AN OPERATION-SAVING VLSI GEOMETRY ENGINE CORE

K. E. Karagianni *G. Diamantakos* *V. Paliouras* *T. Stouraitis*

Dept. of Electrical and Computer Engineering,
University of Patras,
Patras, 26500, Greece
karagian@ee.upatras.gr

ABSTRACT

A floating point geometry engine core is introduced in this paper. The proposed core is optimized for performing the 3-D geometrical transformations, including the hardware evaluation of $\sin x$ and $\cos x$ functions. The architecture exploits the structure of the transformation matrices, thus reducing the number of floating point operations required per transformation. VLSI chip implementation issues for the specific architecture are also discussed.

1. INTRODUCTION

Computer graphics applications have become very popular over the last twenty years, covering a wide range of areas from data or signal representation and Computer Aided Design to entertainment [1]. To cope with the large amount of data to be processed and the strict real-time requirement, the following alternative hardware platforms may be used:

- general purpose DSPs [2, 3] which offer efficient performance for small applications, at a low cost,
- general purpose graphics processors [4, 5, 6] which are usually sophisticated and expensive, with capabilities that some times are more advanced than required, and,
- ASICs [7] i.e., application specific graphics processors that may be optimized in order to suit the demands of a specific application.

The geometry engine is one of the main parts that are usually met in a graphics processor. Typical operations that may be performed by a geometry engine include geometrical transformations, polygon shading, clipping, etc.

In this paper, a geometry engine core that is optimized for performing the 3-D geometrical transformations is introduced. The proposed architecture exploits the special structure of the data that take part in a geometrical transformation, thus leading to an operation-saving implementation. Furthermore, it is enhanced with a hardware unit for the evaluation of the trigonometric functions.

The remainder of this paper is organised as follows: In Section 2 the elementary 3-D geometrical transformations are described. In Section 3 the architecture of the proposed geometry engine core is presented, and in Section 4 VLSI chip implementation issues are discussed. Finally, the paper is concluded in Section 5.

2. GEOMETRICAL TRANSFORMATIONS

The model of a graphics object includes a set of points, each represented by a homogeneous coordinate vector $\underline{v} = [\ x \quad y \quad z \quad w\ ]^T$.

Each geometrical transformation of a coordinate vector corresponds to a certain mathematical procedure and it can be represented as a symbol $\mathcal{T}$. The transformed coordinate vector is

$$\underline{v}' = \mathcal{T}\underline{v}.$$

The elementary 3-D geometrical transformations are shown below [1].

The $x-$axis rotation matrix is

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The $y-$axis rotation matrix is

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The $z-$axis rotation matrix is

$$R_z(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Translation:

$$T(D_x, D_y, D_z) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ D_x & D_y & D_z & 1 \end{bmatrix}.$$

Scaling:

$$S(S_x, S_y, S_z) = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Two successive transformations may be combined into a new transformation, which gives as a result the same result with the two transformations.

3. THE PROPOSED ARCHITECTURE

The proposed architecture appears in Fig. 1.

The coordinate vector $[\ x\ \ y\ \ z\ \ w\ ]^T$ of a point to be transformed as well as a sequence of transformations $\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3, \ldots$ are the input data in the geometry engine. This assumption makes the system compatible with the PHIGS standard. The sequence of transformations iteratively forms a matrix A, called the transformation matrix, the initial value of which is the unit matrix. Each transformation $\mathcal{T}$ in the series, replaces the matrix A with A', which is computed as

$$A' = \mathcal{T} \cdot A \tag{1}$$

or

$$A' = A \cdot \mathcal{T}. \tag{2}$$

The current value of matrix A is kept into MEM1 of Fig. 1. Each coordinate input vector is transformed through matrix A whenever the appropriate instruction is executed.

The main novelty in the proposed approach, through which a significant reduction in the number of operations is achieved, is in the evaluation of (1) and (2). In particular, products (1) and (2) are not treated as 4×4 general matrix products, which require 64 element-wise multiplications, as in other geometry engines of the literature [7], but as an update of the transformation matrix, which requires an operation to be applied only on some of the elements of the matrix. More specifically, suppose that the present state of the matrix A is

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & 1 \end{bmatrix}.$$

Transform.	Computational Load			
	mult.	add.	$\sin x$	$\cos x$
SA	9	0		
AS	9	0		
TA	9	0		
AT	0	3		
R_xA	12	6	yes	yes
AR_x	16	8	yes	yes
R_yA	12	6	yes	yes
AR_y	16	8	yes	yes
R_zA	12	6	yes	yes
AR_z	16	8	yes	yes

Table I. Operations per transformation.

Then, depending on the transformation to be applied, one of the following ten cases may appear: $A' = SA$, $A' = AS$, $A' = TA$, $A' = AT$, $A' = R_xA$, $A' = AR_x$, $A' = R_yA$, $A' = AR_y$, $A' = R_zA$, $A' = AR_z$. For example, in the case of the transformation $A' = SA$, it is

$$S \cdot A = \begin{bmatrix} s_x \cdot a_{11} & s_x \cdot a_{12} & s_x \cdot a_{13} & 0 \\ s_y \cdot a_{21} & s_y \cdot a_{22} & s_y \cdot a_{23} & 0 \\ s_z \cdot a_{31} & s_z \cdot a_{32} & s_z \cdot a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & 1 \end{bmatrix},$$

which requires 9 multiplications only, instead of the 64 multiplications of the general case. A similar approach to computing the remainder of the transformations leads to an operation count reduction. Table I shows the computational load per transformation, according to the proposed approach.

Another point to emphasize is that graphics processors that appear in the literature [8, 9, 5] usually do not have a dedicated unit for the evaluation of the trigonometric functions. In most cases, these trigonometric functions are computed by the CPU of the graphics processor or the host, through an iterative algorithm such as series evaluation, in longer time of course. The proposed geometry engine is supplied with a hardware unit for the calculation of the trigonometric functions $\sin x$ and $\cos x$, thus achieving fast execution time for a complete geometrical transformation. The $\sin x$ unit used herein is a slightly modified version of the architecture proposed by [10], and it was chosen because of the high speed it offers. The function $\cos x$ is computed as $\sin\left(\frac{\pi}{2} - x\right)$.

The datapath of the proposed core (Fig. 1) consists of a unit for the evaluation of the $\sin x$ and $\cos x$ functions, a floating point multiplier, a floating point adder, and three memory units. The memory unit MEM1 is organized into 12 words of 32 bits each, while the units MEM2 and MEM3 store 8 words of 32 bits

Instruction	Binary Code
RESET	0000
RxA	0001
ARx	0010
RyA	0011
ARy	0100
RzA	0101
ARz	0110
SA	0111
AS	1000
TA	1001
AT	1010
TransformV	1011

Table II. The available instructions.

each and are used to store the intermediate results of the transformation. The address of MEM1 is four bits long while the addresses of MEM2 and MEM3 are three bits long.

A Finite State Machine (FSM) controls the execution of each transformation. The FSM initially reads a four-bit instruction that defines the desired transformation and subsequently provides the datapath with the appropriate control waveforms. The instructions and their binary representation are shown in Table II.

The RESET instruction initializes matrix A, which is stored in MEM1, to the 4×4 unit matrix. Since the memory READ and memory WRITE cycles require a precharge phase, instructions that contain such operations correspond to two states of the FSM. The TransformV instruction is the only one that produces output data, while the remaining instructions simply alter the contents of memory MEM1.

The procedure of computing the rotation transformation R_xA on the described hardware is indicatively described. A list of micro-operations that are performed at each clock cycle is offered. More than one micro-operations per state are issued to ensure maximum utilization of the available resources.

1. Commence $\sin x$ computation. Fetch a_{21} from MEM1.
2. Store $\sin x$ computation to register K and address MEM2
3. Commence $\cos x$ computation. Store $a_{21}\sin\theta$ to MEM2. Fetch a_{22} from MEM1.
4. Address MEM2.
5. Store $a_{22}\sin\theta$ to MEM2. Fetch a_{23} from MEM1.
6. Address MEM2.
7. Store $a_{23}\sin\theta$ to MEM2. Fetch a_{31} from MEM1.
8. Store $-\sin\theta$ to K and address MEM2.
9. Store $a_{31}(-\sin\theta)$ to MEM2. Fetch a_{32} from MEM1.

10-13 Similarly compute and store $a_{32}(-\sin\theta)$ and $a_{33}(-\sin\theta)$ to MEM2. Also, in step 13, store $\cos\theta$ to K.

14-25 Fetch from MEM1 the elements a_{31}, a_{32}, a_{33}, a_{21}, a_{22}, a_{23}. Compute the products $a_{31}\cos\theta$, $a_{32}\cos\theta$, $a_{33}\cos\theta$, $a_{21}\cos\theta$, $a_{22}\cos\theta$, $a_{23}\cos\theta$ and store them in MEM3.

26-37 New values for the elements a_{31}, a_{32}, a_{33}, a_{21}, a_{22}, a_{23} of A computed by adding the appropriate partial results stored in MEM2 and MEM3 and writing the final results to MEM1.

In the case of transformations R_xA, R_yA, R_zA, 37 steps are required in order to alter 6 elements of matrix A, while in the case of AR_x, AR_y, and AR_z, 49 steps are required since 8 elements of A need to be altered. The evaluation of AS and SA requires 18 steps, while the evaluation of TA and AT require 30 steps and 6 steps respectively. Finally, the TransformV instruction requires 30 steps to be completed.

Compared to a CORDIC [3], which is capable of performing algorithms that include rotations, the adopted architecture is very much faster because on a CORDIC, each operation needs time proportional to the word length to be executed, which is long for an accuracy equivalent to the one offered by the employed 32-bit floating point number representation.

An alternative arithmetic technique, the Logarithmic Number System (LNS), has been considered for implementation. An ATA-based computational procedure [10] similar to the one used for the evaluation of $\sin x$ was assumed for the implementation of LNS addition and subtraction. While in some transformations the particular arithmetic leads to a significant speed-up over the traditional floating-point performance, for addition-intensive transformations the performance is worse. Furthermore, additional hardware complexity would be imposed. Therefore, floating point arithmetic was selected. In particular, the proposed core conforms to the ANSI/IEEE 754/85 standard for single-precision floating point operations.

4. VLSI DESIGN OF THE CORE

A prototype of the proposed architecture has been designed and simulated at the gate level, using the ES2 digital standard-cell library for the 0.7 μm ecpd07 process technology. The design flow has been based on VHDL [11], because such an approach offers design re-usability, portability to various process technologies, and a degree of CAD software independence. More

specifically, the FSM and the datapath logic were synthesized and optimized starting from VHDL code, while the memories in the system have been produced with the ES2 memory generators.

The designed core comprises approximately 30K equivalent 2-input gates and occupies an area of approximately 30 mm^2. The memory units required for the $\sin x$ computation require 35 mm^2. It follows that the complete architecture can be easily integrated into a single chip.

The circuit has been simulated using the Mentor Graphics QuickSim. The units that define the delay of the design are the floating point multiplier, with a delay of 13.6 nsec, the RAM units with an access time of 10 nsec and the floating point adder that demonstrated a delay of 17 nsec. The computation of the $\sin x$ requires 40 nsec but it does not affect the overall performance severely, as it is used at most once per transformation. The system clock period is set to 52 nsec, which corresponds to a frequency of 19.2 MHz.

It should be noted that the objective of the VLSI design was to prove the feasibility of the proposed computational scheme, and not to design custom high-performance sub-modules. However, if high-performance circuits are adopted, significant performance improvement can be achieved. For better overall performance, the architecture can be modified in order to be pipelined; in this case, a significant speed-up of the throughput rate would be achieved. Furthermore, this core can be used as a Processing Element of a parallel architecture in a graphics hardware environment.

5. CONCLUSIONS

A geometry engine core, optimized for performing 3-D geometrical transformations is proposed in this paper. The architecture exploits the structure of the transformation matrices to reduce the number of floating point operations required per transformation. The inclusion of special hardware for the computation of the $\sin x$ ($\cos x$) function, facilitates rotation-related operations. The fast execution of the geometrical transformations by the proposed processor, in combination with the moderate area requirements, make the core suitable for embedding into a larger multimedia system or a DSP.

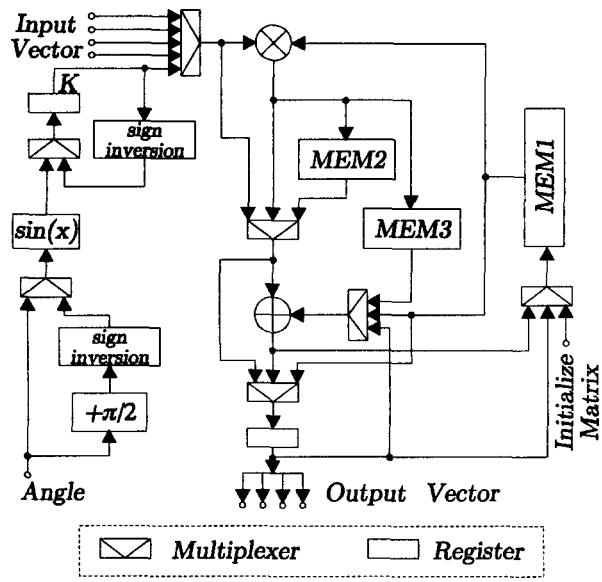

Figure 1. The proposed architecture.

REFERENCES

[1] Foley, Van Dam, Feiner, Hughes, and Phillips, *Introduction to Computer Graphics.* Addison - Wesley, 1994.

[2] *Digital Signal Processing Applications Using the ADSP-2100 Family.* Prentice Hall, 1992.

[3] D. E. Metafas and C. E. Goutis, "A floating-point advanced CORDIC processor," *Journal of VLSI Signal Processing, Kluwer Academic Publishers, Boston*, vol. 10, pp. 53–65, 1995.

[4] *82786 Graphics Coprocessor User's Manual.* intel, 1988.

[5] A. Makoto, O. Tatsushi, Y. Hideki, and S. Shigeru, "3D graphics processor chip set," *IEEE MICRO*, vol. 15, no. 6, pp. 37–45, December 1995.

[6] "Indigo 2 IMPACT breakthrough graphics," *http://www.honcad.com/honcad_home-/html/i2graphics.html*, November 1996.

[7] J. H. Clark, "The geometry engine: a VLSI geometry system for graphics," *Computer Graphics*, vol. 16, no. 3, pp. 127–133, July 1982.

[8] D. Kirk and D. Voorhies, "The rendering architecture of the DN10000VS," *Computer Graphics*, vol. 24, no. 4, pp. 299–307, August 1990.

[9] C. B. Harrell and F. Fouladi, "Graphics rendering architecture for a high performance desktop workstation," *SIGGRAPH*, pp. 93–99, 1993.

[10] W. F. Wong and E. Goto, "Fast evaluation of the elementary functions in single precision," *IEEE Transactions on Computers*, vol. 44, no. 3, pp. 453–457, March 1995.

[11] J. Bergè, A. Fonkoua, S. Maginot, and J. Rouillard, *VHDL Designer's Reference.* Kluwer Academic Publishers, 1992.