Dee95] Deering M.: Geometry compression. *In Proc. SIGGRAPH 95, pp. 13-20, July, 1995*

[Fur02] Furber S.: ARM: System-on-chip architecture. *2nd edition Addison-Wesley Press, 2000*

[Gan98] Gandhi P.: SA1500:A 300MHz RISC CPU with attahced media processor. *In Proc. Hot Chips 10, August, 1998*

[HV01] Hao X., Varshney A.: Variable-precision rendering. *In Proc. the 2001 Symp. on Interative 3D Graphics, pp. 149-158, 2001*

[KKF*03] Kameyama M., Kato Y., Fujimoto H., Negishi H., Kodama Y., Inoue Y., Kawai H.: 3D graphics LSI core for mobile phone –Z3D. *In Proc. SIGGRAPH/Eurogrphics Workshop on Graphics Hardware 2003, pp.60-67, August, 2003*

[Kol02] Kolli G. K.: 3D graphics optimization for ARM architecture. *presented at the Game Developer Conf., San Jose, CA. 2002*

[LKM01] Lindholm E., Kilgard M., Moreton H.: A user-programmable vertex engine. *In Proc. SIGGRAPH 2001, pp. 149-158, August, 2001*

[MBD*97] Montrym, J. S., Baum D. R., Dignam D. L., Migdal C. J.: Infinite Reality: A Real-Time Graphics System. *In Proc. SIGGRAPH 1997, pp. 293-302, August, 1997*

[ODK*00] Owens J. D., Dally W. J., Kapasi U. J., Rixner S., Mattson P., Mowery B.: Polygon rendering on a stream architecture. *In Proc. SIGGRAPH/Eurographics Workshop on Graphics Hardware 2000, pp. 23-32, August, 2000*

[WCS*03] Woo R., Choi S., Sohn J., Song S., Bae Y., Yoo H.: A low power and high performance 2D/3D graphics accelerator for mobile multimedia applications. *In Proc. Hot Chips 15, August, 2003*

[WYK*02] Woo R., Yoon C., Kook J., Lee S., Yoo H.: A 120mW 3D rendering engine with a 6-Mb embedded DRAM and 3.2 GB/s run-time reconfigurable bus for PDA chip. *IEEE J. of Solid-State Circuits, vol. 3, No. 10, pp.1352-1355, October, 2002*