Field programmable gate array based parallel matrix multiplier for 3D affine transformations

F. Bensaali and A. Amira

Abstract: 3D graphics performance is increasing faster than any other computing application. Almost all PC systems now include 3D graphics accelerators for games, computer aided design or visualisation applications. This article investigates the suitability of field programmable gate array devices as an accelerator for implementing 3D affine transformations. Proposed solution based on processing large matrix multiplication have been implemented, for large 3D models, on the RC1000 Celoxica board based development platform using Handel-C. Outstanding results have been obtained for the acceleration of 3D transformations using fixed and floating-point arithmetic.

1 Introduction

Computer graphics algorithms are generally computationally expensive. This fact is the reason why people struggle to accelerate such algorithms using any reasonable means. The traditional sources of speedup are faster processors, parallelism or dedicated hardware. Recent advances in digital circuit technology, especially rapid development of field programmable gate arrays (FPGAs), offer alternative way for acceleration. Attempts to implement such algorithms on FPGA were the subject of several researchers. Various techniques for improving cost effectiveness of graphics applications have been described by styles and LUK. Methods for exploiting custom data formats and datapath widths, and for optimising graphics operations such as texture mapping and hidden-surface removal have been studied. Customised architectures have been implemented on the Xilinx 4000 and Virtex FPGAs in styles and Luk [1] using Handel-C, a C-like language supporting parallelism and flexible data size [2]. The work presented in Holten-Lund [3] is concerned with the implementation of a 3D graphic accelerator for an FPGA based system. Results obtained show that an FPGA based embedded system is capable of most tasks in a single chip solution, without requiring additional CPU or graphics chips. A new method of hardware texture mapping in which texture images are synthesised using FPGAs was presented in Ye and Lewis [4]. Conclusion from this work was that using FPGAs, procedural textures can be synthesised at high speed, with a small hardware cost. Matrix algorithms are very important in many types of image processing applications including 3D graphics, particularly matrix multiplication that is used in 3D affine transformations [5–7]. It is the aim of the work presented in this article to use FPGAs as an accelerator to develop and implement two matrix multipliers for 3D affine transformations using Handel-C and to evaluate the performance of the Xilinx's CoreGen [8] and Celoxica fixed-point and floating point libraries [9] for the implementation.

The target hardware for this work is Celoxica RC1000 FPGA development board fitted with a Xilinx XCV2000E-6 Virtex-E FPGA [10, 11].

IEE Proceedings online no. 20045076

doi:10.1049/ip-vis:20045076

Paper first received 30th June 2004 and in revised form 25th September 2005

F. Bensaali is with the School of Electronic, Communication and Electrical Engineering, University of Hertfordshire, College Lane, Hatfield, Herts AL10 9AB, UK

A. Amira is with Electronics and Computer Engineering, School of Engineering and Design, Brunel University, Uxbridge, Middlesex UB8 3PH, UK

E-mail: f.bensaali@herts.ac.uk

2 3D affine transformations: a review

In computer graphics the most popular method for representing an object is the polygon mesh model. In a simplest case, a polygon mesh is a structure that consists of polygons represented by a list of (x, y, z) coordinates that are called polygon vertices. Thus the information we store to describe an object is finally a list of points or vertices [12] (Fig. 1).

3D affine transformations are the transformations that involve rotation, scaling, shear and translation. A matrix can represent an affine transformation and a set of affine transformations can be combined into a single overall affine transformation. Technically, it can be said that an affine transformation is made up of any combination of linear transformations (rotation, scaling and shear) followed by translation (technically, translation is not a linear transformation) [12]. A set of vertices or 3D points belonging to an object can be transformed into another set of points by a linear transformation. Matrix notation is used in computer graphics to describe such transformations. Using matrix notation, a vertex V is transformed to V^* (* denotes the transformed vertex) under translation, scaling and rotation, which are the most commonly used transformations in computer graphics, as

$$V^* = \boldsymbol{D} + V, \quad V^* = \boldsymbol{S} \times V, \quad V^* = \boldsymbol{R} \times V \qquad (1)$$

where $\boldsymbol{D}$ is a translation vector, $\boldsymbol{S}$ and $\boldsymbol{R}$ are the scaling and rotation matrices [12, 13].

A uniform representation of all transformations in matrix notation is necessary for implementing these transformations in hardware. As it is not possible to describe the

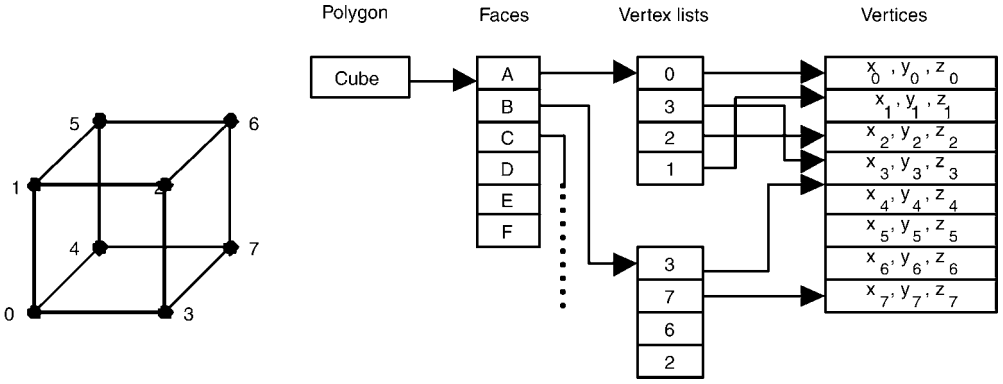


Fig. 1 *Data structure for object representation*

translation in matrix notation in Cartesian coordinates, the homogeneous coordinates have to be used. But it is very easy to transform Cartesian into homogeneous coordinates and vice versa. In a homogeneous system a vertex $V(x, y, z)$ is presented as $V(X, Y, Z, w)$ for any scale factor $w = 1$. The 3D Cartesian coordinate representation is then

$$x = \frac{X}{w}, \quad y = \frac{Y}{w}, \quad z = \frac{Z}{w} \tag{2}$$

In computer graphics w is always taken to be one and the matrix representation of a point is $(x\ y\ z\ 1)^{\mathrm{T}}$. Translation can now be treated as a matrix multiplication operation, like the other two transformations, and becomes

$$\begin{pmatrix} x^* \\ y^* \\ z^* \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = T \times \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{3}$$

Therefore in homogeneous coordinates it is possible to describe any transformation in a matrix notation

$$\begin{pmatrix} x^* \\ y^* \\ z^* \\ 1 \end{pmatrix} = \left(\begin{array}{ccc|c} A & D & G & J \\ B & E & H & K \\ C & F & I & L \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \times \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{4}$$

This universal matrix for transformations can be divided into four function blocks

$$\left(\begin{array}{c|c} \text{scaling and rotation} & \text{translation} \\ \hline \text{part of the homogeneous representation} & 1 \end{array}\right) \tag{5}$$

The matrix representations for the two others most commonly used transformations are as follows.

2.1 Scaling

$$V^* = \boldsymbol{S} \times V \tag{6}$$

$$\boldsymbol{S} = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{7}$$

Here S_x, S_y, and S_z are the scaling factors. For a uniform scaling: $S_x = S_y = S_z$.

2.2 Rotation

To rotate an object in a 3D space, an axis of rotation needs to be specified. This can have any spatial orientation in a 3D space, but it is easier to consider rotations that are parallel to one of the coordinate axes. The transformation matrices for rotation about the X, Y and Z-axes, respectively are

$$\boldsymbol{R}_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\boldsymbol{R}_y = \begin{pmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{8}$$

$$\boldsymbol{R}_z = \begin{pmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

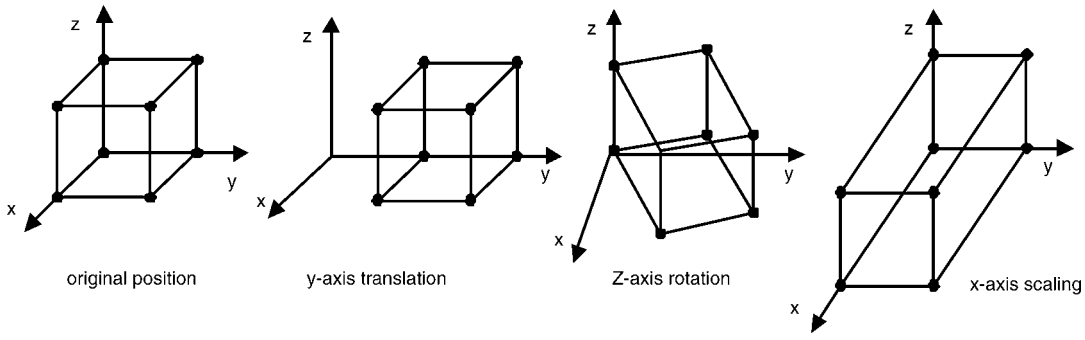
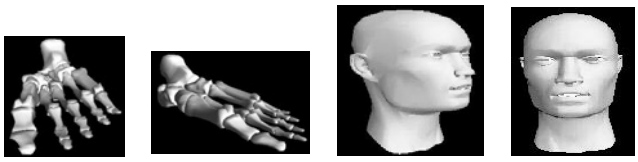


Fig. 2 *3D transformation examples*



a b

Fig. 3 *Examples of 3D objects*
a Foot skeleton contains 2154 vertices
b Face contains 5597 vertices

It is worth noting that a sequence of transformations can be represented by one matrix $\boldsymbol{T} = T_1 \times T_2 \times \cdots \times T_N$
Fig. 2 shows different examples of a cube containing eight vertices when applying different transformations.

3 Modeling 3D affine transformations using large matrix multiplication

Consider an object represented with N vertices. The new position (***NP***) of the object when applying a transformation can be calculated as follows

$$\boldsymbol{NP} = \boldsymbol{T} \times \boldsymbol{OP} \tag{9}$$

where $\boldsymbol{T}$ is the matrix transform, ***OP*** a (4, N) matrix contains the old vertices position, ***NP*** a (4, N) matrix contains the new vertices position.

$$\begin{pmatrix} x_0^* & x_1^* & \cdots & x_{N-1}^* \\ y_0^* & y_1^* & \cdots & y_{N-1}^* \\ z_0^* & z_1^* & \cdots & z_{N-1}^* \\ 1 & 1 & \cdots & 1 \end{pmatrix} = \begin{pmatrix} A & D & G & J \\ B & E & H & K \\ C & F & I & L \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 & x_1 & \cdots & x_{N-1} \\ y_0 & y_1 & \cdots & y_{N-1} \\ z_0 & z_1 & \cdots & z_{N-1} \\ 1 & 1 & \cdots & 1 \end{pmatrix} \tag{10}$$

Fig. 3 shows two 3D objects, a foot skeleton ($N = 2154$) and a face ($N = 5597$).

4 FPGA implementation

4.1 Implementation approach

Handel-C is a high-level language that is at the heart of a hardware compilation system known as Celoxica development kit (DK) [9], which is designed to compile programs written in a C-like high-level language into synchronous hardware. One of the advantages of using hardware is the ability to exploit parallelism directly. Because standard C is a sequential language, Handel-C has additional constructs to support the parallelisation of code and to allow fine control over what hardware is generated.

DK produces a Netlist file, which is used during the place and route stage to generate the image or bitstream file [9] (Fig. 4).

The RC1000 co-processor board used is a standard PCI bus card equipped with a large FPGA chip. It has 8 MB of SRAM directly connected to the FPGA in four 32-bit wide memory banks. All are accessible by the FPGA and any device on the PCI bus. Different methods of data transfer from the host PC or the environment to the FPGA are available as follows:

- Bulk transfers of data between FPGA and PCI bus is performed through the memory banks 0–3.
- Streams of bytes are most conveniently communicated through the unidirectional 8-bit control and status-ports (Fig. 4).

The RC1000 board is supported with a macro-library that simplifies the process of initialising and talking to the hardware. This library comprises a set of driver functions with the following functionality:

- Initialisation and selection of a board
- Handling of FPGA configuration files
- Data transfer between PC and the RC1000 board
- Function to help with error checking and debugging

These library functions can be included in a C or C++ program that runs on the host PC and performs data transfer via the PCI bus [9].

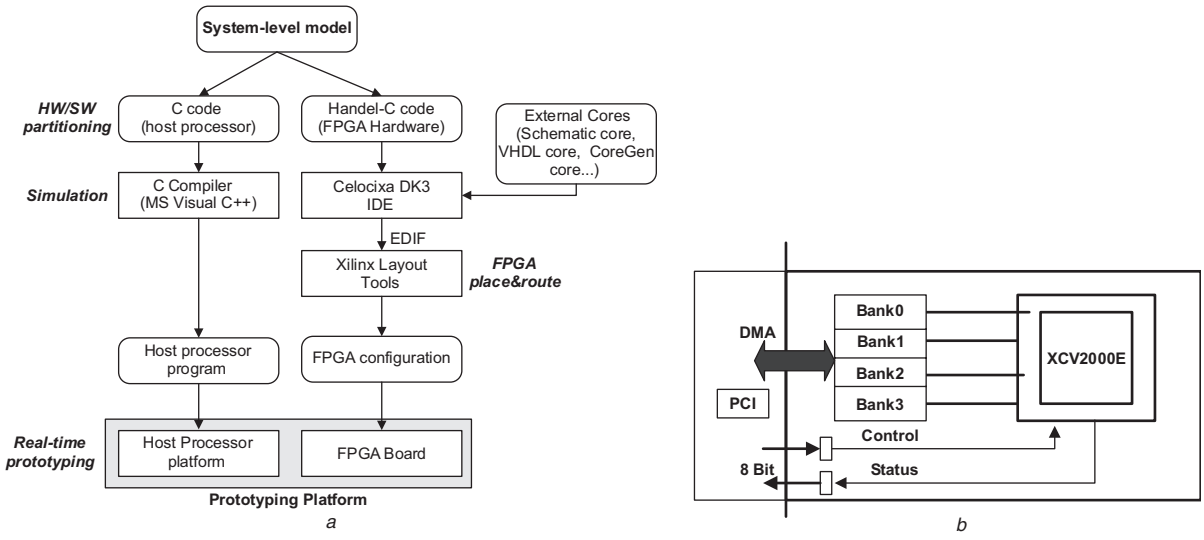


Fig. 4 *Hardware–software tools used for the implementation*
a Handel-C design flow
b Schematic view of the FPGA/banks part in the RC1000 board

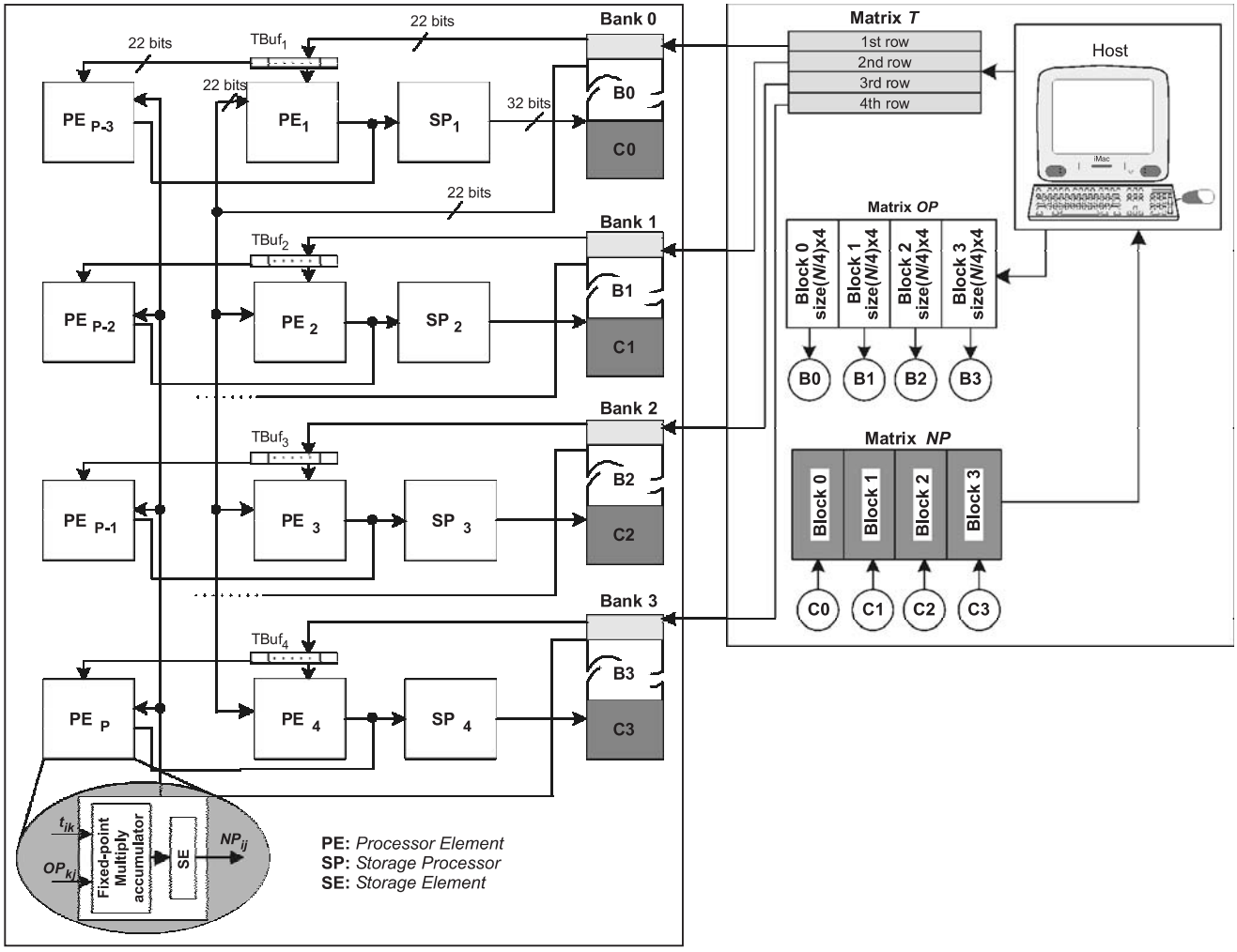


Fig. 5 *Proposed parallel fixed-point matrix multiplier for 3D affine transformations*

4.2 Proposed parallel matrix multipliers for 3D affine transformations

The vertices' coordinates require real-number representation. Therefore floating-point or fixed-point representations can be used. Celoxica provides two libraries (pipelined floating-point and fixed-point), which allow different widths to be specified, allowing designers to use the minimum numbers of bits to represent data and consequently generate smaller hardware.

In this section two parallel matrix multipliers are described. The first proposed structure is based on fixed-point representation, whereas the second one is based on the floating-point representation.

The two multipliers have been implemented, using Handel-C and compiled using DK version 2 (DK2), on the RC1000 board. In order to perform this multiplication the four external memories have been exploited for data storage to allow the multiplication to be performed by the FPGA.

4.2.1 Proposed parallel fixed-point matrix multiplier: Fig. 5 shows the proposed parallel fixed-point matrix multiplier, which can be used to perform the matrix multiplication described in Section 3.

The proposed architecture consists of p identical PEs, which should be a multiple of four. Each PE comprises a fixed-point multiply accumulator (MAC) and a register for final result storage. The MAC has been implemented using two approaches:

1. MAC based Celoxica fixed-point library: This is a device independent hardware library that allows the width of the fractional and integer part of the number to be defined and provides macros to execute arithmetic operations [9].
2. MAC based Xilinx's CoreGen: Xilinx's CoreGen utility contains many designs that can often save time for a programmer and it is possible to integrate Xilinx CoreGen blocks with a program in Handel-C using the interface declaration [8]. Two components have been used, a parallel signed integer multiplier and a parallel signed integer adder, which are suitable for the Xilinx XCV2000E-6 Virtex-E FPGA (Fig. 6).

In both cases, the vertices' coordinates are represented with 22 bits (14 bits for fractional part, 7 bits for integer part, one sign bit). The input transform matrix $\boldsymbol{T}$ is partitioned into four rowwise blocks, which gives one row per block. Each block is stored in one of the four available banks. The matrix $\boldsymbol{OP}$ is partitioned into four columnwise

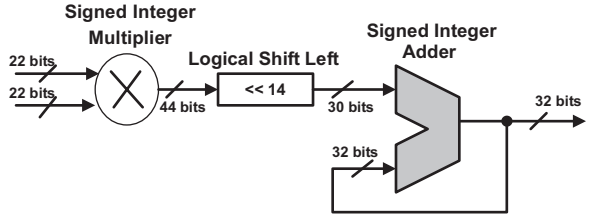


Fig. 6 *Fixed-point MAC*

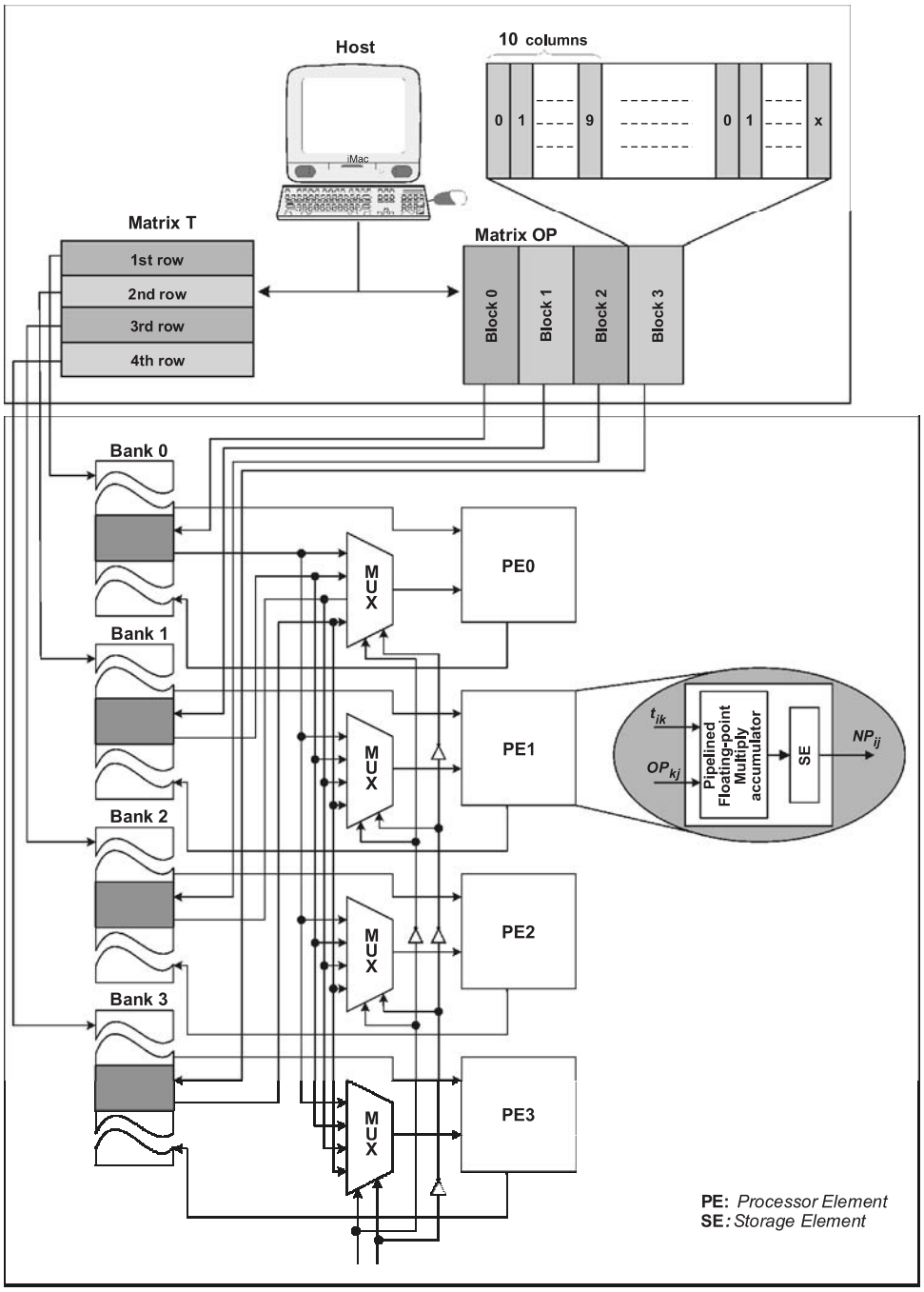


Fig. 7 *Proposed parallel floating-point matrix multiplier for 3D affine transformations*

blocks, likewise in matrix ***T***, each block is stored in one of the banks. Because of the problem of accessing different elements stored in the same SRAM simultaneously, four buffers (TBuf1, TBuf2, TBuf3, TBuf4) for storing the four rows of ***T*** have been used to avoid a memory conflict. Data are transferred from the banks to the buffers. Columns of the matrix ***OP*** are transferred from SRAMs to the PEs in parallel. Each PE computes one element of the output matrix ***NP*** at once. The four storage processors (SPs), which have access to the PEs registers, are used to transfer the final results to the banks and operate as an interface between the p PEs and the four memory banks. Each group of four PEs is working as a matrix-vector multiplier. Therefore four PEs are used to compute the ***NP*** of a vertex. The entire computation of the matrix ***NP*** can be carried out in $[2 \times (4 \times N)/p + BI + N/NB]$ clock cycles (the number of clock cycles has been computed from the Handel-C code), where 2 is the number of clock cycles needed by the MAC for one accumulation, N the number of object vertices, p the number of PEs used, $BI = 4$ the number of clock cycles needed for buffers initialisation, $NB = 4$ the number of memory banks available and N/NB the number of clock cycles needed for final result storage.

It is worth noting that the number of vertices N is always rounded to a multiple of four. Therefore the last partition of

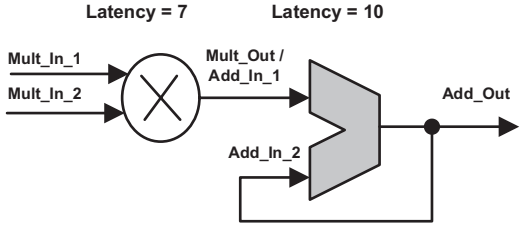


Fig. 8 *Floating-point MAC*

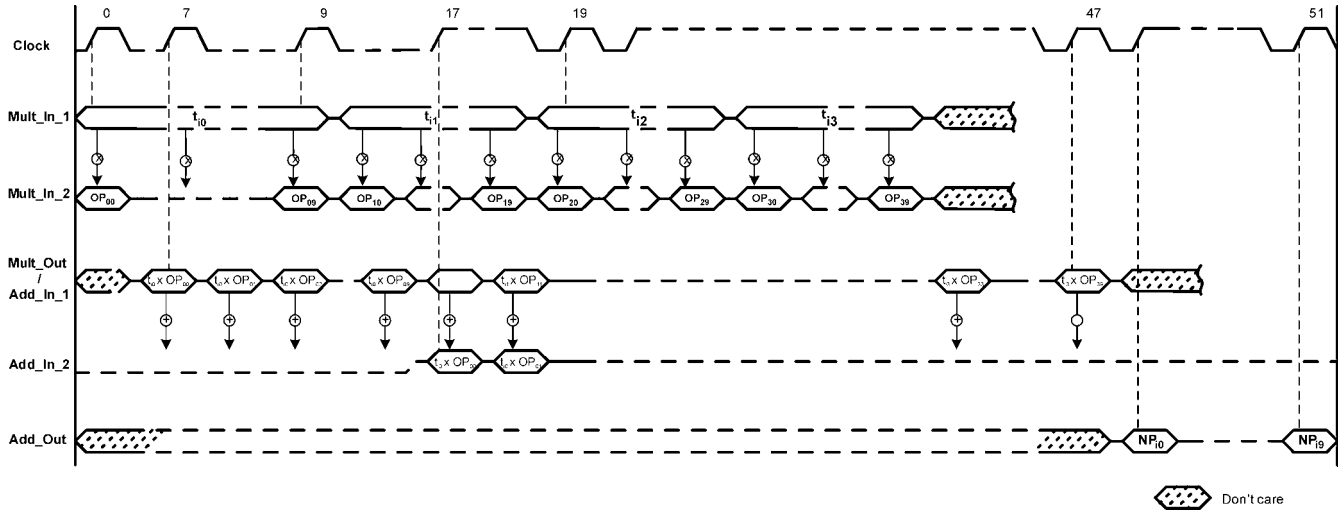


Fig. 9 *Timing diagram when performing a multiplication of one row of the transform matrix **T** with one sub-block of the matrix **OP***

the matrix ***OP*** should be padded with, at most, three columns of zeros (e.g. for $N = 2154$ two columns of zeros should be supplemented to the matrix ***OP***, in this case each partition has a size of 539×4). As the number of clock cycles needed in order to perform a transformation is controlled by the slowest processor element, adding columns of zeros will not affect it.

4.2.2 Proposed parallel floating-point matrix multiplier

Because of the large dynamic range and very high precision offered by floating-point arithmetic, a parallel floating-point matrix multiplier has been implemented as shown in Fig. 7.

Each PE comprises a pipelined floating-point MAC (Fig. 8) and a register for final result storage. The MAC has been implemented using the pipelined floating-point library from Celoxica. The vertices' coordinates are represented using the IEEE standard floating-point format for single-precision real numbers (32 bits: one sign bit, eight bits of the exponent and 23 bits for the fraction) [14]. The adders and multipliers used are pipelined and have a latency of ten and seven, respectively. The two input matrices are partitioned using the same partitioning strategy used for the previous fixed-point architecture. In addition, because of the adder latency, each block of the matrix ***OP*** is partitioned into columnwise sub-blocks. Each sub-block contains ten columns and the last one is padded with columns of zeros if the size is not ten.

Fig. 9 illustrates the timing diagram when performing a multiplication of one row of the transform matrix ***T*** with one sub-block of the matrix ***OP*** as shown in (11)

$$\begin{pmatrix} NP_{i0} & NP_{i1} & \cdots & NP_{i9} \end{pmatrix} = \begin{pmatrix} t_{i0} & t_{i1} & t_{i2} & t_{i3} \end{pmatrix} \begin{pmatrix} OP_{00} & OP_{01} & \cdots & OP_{09} \\ OP_{10} & OP_{11} & \cdots & OP_{19} \\ OP_{20} & OP_{12} & \cdots & OP_{29} \\ OP_{30} & OP_{13} & \cdots & OP_{39} \end{pmatrix} \quad (11)$$

The number of clock cycles required for the entire computation of the matrix NP is

$$C = \left(\frac{N}{p \times \mathrm{AL}} \right) \times ((\mathrm{AL} + \mathrm{ML} - 1) + (4 \times 10)) \quad (12)$$

where p is the number of PEs, AL the adder latency ($\mathrm{AL} = 10$), ML the multiplier latency ($\mathrm{ML} = 7$) and (4×10) the size of the ***OP*** sub-matrices.

4.3 Proposed environment for 3D affine transformations on FPGA

Fig. 10 shows a general view of the entire proposed system.

The environment consists of a host application (GUI), a 3D object database, the open graphics library (OpenGL)

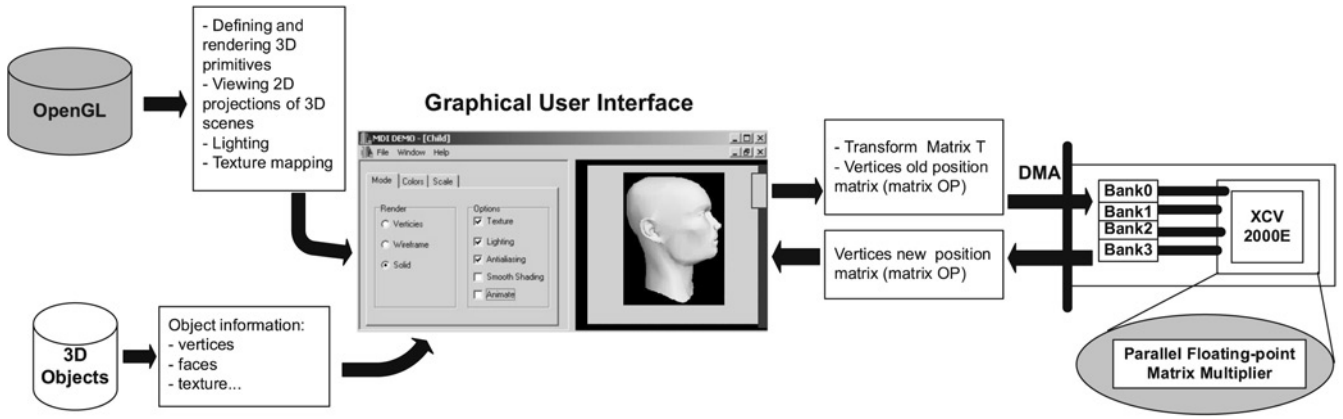


Fig. 10 *Proposed system for 3D affine transformations on FPGA*

[15] and the single FPGA-chip coprocessor based on the RC1000 development board.

- 3D object database contains the 3D model files (.OBJ, .3DS, .ASE).
- OpenGL is the specification of a powerful set of more than 350 graphics routines for 2D and 3D graphics processing. OpenGL includes facilities for
  - Defining and rendering 3D primitives such as points, lines, polygons, spheres and cones
  - Viewing 2D projections of 3D scenes
  - Manipulating coordinate transformations
  - Lighting: light sources and material properties
  - Texture mapping
- Coprocessor performs the 3D affine transformations.
- Host application (GUI) implemented using Borland C++, gives the user the ability to select a 3D model from the 3D object database and display it on the available 3D viewer. The user can apply different algorithms on the object, such as texture, lighting and antialiasage, which involve calls to OpenGL functions. Since C++ does not support fixed-point formats, a floating-point to fixed-point converter has been implemented. The vertices coordinates are converted from floating-point to fixed-point before performing the DMA transfer. The inverse operation is applied to the result in order to reconstruct the transformed 3D model.

5 Results and analysis

The two architectures have been implemented and tested on the RC1000 board. Table 1 illustrates the performance obtained for the proposed architectures when using the three different design approaches for the MAC implementation.

Table 1: Area/speed implementation report for the proposed parallel matrix multipliers – target FPGA XCV2000E (bg560-6)

MAC used	Number of PEs	Area (%)	Speed (MHz)
Celoxica fixed-point library	4	3264	22
	8	6720	20
	16	14 400	17
Xilinx's CoreGen	4	1920	35
	8	3840	30
	16	8064	24
Celoxica pipelined floating-point library	4	19 010	50

Recent innovations in FPGA architecture have changed the design tradeoff space by providing new fixed embedded cores that can be employed within a complex design. These include high density multiplier blocks and shift registers. The floating-point library and the Xilinx's CoreGen support the Virtex-II and the Virtex-II pro FPGA chip, which includes up to 168 (18 bits) multiplier blocks for the Virtex-II and 556 for the Virtex-II pro. This allows multiplication to be done inside the FPGA without the use of large numbers of look up tables (LUTs) and also at a low power cost [16, 17].

Table 2 illustrates the performance obtained when the 18 bit multipliers employed for the implementation of our architectures.

From Tables 1 and 2 it can be seen that the use of the embedded multiplier cores results in significant improvement in the clock speed compared to similar implementation based solely on conventional LUT/FF logic.

The implementation using the MAC based Xilinx's CoreGen shows better performances when comparing with the one based Celoxica's fixed-point library due to the suitability of the cores used for the FPGA chip selected.

In addition, Virtex-II pro device gives higher maximum running frequency compare to Virtex-II due to the different process technology used.

Floating-point representation provides large dynamic range and very high precision, but has large resource requirements compare to the fixed-point one. If the range of real-numbers values that must be represented is small or can be scaled in order to make it smaller, fixed-point arithmetic is one way of providing cheap fast non-integer support. Fixed-point arithmetic is appropriate for our application because the range of the values is small.

There exist expensive 3D cards, which support the manipulation of coordinate transformations. Our PC (Pentium 4 CPU 2.00 GHz) is equipped by an ATI RADEON FSC 32 MB graphics card, which belongs to this category of cards. This card delivers immerse, realistic colour and 3D graphics at a fast frame rate. The RADEON Charisma Engine, which takes care of the geometry processing, and the pixel tapestry architecture, which is the rendering engine, support full transformation, clipping and lighting for improvement in 3D details. With full support for DirectX and OpenGL, it accelerates all today's top 3D games.

Table 3 shows the performances obtained in terms of maximum running frequency and computation time for

Table 2: Area/speed implementation report for the proposed parallel matrix multipliers – target FPGAs XC2V8000 (ff1152-5) and XC2VP125 (ff1704-7)

MAC used	Number of PEs	Area (%)		18 × 18 Mults used		Speed (MHz)	
		V-II	V-II pro	V-II	V-II pro	V-II	V-II pro
Celoxica fixed-point library	4	2795	2780	—	—	36	52
	8	6057	5997	—	—	30	45
	16	14443	14360	—	—	26	38
Xilinx's CoreGen	4	931	925	16	16	83	114
	8	2795	2624	32	32	78	110
	16	5591	5494	56	56	71	102
Celoxica pipelined floating-point library	4	24693	24493	64	64	108	119

Table 3: Computation time comparison of the proposed structures with the RADEON FSC 32 MB graphics card

MAC used	Number of PEs	Maximum frequency (MHz)		Computation time (μs)	
		V-II	V-II Pro	V-II	V-II pro
FPGA					
Celoxica fixed-point library	4	36	52	134.74	93.28
	8	30	45	89.88	59.92
	16	26	38	36.33	24.85
Xilinx's CoreGen	4	83	114	58.43	42.55
	8	78	110	34.57	24.51
	16	71	102	13.30	9.26
Celoxica pipelined floating-point library	4	108	119	27.92	25.34
RADEON FSC 32 MB graphics card	—	—	—	—	14.81

the RADEON graphics card and our FPGA implementation with different number of PEs when performing a rotation on the foot skeleton object, which contains 2154 vertices. The same approach can be taken for the other transformations.

The pseudo-code used to retrieve the computation time is shown in algorithm 1.

It can be seen from Table 3 that the fixed-point matrix multiplier based CoreGen MAC gives better result in terms of the computation time when using 16 PEs compared to the graphic card. The performance of the fixed-point matrix multiplier dedicated for 3D affine transformations demonstrates that the FPGA can be used as an effective acceleration solution.

Algorithm 1: Pseudo-code used for the computation time
QueryPerformanceFrequency(&Frequency); {retrieve the frequency of the high-resolution performance counter}

QueryPerformanceCounter(&t1); {get the value of the performance counter before the performing a 3D affine transformation}

3D affine transformations {OpenGL functions or FPGA program call}

QueryPerformanceCounter(&t2); {get the value of the performance counter after the performing the transformation}

t.QuadPart = (t2.QuadPart-t1.QuadPart){Calculate the time interval between measurements}

6 Conclusion

The hardware-accelerated architectures for computer graphics and image processing are one of the promising approaches in computer graphics. The common rule is that functions integrated in silicon are much faster than functions integrated in software. There exist expensive cards, in which all OpenGL algorithms are completely integrated in silicon and therefore those cards are very fast. As FPGAs have grown in capacity, improved in performance, and decreased in cost, they have become a viable solution for performing computationally intensive tasks, with the ability to tackle applications for custom chips and programmable DSP devices. In this article the use of FPGAs as an accelerator for OpenGL 3D affine transformations using a parallel large matrix multiplication approach has been presented. Different data types, including fixed and floating-point arithmetic have been investigated and results obtained show better performances in comparison with existing systems.

7 References

1 Styles, H., and Luk, W.: 'Customising graphics applications: techniques and programming interface'. Proc. IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM), Napa, CA, April 2000
2 Celocixa Ltd. 'Handel-C language reference manual', 2003, Manual: www.celoxica.com
3 Holten-Lund, H.: 'Embedded 3D graphics core for FPGA-based system-on-chip application'. The FPGAword Conference, Stockholm, Sweden, September 2005
4 Ye, A.G., and Lewis, D.M.: 'Procedural texture mapping on FPGAs'. ACM/SIGDA Int. Symp. on Field Programmable Gate Arrays, Monterey, CA, February 1999, pp. 112–120
5 Amira, A.: 'A custom coprocessor for matrix algorithm'. PhD Dissertation, Queen's University of Belfast, UK, 2001
6 Bensaali, F., Amira, A., Uzun, I.S., and Ahmedsaid, A.: 'An FPGA implementation of 3D affine transformations'. 10th IEEE Int. Conf. on Electronics, Circuits and Systems (ICECS'03), Sharjah, UAE, December 2003
7 Bensaali, F., Amira, A., Uzun, I.S., and Ahmedsaid, A.: 'Efficient implementation of large parallel matrix product for DOTs'. Int. Conf. on Computer, Communication and Control Technologies (CCCT'03), FL, USA, July 2003
8 Application note, Xilinx CoreGen and Handel-C, AN 58 (v1.0), 2001
9 URL: www.celoxica.com
10 'RC1000 reconfigurable hardware development platform' (Celocixa Ltd., 2001), Datasheet, www.celoxica.com
11 URL: www.xilinx.com
12 Watt, A.: '3D computer graphics' (Addison-Wesley, 2000)
13 Ferguson, R.S.: 'Practical algorithms for 3D computer graphics' (A K Peters, 2001)
14 'IEEE Standard for binary floating-point arithmetic', ANSI/IEEE Std 754-1985, NY, USA, 1985
15 URL: www.opengl.org
16 Datasheet, 'Virtex-II platform FPGAs: complete data sheet', DS031 (v3.3), (Xilinx Inc., 2004)
17 Datasheet, 'Virtex-II pro and Virtex-II pro X platform FPGAs: complete data sheet', DS083 (v4.1), (Xilinx Inc., 2004)