LOW COST AND LATENCY EMBEDDED 3D GRAPHICS RECIPROCATION

Dan Crisu, Stamatis Vassiliadis, Sorin Cotofana

Computer Engineering Laboratory, EEMCS
Delft University of Technology, Delft, The Netherlands
E-mail: {dan, sorin, stamatis}@ce.et.tudelft.nl

Petri Liuha

NOKIA Research Center
Tampere, Finland
E-mail: petri.liuha@nokia.com

ABSTRACT

The paper presents low cost and latency reciprocation for fixed-point datapath of embedded 3D graphics accelerators. The algorithm exploits the limitations of the human visual system that allows a reasonable amount of error to be introduced in the computation process without inducing noticeable image artifacts. In the example given in the paper, excerpted from the antialiasing datapath of an embedded QVGA graphics hardware accelerator, for a 14-bit operand, the reciprocal implementation requires an inexpensive operand prescaler, one 1k lookup table with 10-bit entries, and a 5-bit adder, for a maximum relative error of the result of only 1.5% over the entire range of the operand. Hardware synthesis in a typical 0.18μm process technology has indicated that the hardware implementation requires only 1600 standard cells to achieve a latency of 2.5ns.

1. INTRODUCTION

For embedded graphics hardware it can be shown [1, 2] that reducing the internal precision for certain operations and applying a number of computational approximations on commonly encountered and difficult arithmetic operators, like division, can be beneficial for area and power consumption reduction at the same performance level without compromising the image quality.

This paper presents a low-cost, low-latency reciprocation hardware algorithm suitable to be implemented in the fixed-point datapath of low-power, low-cost embedded 3D graphics accelerators. Assuming a 14-bit operand and the antialiasing datapath of an embedded QVGA graphics hardware accelerator, it is shown that:

- the reciprocal implementation requires for a maximum relative error of the result of only 1.5% over the entire range of the operand an inexpensive operand prescaler, one 1k lookup table with 10-bit entries and a 5-bit adder.

Furthermore it is shown that:

- when it is synthesized in a typical 0.18μm process technology, the hardware implementation requires only 1600 standard cells to achieve a latency of 2.5ns.

The rest of the paper is organized as follows. The reciprocal hardware algorithm we propose for embedded 3D graphics is presented in Section 2. The impact on the quality of the rendered images using the proposed algorithm and hardware synthesis results are presented in Section 3. Finally, Section 4 draws the conclusions.

2. BACKGROUND AND RECIPROCAL ALGORITHM

There are various algorithms for hardware rasterization implementation. One common method of triangle rasterization is based on the algebraic representation of triangle's edges with edge functions [3]. The sign of the edge function determines the half-plane of the current rasterization position (x_M, y_M) in respect to the edge vector. The rasterization with edge functions is presented in a self explanatory manner in Figure 1. Additionally, if the edge function is properly normalized, its evaluation yields also the distance from the edge vector to the pixel position (x_M, y_M) that can be used in antialiasing. Such a scheme was presented in the Exact Area Sampling Algorithm (EASA) [4] based on [3]. The normalized edge function formulation of EASA is as follows:

$$\begin{aligned} d_{L_1}(M) &= \tfrac{E(x_M,y_M)}{|\Delta x|+|\Delta y|} \\ &= (x_M - x_A)\cdot \tfrac{\Delta y}{|\Delta x|+|\Delta y|} - (y_M - y_A)\cdot \tfrac{\Delta x}{|\Delta x|+|\Delta y|} \\ &= (x_M - x_A)\cdot de_x(\alpha) - (y_M - y_A)\cdot de_y(\alpha) \end{aligned} \quad (1)$$

From the formulation it can be seen that a reciprocal is required by $de_x(\alpha)$ and $de_y(\alpha)$ parameter computation.

To rasterize a triangle, in the *triangle setup stage* the exact values for the edge functions, z, colors, and texture coordinates are computed for a conveniently chosen pixel (x, y) on the screen as well as their interpolation steps (gradients) along the x and y axes. In the triangle setup stage reciprocal computations are required. The expressions for the gradient setup used in the depth (z value) linear interpolation during the rasterization of the triangle with the vertices A, B, C

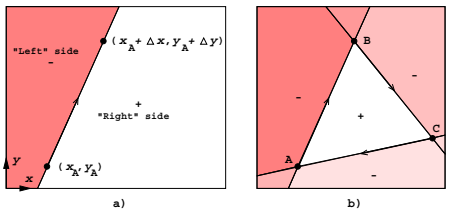

Fig. 1. Triangle representation using edge functions: a) The edge is defined by a vector starting at the point A with the slope $\Delta y/\Delta x$, b) The interior of the triangle corresponds to rasterization positions where the three edge functions have the same sign.

are:

$$\begin{array}{rcl} \frac{\delta z}{\delta x} & = & \frac{(\Delta y_{CA} \cdot \Delta z_{AB} - \Delta z_{CA} \cdot \Delta y_{AB})}{E_{AB}(x_C, y_C)} \\ \frac{\delta z}{\delta y} & = & \frac{(\Delta z_{CA} \cdot \Delta x_{AB} - \Delta x_{CA} \cdot \Delta z_{AB})}{E_{AB}(x_C, y_C)} \end{array} \quad (2)$$

where $E_{AB}(x_C, y_C)$ represents the expression $E(x_C, y_C)$ of Equation (1) for the oriented edge AB. The reciprocal of the $E_{AB}(x_C, y_C)$ expression is also required by the texture coordinates setup [5].

In the remaining, we will focus on reciprocal computation thus transforming the division in a multiplication of the numerator with the reciprocal of the denominator [6]. Instead of providing datapath bit-exact arithmetic at a high expense regarding cost, latency, and power-consumption, we reduce the cost of the reciprocal by exploiting the limitations of the human visual system which allows a reasonable amount of error to be introduced in the computation process without introducing noticeable artifacts. The only question that remains to be answered is how large the error has to be in order to be tolerable. Our experiments on a QVGA display have suggested that a relative error of maximum 5% does not introduce visible artifacts in the generated image [5]. Therefore, the goal is to perform the reciprocal computation with less than 5% relative error. The relative error $\epsilon_{rel}(1/X)$ of the reciprocal computation is defined as:

$$\epsilon_{rel}\left(\frac{1}{X}\right) = \frac{\left(\frac{1}{X}\right)_{approx} - \frac{1}{X}}{\frac{1}{X}} \quad (3)$$

where $(1/X)_{approx}$ represents the reciprocal computed in hardware and $(1/X)$ represents the true value of the reciprocal of a given value X.

To illustrate the different design trade-offs in the design of the reciprocal we will work on the example given by Equation (1). The screen coordinates x and y for a QVGA display (with a resolution of 320×240) will be represented as unsigned fixed-point numbers in the format 9.4 (meaning 9 integer bits and 4 fractional bits). The fractional part of the coordinates is necessary to eliminate drop-outs and overlaps in the rasterized image [7]. This means that the quantity $|\Delta x| + |\Delta y|$ will be represented as an unsigned fixed-point

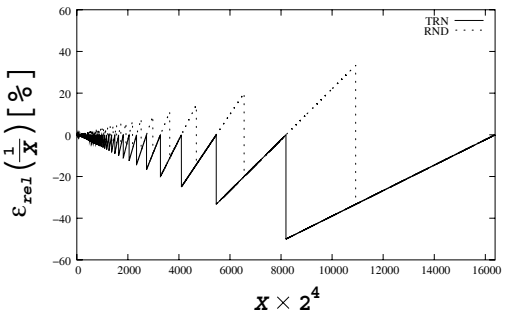

Fig. 2. Preliminary reciprocal relative error.

Index(X)	Reciprocal $\left(\frac{1}{X}\right)$		Entry used
	Mantissa	Exponent	
0000000000.0000	(1).xxxxxx	x	No
0000000000.0001	(1).000000	+4	Yes
0000000000.0010	(1).000000	+3	Yes
0000000000.0011	(1).010101	+2	Yes
...	...	...	...
1111111111.1110	(1).000000	−10	Yes
1111111111.1111	(1).000000	−10	Yes

Table 1. Preliminary lookup table content.

number in format 10.4. Moreover, very small triangles will be culled in the software driver if $|\Delta x| + |\Delta y| < 2^{-4}$ imposing that the reciprocal should be less than or equal to 2^4. Also, $|\Delta x| + |\Delta y| \leq (2^{10} - 2^{-4})$ establishing the minimum value of the reciprocal to approximative 2^{-10}.

To summarize, the objective is to compute the reciprocal of an unsigned non-zero fixed-point number X represented in a 10.4 format with a maximum relative error of only 5%.

If we attempt to directly implement the reciprocal with a lookup table of 16k (X is represented with 14 bits) fixed-point 15-bit entries (5 bits for the integer part and 10 bits for the fractional part) using truncation or rounding produces errors (see Figure 2) larger than 5%. Consequently the requirement is not met. By introducing a floating-point format with a 1.6 mantissa format (1 bit for the integer part and 6 bits for the fractional part) and a 5-bit exponent in two's complement notation the relative error can be reduced to 1.5% (see Table 1 for the lookup table content and Figure 3

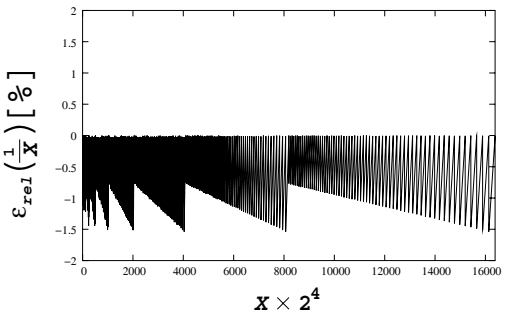

Fig. 3. Preliminary reciprocal relative error.

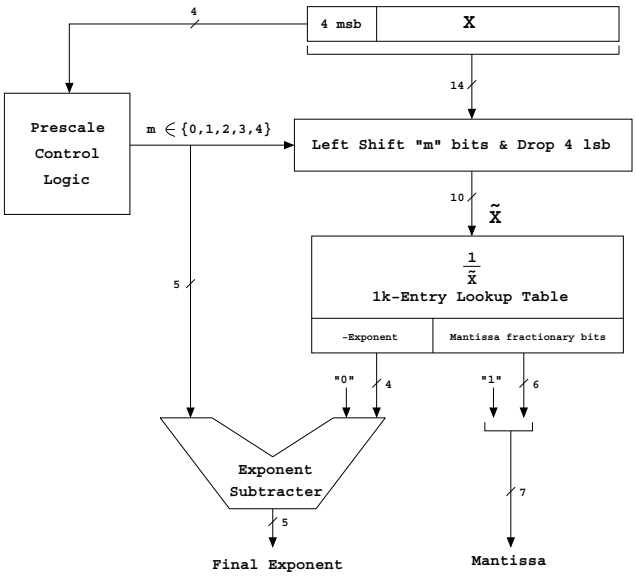

Fig. 4. Block diagram of the reciprocal hardware.

$X[13:10]$	Required Left Shift Positions (m)
0000	4
0001	3
001x	2
01xx	1
1xxx	0

Table 2. Denominator (X) prescaling control logic.

for the relative error of the reciprocal). The width for mantissa was chosen to prevent loss of precision in subsequent computations in the datapath. While the computational goal regarding precision is met, the lookup table with 16k 11-bit entries is excessive in terms of area. We observe that due to its nonlinearity, the reciprocal computation can be implemented using only a lookup table of 1k entries by computing the reciprocal of small numbers with the accuracy given by the lookup table, and for large numbers introducing an insignificant additional error by ignoring some or all of their 4 bits of the fractional part. The scheme is depicted

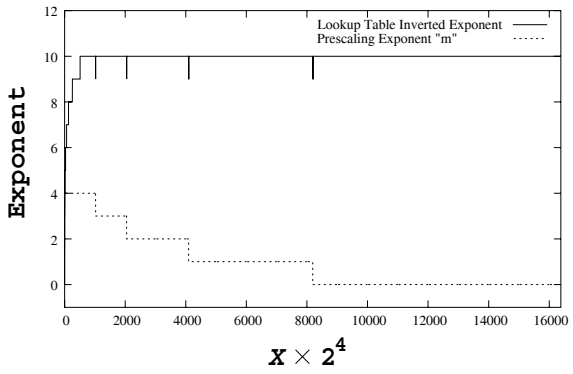

Fig. 5. Exponents behavior over the denominator range.

Index$(\tilde{X})$	Reciprocal $\left(\frac{1}{\tilde{X}}\right)$ Mantissa	−Exponent	Entry used
0000000000	(1).xxxxxx	x	No
0000000001	(1).000000	0	Yes
0000000010	(1).000000	+1	Yes
0000000011	(1).010101	+2	Yes
...	...	...	...
1111111110	(1).000000	+10	Yes
1111111111	(1).000000	+10	Yes

Table 3. Final lookup table content.

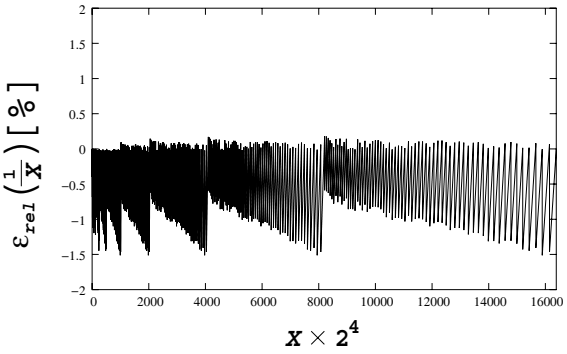

Fig. 6. Final reciprocal relative error.

in Figure 4. The lookup table for reciprocal is presented in Table 3 and stores values only for the reciprocal of positive integers that can be represented on 10 bits. Given that all the exponents are now non-positive numbers an extra bit per entry can be saved by inverting and storing them as unsigned numbers. To avoid losing precision, the original 14-bit denominator has to be prescaled by left shifts up to 4 bit positions whenever the most 4 significant bits contain leading zeros. The rule for computing the required number of shifts from the 4 most significant bits of the denominator is presented in Table 2. After shifting or even when no shifting is performed, the least four significant bits of the denominator are thrown away and the the surviving 10 bits in the result are used as an index in the lookup table to fetch the reciprocal. This is the role of the prescaling control logic and shifter depicted in Figure 4. The number of shifts performed are recorded and sent to a small two's complement subtracter, also sketched in Figure 4, which compensates the inverted exponent fetched from the lookup table forming the true exponent of the reciprocal. By taking into account the behavior of the exponents presented in Figure 5, a 5-bit width subtracter is required. The relative error of the reciprocal computation using this method over the entire range of possible 14-bit values of the denominator is depicted in Figure 6. Comparing Figures 6 and 3, it can be seen that the final relative error (Figure 6) is almost the same with the preliminary relative error (Figure 3) with the notable difference that the final version uses a 1k 10-bit entry lookup table instead of a 16k 11-bit entry lookup table. Thus, the

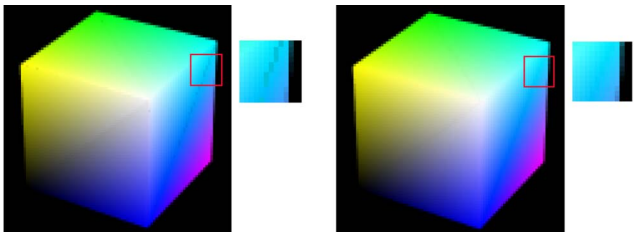

Fig. 7. A blowup on the image generated by "aapoly" OpenGL application — Left image: reciprocal computation using direct lookup implementation; Right image: reciprocal computation according to our method.

last method offers a low-cost hardware algorithm for reciprocation suitable to embedded 3D graphics.

3. RESULTS

To assess the effectiveness of the proposed method over direct lookup fixed-point implementation we perform the following. The hardware model of the two versions was employed in the datapath of our OpenGL 1.2 compliant 3D graphics hardware accelerator SystemC model for an ARM based SOC platform. Referring to the internal organization, the graphics accelerator adopts a tile-based rasterization approach. The tile size chosen for this particular implementation was set at 32×16 pixels which implies that all the internal buffers (color buffer, depth buffer, stencil buffer) composing the tile frame buffer have this size. The display size resolution was set at 320×240 pixels (a quarter VGA), meaning that the display can be conceptually divided into 10×15 tiles. The graphics accelerator has only one pixel processing pipeline. The fixed-point formats utilized at the interface with the internal datapath are all unsigned. The screen coordinates (X, Y) are represented on 9.4 bits (9 integer, 4 fractional), the color components (R,G,B,A) on 0.8 bits, the depth component (Z) on 0.24 bits, and the stencil component on 8.0 bits.

The "aapoly" OpenGL application from [8] was executed on our virtual SOC platform. The resultant image is presented in Figure 7. It can be seen that the reciprocal employing direct fixed-point table lookup leads to noticeable artifacts in the image, whereas the reciprocal we propose does not introduce artifacts in the image. The results of the hardware synthesis on the reciprocal SystemC RTL model using the algorithm we propose are presented in Table 4.

4. CONCLUSIONS

In this paper we presented a hardware design of a reciprocation suitable for 3D embedded graphics engines. We have shown that for the antialiasing datapath of an embedded QVGA graphics hardware accelerator, for a 14-bit operand, the reciprocal implementation requires an inexpensive operand prescaler, one 1k lookup table with 10-bit entries, and a 5-bit adder, for a maximum relative error of the result of only 1.5% over the entire range of the operand. Hardware synthesis in a typical 0.18μm process technology has indicated that the hardware implementation requires only 1600 standard cells to achieve a latency of 2.5ns.

IC Technology		Std. Cell Library
UMC *Logic18-1.8V/3.3V-1P6M*		VST *e*Si-Route/11
Latency	Std. Cell No.	Total Cell Area
2.5ns	1535	$37587\mu m^2$

Table 4. Reciprocal hardware synthesis results.

5. REFERENCES

[1] T. Akenine-Moller and J. Strom, "Graphics for the Masses: A Hardware Rasterization Architecture for Mobile Phones," *ACM Transactions on Graphics*, vol. 22, pp. 801–808, July 2003.

[2] M. Kameyama et al., "3D Graphics LSI Core for Mobile Phone Z3D," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics hardware*, July 2003, pp. 60–67.

[3] J. Pineda, "A Parallel Algorithm for Polygon Rasterization," in *Computer Graphics (ACM SIGGRAPH '88 Conference Proceedings)*, 1988, vol. 22(4), pp. 17–20.

[4] A. Schilling, "A New Simple and Efficient Antialiasing with Subpixel Masks," in *Computer Graphics (ACM SIGGRAPH '91 Conference Proceedings)*, 1991, vol. 25(4), pp. 133–141.

[5] D. Crisu, S. Cotofana, and S. Vassiliadis, "A Proposal of a Tile-Based OpenGL compliant Rasterization Engine," Tech. Rep. (2002-02), Computer Engineering Laboratory, EEMCS, Delft University of Technology, June 2002.

[6] M. D. Ercegovac and T. Lang, *Division and Square Root:Digit-Recurrence Algorithms and Implementations*, Kluwer Academic Publishers, 1994.

[7] O. Lathrop, D. Kirk, and D. Voorhies, "Accurate Rendering by Subpixel Addressing," *IEEE Computer Graphics and Applications*, vol. 10, no. 5, pp. 45–53, September/October 1990.

[8] M. Woo, J. Neider, T. Davis, and D. Shreiner, *OpenGL Programming Guide, Third Edition, The Official Guide to Learning OpenGL, Version 1.2*, Addison-Wesley, 1999.