Software-Based Geometry Operations for 3D Computer Graphics

Mihai Sima[a], Daniel Iancu[b], John Glossner[b,c], Michael Schulte[d], and Suman Mamidi[d]

[a]University of Victoria, Department of Electrical and Computer Engineering,
P.O. Box 3055 Stn CSC, Victoria, B.C. V8W 3P6, Canada
e-mail: **msima@ece.uvic.ca**
[b]Sandbridge Technologies, Inc., 1 North Lexington Avenue, White Plains, NY 10601, U.S.A.
e-mail: {**DIancu,JGlossner**}**@sandbridgetech.com**
[c]Delft University of Technology, Department of E.E.M.C.S., Delft, The Netherlands
[d]University of Wisconsin-Madison, Department of Electrical and Computer Engineering,
1415 Engineering Drive, Madison, WI 53706, U.S.A.
e-mail: **schulte@engr.wisc.edu**, **mamidi@cae.wisc.edu**

ABSTRACT

In order to support a broad dynamic range and a high degree of precision, many of 3D renderings fundamental algorithms have been traditionally performed in floating-point. However, fixed-point data representation is preferable over floating-point representation in graphics applications on embedded devices where performance is of paramount importance, while the dynamic range and precision requirements are limited due to the small display sizes (current PDA's are 640×480 (VGA), while cell-phones are even smaller). In this paper we analyze the efficiency of a CORDIC-augmented Sandbridge processor when implementing a vertex processor in software using fixed-point arithmetic. A CORDIC-based solution for vertex processing exhibits a number of advantages over classical Multiply-and-Acumulate solutions. First, since a single primitive is used to describe the computation, the code can easily be vectorized and multithreaded, and thus fits the major Sandbridge architectural features. Second, since a CORDIC iteration consists of only a shift operation followed by an addition, the computation may be deeply pipelined. Initially, we outline the Sandbridge architecture extension which encompasses a CORDIC functional unit and the associated instructions. Then, we consider rigid-body rotation, lighting, exponentiation, vector normalization, and perspective division (which are some of the most important data-intensive 3D graphics kernels) and propose a scheme to implement them on the CORDIC-augmented Sandbridge processor. Preliminary results indicate that the performance improvement within the extended instruction set ranges from $3\times$ to $10\times$ (with the exception of rigid body rotation).

1. INTRODUCTION

Multimedia applications for 3D graphics such as computer animation, video games, medical imaging, scientific visualization and simulation, virtual reality, CAD tools etc. have tremendous computational requirements. To support high-speed computations, 3D applications have traditionally been implemented in specialized hardware accelerators such as geometry and rasterization engines. The geometry engine, which is also referred to as a *vertex processor*, executes the geometric transformation and performs the lighting calculation for each vertex. The rasterization engine, which is also referred to as a *fragment processor*, performs pixel value calculation, texture mapping, scan conversion, shading and hidden surface removal. However, such dedicated hardware is expensive and not flexible; since a different full-custom circuit is needed for each distinct task, a considerable engineering effort is needed to keep pace with the rapid evolving standards in the 3D graphics application domain.

To alleviate the lack of flexibility of dedicated hardware, the newer graphics processors, such as the GeForce series by nVIDIA[1] and the Radeon series by ATI,[2] support more geometry processing operations at the front-end of the 3D graphics pipeline with increased programmability. For example, GeForce3[1] is a multi-threaded vector processor operating on quad-float data. A similar engine has been proposed by Ide et al..[3] It is a VLIW processor that consists mainly of a vector unit with four floating-point Multiply-and-Accumulate units and one floating-point division and square-root unit. Aside of building graphics processors, there is the approach of augmenting the instruction set of a general-purpose processor with

Multimedia on Mobile Devices II, edited by Reiner Creutzburg, Jarmo H. Takala, Chang Wen Chen,
Proc. of SPIE-IS&T Electronic Imaging, SPIE Vol. 6074, 60740B, © 2006 SPIE-IS&T · 0277-786X/06/$15

Downloaded from SPIE Digital Library on 21 May 2010 to 131.180.130.114. Terms of Use: http://spiedl.org/terms

instructions 3D graphics. The PLX FP extension belongs to this class.[4] Our approach is slightly different; we propose to use the Sandbridge multi-threaded vector processor augmented with a CORDIC unit for 3D graphics. Using the CORDIC algorithm for 3D graphics has been suggested by Lang and Antelo.[5] Since the 3D operations are of a geometric nature, it is natural to express them using CORDIC-type primitives. One of the advantage of using CORDIC is that a single primitive is used to describe the computations, and as such the implementation is highly scalable.

Each of the mentioned 3D engines is at the core of a high-end graphics processor. To provide a large computational resolution and dynamic range, a floating-point format is necessary. However, there are systems for which the performance is of paramount importance, while the dynamic range and precision requirements are limited due to the small display sizes (for example, current PDA's are 640×480, while the cell-phones are even smaller). If the dynamic range and precision requirements can be constrained , then fixed-point arithmetic can be as accurate as and much faster than floating-point arithmetic. A low-end graphics engine that uses 16-bit, fixed-format is more than adequate for such applications.

In this paper we describe a fixed-point implementation of the vertex processor operations using the COordinate Rotation DIgital Computer (CORDIC) instruction for the Sandbridge multithreaded processor. In particular, we will address rigid-body rotation, lighting, exponentiation, vector normalization, and perspective division (which are some of the most important data-intensive 3D graphics kernels). A CORDIC-based solution for vertex processing exhibits a number of advantages over classical Multiply-and-Acumulate solutions. First, since a single primitive is used to describe the computation, the code can easily be vectorized and multithreaded, and thus fits the major Sandbridge architectural features. Second, since a CORDIC iteration consists of only a shift operation followed by an addition, the computation may be deeply pipelined. Third, the CORDIC algorithm produces one bit of accuracy per iteration. Thus, all CORDIC-based rotations will have the same latency for a given precision, which allows trade-offs to be made between precision and latency at run-time.

The paper is organized as follows. For background purposes, we outline the CORDIC algorithm in Section 2 and the 3D graphics pipeline in Section 3. Sections 4 and 5 outlines the Sandbridge multithreaded processor and the CORDIC architectural extension. The execution scenario of the 3D graphics pipeline within the extended instruction set along with experimental results is discussed in Section 6. Section 7 completes the paper with some conclusions and closing remarks.

2. COORDINATE ROTATION DIGITAL COMPUTER

A Givens transformation[6] is a 2-by-2 orthogonal matrix $R(\theta)$ of the form described in Equation (1). It can be observed that multiplication by $R(\theta)$ of a vector $[x, y]^T$ amounts to a counterclockwise rotation of θ radians in plane.

$$R(\theta) \cdot \begin{bmatrix} x \\ y \end{bmatrix} \equiv \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix} \tag{1}$$

Historically, the Givens transformation has been used in QR factorization,[7] since it can zero matrix elements selectively. Clearly, by setting

$$\cos\theta = \frac{x}{\sqrt{x^2+y^2}}, \quad \sin\theta = \frac{y}{\sqrt{x^2+y^2}} \tag{2}$$

it is possible to force the second entry in the vector $[x, y]^T$ to zero:

$$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \sqrt{x^2+y^2} \\ 0 \end{bmatrix} \tag{3}$$

The Givens transformation is computationally demanding. For example, given an arbitrary angle θ, the direct evaluation of the rotation (Equation (1)) requires four multiplications, two additions, and a large memory storing the cosine and sine tables. Also, finding the angle θ which satisfies the trigonometric Equations (2), translates to a sequence of multiplications, additions, and memory look-up operations if the common Taylor series expansion is employed.

COordinate Rotation DIgital Computer (CORDIC) is an iterative method performing vector rotations by arbitrary angles using only shifts and additions. The main idea is to first split the rotation angle θ into a sequence of subrotations of angles $\theta(n)$, where the rotation for iteration n is

Downloaded from SPIE Digital Library on 21 May 2010 to 131.180.130.114. Terms of Use: http://spiedl.org/terms

$$\begin{bmatrix} x(n+1) \\ y(n+1) \end{bmatrix} = \begin{bmatrix} \cos\theta(n) & \sin\theta(n) \\ -\sin\theta(n) & \cos\theta(n) \end{bmatrix} \cdot \begin{bmatrix} x(n) \\ y(n) \end{bmatrix} \tag{4}$$

Then, the rotation matrix $R(\theta(n))$ is written as

$$R(\theta(n)) = \cos\theta(n) \cdot \begin{bmatrix} 1 & \tan\theta(n) \\ -\tan\theta(n) & 1 \end{bmatrix} \tag{5}$$

and the rotation angles are restricted so that $\tan\theta(n) = \pm 2^{-n}$. This way, the multiplication by the tangent factor is reduced to simple shift operations.

Arbitrary rotation angles can be obtained by performing a series of successively smaller elementary rotations. If the decision at each iteration, n, is which direction to rotate rather than whether or not to rotate, then the factor $\cos\theta[n]$ becomes a constant for the current iteration (since $\cos\theta[n] = \cos(-\theta[n])$). Then, the product of all these cosine values is also a constant and can be applied anywhere in the system or treated as system processing gain.

The angle of a composite rotation is uniquely defined by the sequence of the directions of the elementary rotations. That sequence can be represented by a decision vector. The set of all possible decision vectors is an angular measurement system based on binary arctangents. Conversions between this angular system and any other can be accomplished using an additional adder-subtractor that accumulates the elementary rotation angles at each iteration. The elementary angles are supplied by a small look-up table (one entry per iteration), or are hardwired, depending on the implementation. The angle accumulator adds a third difference equation to the CORDIC algorithm.

$$z(n+1) = z(n) - d(n)\arctan\left(2^{-n}\right) \tag{6}$$

The CORDIC rotator is operated in one of two modes: rotation or vectoring. [8]

- In **rotation mode**, the angle accumulator is initialized with the desired rotation angle. The rotation decision at each iteration is made to diminish the magnitude of the residual angle in the angle accumulator.
- In **vectoring mode**, the CORDIC unit rotates the input vector through whatever angle is necessary to align the result vector with the x axis. The result of the vectoring operation is a rotation angle and the scaled magnitude of the original vector (the x component of the result).

Using CORDIC, a large number of transcendental functions, e.g., polar to cartesian or cartesian to polar transformations, can be calculated with the latency of a serial multiplication. By providing an additional parameter, the basic CORDIC method can be generalized to perform rotations in a linear or hyperbolic coordinate system, [9] thus providing a more powerful tool for function evaluation. For example, the exponential and the logarithm functions can be evaluated with the speed of a serial multiplier. Of particular importance for this paper is CORDIC operating in vectoring mode in the linear coordinate system, since it provides a method for evaluating ratios.

3. THE 3D GRAPHICS RENDERING PIPELINE

The process of converting the geometric description of a 3D model to a 2D image to be displayed on a monitor is referred to as 3D graphics rendering. The 3D graphics pipeline is thus the underlying tool for this real-time process. The 3D graphics pipeline itself consists of two distinct stages: **geometric transformation** and **rasterization**, where each of these stages is a pipeline in itself. Rasterization is not of interest for this paper; therefore, it is not discussed any longer. The sequence of steps involved in geometric transformation is presented in Figure 1, and consists of the following stages [10]:

- *Model and viewing transformation* positions the primitives in space. It can be described by a multiplication of the vertex coordinate by a 4×4 matrix.

Downloaded from SPIE Digital Library on 21 May 2010 to 131.180.130.114. Terms of Use: http://spiedl.org/terms

- *Lighting* evaluates the colour of the vertices given the direction of light, the vertex position, the surface normal vector. It requires the calculation of the sine and the cosine of the angles between the light, normal, view, and tangent unit vectors on the surface.
- *Projection transformation* which projects objects onto the screen. It can also be described by a multiplication by a 4×4 matrix.
- *Clipping* removes the objects that are outside the viewable area. It requires 6 comparisons per vertex.
- *Perspective division* divides the x, y, z by w to convert the vertex to Cartesian coordinates.
- *Viewport mapping* maps the vertices from the projected coordinate system to the viewport on the computer screen.

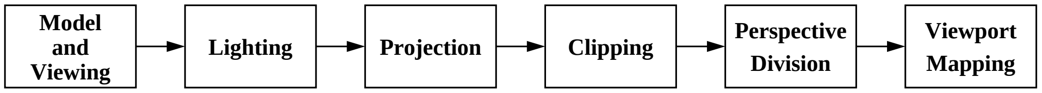

Figure 1. Geometric transformation stage of the 3D graphics pipline – adapted from Mitra and Chiueh.[10]

Out of these stages a number of computationally demanding operations emerges. It is generally accepted that these are *lighting*, *exponentiation*, *vector normalization*, *perspective division*, and *rigid body rotation*.[4,5,11] These are the operations which we will address in a subsequent section.

4. OVERVIEW OF THE SANDBRIDGE PROCESSOR

In this section we describe the most important issues of the Sandbridge architecture and microarchitecture. In particular, our emphasis will be on the multi-threading capability and SIMD-style Vector Unit.

4.1. Sandbridge processor

Sandbridge Technologies has designed a multithreaded processor capable of executing DSP, embedded control, and Java code in a single compound instruction set optimized for handset radio applications.[12–14] The Sandbridge Sandblaster design overcomes the deficiencies of previous approaches by providing substantial parallelism and throughput for high-performance DSP applications, while maintaining fast interrupt response, high-level language programmability, and low power dissipation.

The Sandbridge processor[12–14] is partitioned into three units; an instruction fetch and branch unit, an integer and load/store unit, and a SIMD-style vector unit. The design utilizes a unique combination of techniques including hardware support for multiple threads, SIMD vector processing, and instruction set support for Java code. Program memory is conserved through the use of powerful compounded instructions that may issue multiple operations per cycle. The resulting combination provides for efficient execution of DSP, control, and Java code. The instructions to speed up CORDIC operations are executed in the Sandbridge Vector Unit described in Subsection 4.4.

4.2. Sandbridge pipeline

The pipelines are different for various operations as shown in Figure 2. The Load/Store (Ld/St) pipeline has nine stages. The integer and load/store unit has two execute stages for Arithmetic and Logic Unit (ALU) instructions and three execute stages for integer multiplication (I_MUL) instructions. A *Wait* stage for the ALU and I_MUL instructions causes these instructions to read from the general-purpose register file one cycle later than Ld/St instructions. This helps reduce the number of register file read ports. The vector multiplication (V_MUL) has four execute stages – two for multiplication and two for addition. It should be noted that once an instruction from a particular thread enters the pipeline, it runs to completion. It is also guaranteed to write back its result before the next instruction from the same thread reads the result.

Downloaded from SPIE Digital Library on 21 May 2010 to 131.180.130.114. Terms of Use: http://spiedl.org/terms

Ld/St	Inst Dec	RF Read	Agen	XFer	Int Ext	Mem 0	Mem 1	Mem 2	WB
ALU	Inst Dec	Wait	RF Read	Exec 1	Exec 2	XFer	WB		
I_Mul	Inst Dec	Wait	RF Read	Exec 1	Exec 2	Exec 3	XFer	WB	
V_Mul	Inst Dec	VRF Read	Mpy1	Mpy2	Add1	Add2	XFer	VRF WB	

Figure 2. Sandbridge pipeline.

V_Mul	Inst Dec	VRF Read	Mpy1	Mpy2	Add1	Add2	XFer	WB			
V_Mul									Inst Dec	VRF Read	Mpy1

Figure 3. Two consecutive Vector Multiply instructions that issue from the same thread.

4.3. Sandbridge multithreading

The Sandblaster architecture supports multiple concurrent program execution by the use of hardware thread units. Multiple copies (e.g., banks and/or modules) of memory are available for each thread to access. The Sandblaster processor uses the Token Triggered Threading (T^3) form of interleaved multithreading,[13] in which each thread is allowed to simultaneously execute an instruction, but only one thread may issue an instruction on a cycle boundary. The microarchitecture currently supports up to eight concurrent hardware threads. Multi-threading effectively hides true dependencies which typically occur in connection with long-latency operations.

4.4. The vector processing unit

The Vector Processing Unit (VPU) consists mainly of four Vector Processing Elements (VPEs), which perform arithmetic and logic operations in SIMD fashion on 16-bit, 32-bit, and 40-bit fixed-point data types. High-speed 64-bit data busses allow each PE to load or store 16 bits of data each cycle in SIMD fashion. Support for SIMD execution significantly reduces code size, as well as power consumption, since multiple sets of data elements are processed with a single instruction.[15]

Most SIMD vector instructions go through eight pipeline stages. For example, a vector MAC (V_MAC) instruction goes through the following stages: Instruction Decode, Vector Register File (VRF) Read, Mpy1, Mpy2, Add1, Add2, Transfer, and Write Back. The Transfer stage is needed due to the long wiring delay between the bottom of the VPU and the VRF. Since there are eight cycles between when consecutive instructions issue from the same thread, results from one instruction in a thread are guaranteed to have written their results back to the VRF by the time the next instruction in the same thread is ready to read them. Thus, the long pipeline latency of the VPEs is effectively hidden, and no data dependency checking or bypass hardware is needed. This is illustrated in Figure 3, where two consecutive vector multiply instructions issue from the same thread. Even if there is a data dependency between the two instructions, there is no need to stall the second instruction, since the first instruction has completed the Write Back stage before the second instruction enters the VRF Read stage.

5. AN ARCHITECTURAL EXTENSION FOR SANDBRIDGE PROCESSOR

The instructions investigated are `CFG_CORDIC` that configures the CORDIC unit in one of the execution modes (rotation, vectoring) and one of the coordinate systems (circular, linear, hyperbolic), and `RUN_CORDIC` which launches the configured CORDIC operation. Assuming that 16-bit precision is needed, then the CORDIC algorithm reads in two 16-bit arguments and produces two 16-bit results. If not all the CORDIC iterations can be performed by a single `RUN_CORDIC` call, then the angle and iteration number must be saved between successive `RUN_CORDIC` calls.

The proposed `RUN_CORDIC` instruction is a vector instruction that goes through eight pipeline stages; that is, the execution itself has a latency of 4 thread cycles. The CORDIC functional unit can perform a single CORDIC iteration in a thread

Downloaded from SPIE Digital Library on 21 May 2010 to 131.180.130.114. Terms of Use: http://spiedl.org/terms

cycle (one addition and one shift), and is shared by four SIMD units. Consequently, `RUN_CORDIC` will execute 16 times, i.e., it will take up 16 instruction cycles, for a 16-bit precision, and will perform 4 conversions in SIMD style.

This is the result where the CORDIC unit is added to the vector unit by adding one adder, one shifter, a comparator and some control logic to the existing pipeline. Deploying an autonomous CORDIC unit for each SIMD unit of each thread will translate into both a memory bandwidth problem and a hardware problem, i.e., too much added hardware (32 adders, 32 shifters, 32 comparators), and the operands cannot be fetched from, or the results cannot be stored back to memory anyway.

The CORDIC instructions are defined as follows.

- `CFG_CORDIC`

  for(i=0; i<4; i++) {

  - Read 8 bits of configuration data
  - Configure the CORDIC unit:
    - Mode (1 bit): rotation or vectoring
    - Coordinate system (2 bits): circular, linear, or hyperbolic
    - Iteration Identifier (5 bits): ranges from 0 to 31

  }

- `RUN_CORDIC`

  for(i=0; i<4; i++) {

  - Reads in the first 32-bit vector register packing:
    - 16-bit modulus and 16-bit angle for rotation mode
    - 16-bit x-value and 16-bit y-value for vectoring mode
  - Reads in the second 32-bit vector register storing:
    - 16-bit angle for vectoring mode
  - Performs one CORDIC iteration (one addition and one shift)
  - Writes back one 32-bit vector register packing:
    - 16-bit x-value and 16-bit y-value for rotation mode
    - 16-bit modulus and 16-bit angle for vectoring mode

  }

6. VERTEX PROCESSOR EXECUTION SCENARIO

3D applications usually use polygonal primitives (e.g., triangles) to represent objects in application database. Geometry procesing only operates on vertices, while the rasterization stage takes those transformed vertices and fills in the interiors of polygons. The 3D graphics pipeline is computationally intensive, but is amenable to parallel implementation. The Sandbridge parallel architecture shows its real advantage when a large amount of data is to be processed. Concerning 3D graphics, this requirements translates to a large number of vertices per polygon. Yang et al. [11] analized a set of benchmarks from Viewperf[16] and determined that *Awadvs* has only 3.4 vertices per glBegin/glEnd pair. That means that only 3 out of four slots in the 4-way SIMD CORDIC will be filled-in with operations. All the subsequent estimations are carried out assuming this worse case scenario.

Downloaded from SPIE Digital Library on 21 May 2010 to 131.180.130.114. Terms of Use: http://spiedl.org/terms

6.1. Lighting

During lighting stage the sine and the cosine of angles θ between 3D unit vectors are to be computed.[17] Assuming $v_1 = [x_1\ y_1\ z_1]$ and $v_2 = [x_2\ y_2\ z_2]$ are two units vectors, calculating $\cos\theta$ is equivalent to calculating the dot product as in Equation 7.

$$\cos\theta = x_1 \cdot x_2 + y_1 \cdot y_2 + z_1 \cdot z_2 \tag{7}$$

To calculate $\sin\theta$, a multiplication and an addition followed by a 32-bit square root operation are carried out:

$$\sin\theta = \sqrt{1 - \cos^2\theta} \tag{8}$$

As opposed to Equation 7, Equation 8 is not easily vectorizable. This is due to the fact that division is essentially a sequential algorithm and a square root unit is not available.

The strategy to calculate $\cos\theta$ and $\sin\theta$ by using CORDIC is similar to the one proposed by Lang and Antelo.[5] Basically, a rotation in 3D is emulated by two rotations in 2D. From the computation point of view, this strategy can be summarized as in Figure 4. First, two CORDIC rotations are carried out to align one of the vectors with the *x* axis. This gives $\cos\theta$. Then, three CORDIC rotations align the second vector with the yOz plane. This gives $\sin\theta$.

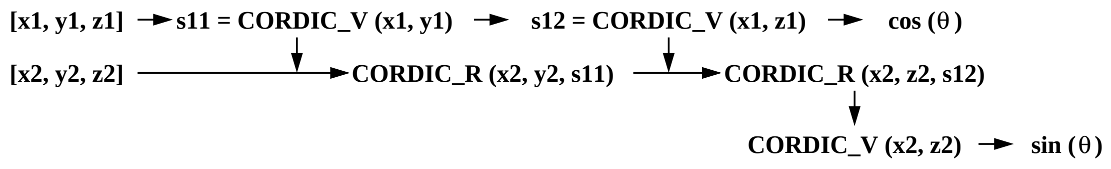

Figure 4. Flowchart of CORDIC-based lighting computation.

Assume 16-bit precision: it requires 16 vector instructions to complete a CORDIC rotation. Therefore, 90 vector instructions are used to calculate the cosine and the sine of an angle between two 3D unit vectors. It is easy to figure out that approx. $5 \times 16 = 90$ instructions are needed to calculate the cosine and the sine. This figure translates to 45 cycles per cosine or sine. For SIMD-style processing, 15 cycles per cosine or sine per vertex are encountered for the *Awadvs* benchmark. Compared to the pure software solution which requires 132 cycles, the CORDIC-based approach provides a speed-up of almost $9\times$.

6.2. Exponentiation

Exponentiation $d = a^b$ is a key operation in 3D graphics. The implementation strategy is similar in both the pure software and CORDIC-based approach: $a^b = e^{b \ln a}$. In software, exponentiation is a very expensive operation, as a sequence of MAC operations are needed to implement a power series expansion. Assuming a Taylor series expansion, for example, 11 terms are needed for 16-bit precision, which translates into 40 MAC instructions. Using CORDIC, 36 instructions are needed for the same precision (16 for the hyperbolic vectoring mode to calculate the natural logarithm, 16 for the hyperbolic rotation mode to calculate the exponential function, and some glue instructions). This translates into 12 instructions per vertex for the *Awadvs* benchmark.

6.3. Vector normalization

Vector normalization consists of a dot product, a reciprocal square root and vector scaling:

$$\mathbf{V}' = \begin{pmatrix} x' & y' & z' & 0 \end{pmatrix}^T = \mathbf{V}/|\mathbf{V}| = \begin{pmatrix} x/w & y/w & z/w & 1 \end{pmatrix}^T / \sqrt{x^2 + y^2 + z^2} \tag{9}$$

A pure software implementation requires three multiplications, two additions, one square root and one division. Since a 32-bit square root and a 16-bit division take 130 and 50 instructions, respectively, 185 instructions including overhead are needed to perform vector normalization in pure software.

Downloaded from SPIE Digital Library on 21 May 2010 to 131.180.130.114. Terms of Use: http://spiedl.org/terms

Kernel	pure software (cycles)	CORDIC-based (cycles)
Lighting	132	15
Exponentiation	40	12
Vector normalization	185	19
Perspective division	50	16
Rigid body rotation	12	n/a

Table 1. 3D graphics figures per vertex for *Awadvs* benchmark.

The CORDIC approach is as follows.[5] First, two CORDIC vectoring rotations are needed to calculate the angles with axes *y* and *z*. Then, a rotation of a unit vector along the *x* axis with the same angles will generate a unit vector (that is, normalized) having the same direction as the initial vector. Thus $4 \times 16 = 64$ instructions are needed for 16-bit precision. For the *Awadvs* that has 3.4 vertices per glBegin/glEnd pair on average, 19 instructions per vertex are needed for a vector normalization task. This represents an improvement of $9.7\times$ versus the pure software solution.

6.4. Perspective division

It consists of a division of a vector by a scalar:

$$\mathbf{V}' = \begin{pmatrix} x' & y' & z' & w' \end{pmatrix}^T = \mathbf{V}/w = \begin{pmatrix} x/w & y/w & z/w & 1 \end{pmatrix}^T \tag{10}$$

CORDIC unit operating in the linear vectoring mode implements essentially the non-restoring division algorithm. Although this is a sequential division, it can be vectorized when CORDIC is used. The performance of the CORDIC-based perspective division is 16 instructions per vertex. The performance of the perspective division in software is 50 instructions per vertex. Since the software division cannot be vectorized, the performance improvement of the CORDIC-based solution versus the pure software solution is significant: $3\times$.

6.5. Rigid body rotation

It is described by a 4×4 matrix:

$$\mathbf{V}' = \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \tag{11}$$

This operation does not pose any problem for Sandbridge processor. Since this operation fits perfectly the Sandbridge architecture, only 12 instructions are needed to perform the rigid body rotation described by Equation 11. Thus, CORDIC is not needed for rigid body rotation.

6.6. Experimental results – synoptic table

The computing performance has been evaluated for a pure software solution and also when CORDIC operation benefits from customized instruction set. The experimental figures are summarized in Table 1.

We would like to note that although CORDIC is essentially a sequential algorithm (it can compute a number of functions in a serial way, one bit per iteration), it has the very important property of being vectorizable and pipelineable. This explains the very good performance provided by the 4-way CORDIC unit when doing 3D graphics kernels over the pure software solution. The improvement ranges from $3\times$ to $10\times$. Given the fact that Sandbridge is a multi-threaded DSP-oriented processor, such an improvement within 3D graphics processing domain indicates that extending the Sandbridge instruction set with CORDIC instructions is a promising approach.

Downloaded from SPIE Digital Library on 21 May 2010 to 131.180.130.114. Terms of Use: http://spiedl.org/terms

7. CONCLUSIONS

We have analyzed the performance improvement provided by a CORDIC functional unit attached to a Sandbridge processor when 3D computer graphics is performed. Preliminary results indicate a performance improvement over the base instruction set architecture that ranges from $3\times$ to $10\times$. As future work, we intend to address the entire 3D graphics pipeline and to evaluate the overall system improvement for the CORDIC-augmented Sandbridge processor.

REFERENCES

1. E. Lindholm, M. J. Kilgard, and H. Moreton, "A User-Programmable Vertex Engine," in *Proceedings of the ACM SIGGRAPH 2001*, pp. 149–158, ACM, (Los Angeles, USA), August 2001.
2. ATI Technologies, Inc., "Radeon family of graphics processors." **`http://www.ati.com/`**.
3. N. Ide, M. Hirano, Y. Endo, S. Yoshioka, H. Murakami, A. Kunimatsu, T. Sato, T. Kamei, T. Okada, and M. Suzuoki, "2.44-GFLOPS 300-MHz Floating-Point Vector-Processing Unit for High-Performance 3-D Graphics Computing," *IEEE Journal of Solid-State Circuits* **35**, pp. 1025–1033, July 2000.
4. X. Yang and R. B. Lee, "PLX FP: An Efficient Floating-Point Instruction Set for 3D Graphics," in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME '04)*, **1**, pp. 137–140, IEEE, (Taipei, Taiwan), June 2004.
5. T. Lang and E. Antelo, "High-Throughput CORDIC-Based Geometry Operations for 3D Computer Graphics," *IEEE Transactions on Computers* **54**, pp. 347–361, March 2005.
6. G. H. Golub and C. F. van Loan, *Matrix Computations*, The Johns Hopkins University Press, 2715 North Charles Street, Baltimore, Maryland 21218-4363, 3rd ed., 1996.
7. G. Strang, *Introduction to Linear Algebra*, Wellesley-Cambridge Press, Box 812060, Wellesley, MA 02482, 3rd ed., 2003.
8. J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Transactions on Electronic Computers* **EC-8**, pp. 330–334, September 1959.
9. J. Walther, "A unified algorithm for elementary functions," in *Proceedings of the Spring Joint Computer Conference of the American Federation of Information Processing Societies (AFIPS)*, **38**, pp. 379–385, AFIPS Press, (Arlington, Virginia), 1971.
10. T. Mitra and T. cker Chiueh, "Three-dimensional computer graphics architecture," *Current Science* **78**, pp. 838–846, April 2000.
11. C.-L. Yang, B. Sano, and A. R. Lebeck, "Exploiting Parallelism in Geometry Processing with General Purpose Processors and Floating-Point SIMD Instructions," *IEEE Transactions on Computers* **49**, pp. 934–946, September 2000.
12. J. C. Glossner, E. Hokenek, and M. Moudgill, "Multithreaded Processor for Software Defined Radio," in *Proceedings of the 2002 Software Defined Radio Technical Conference*, **I**, pp. 195–199, (San Diego, California), November 2002.
13. M. J. Schulte, J. C. Glossner, S. Mamidi, M. Moudgill, and S. Vassiliadis, "A Low-Power Multithreaded Processor for Baseband Communication Systems," in *Proceedings of the Third and Fourth International Annual Workshops on Systems, Architectures, MOdeling, and Simulation (SAMOS)*, A. D. Pimentel and S. Vassiliadis, eds., *Lecture Notes in Computer Science* **3133**, pp. 393–402, Springer, (Samos, Greece), July 2004.
14. J. C. Glossner, M. J. Schulte, M. Moudgill, D. Iancu, S. Jinturkar, T. Raja, G. Nacer, and S. Vassiliadis, "Sandblaster Low-Power Multithreaded SDR Baseband Processor," in *Proceedings of the 3rd Workshop on Applications Specific Processors (WASP'04)*, pp. 53–58, (Stockholm, Sweden), September 2004.
15. J. Sebot and N. Drach, "SIMD ISA Extensions: Reducing Power Consumption on a Superscalar Processor for Multimedia Applications," in *IEEE Symposium on Low-Power and High-Speed Chips (Cool Chips) IV*, (Tokyo, Japan), April 2001.
16. Standard Performance Evaluation Corporation, "OpenGL Performance Benchmark Viewperf." **`http://www.specbench.org/`**.
17. T. Akenine-Möller and E. Haines, *Real-time rendering*, CWI Tract, A K Petres, 888 Worcester Street, Suite 230, Wellesley, MA 02482, U.S.A., 2nd ed., 2002.

Downloaded from SPIE Digital Library on 21 May 2010 to 131.180.130.114. Terms of Use: http://spiedl.org/terms