Power Operation Accelerator to speed up lighting in 3D Graphics

Young-Su Kwon, In-Cheol Park, and Chong-Min Kyung

Dept. of EE, KAIST,
Kusong-dong, Yousong-gu, Taejon 305-701, Korea
Tel : +82-42-869-3461
e-mail : {yskwon,icpark,kyung}@duo.kaist.ac.kr

Abstract—**This paper presents a design of special hardware developed for enhancing the floating-point power operations which are actively used at the lighting stage to calculate the specular term in 3D graphics geometry engines. The power operation takes just 4 cycles in our floating-point multiplier while it takes about 100-200 cycles in conventional floating-point units. Although an approximation algorithm is employed in the power operation to reduce the hardware complexity required, the error of power value from the developed floating-point multiplier is so minimal that no difference can be found by human eyes.**

I. Introduction

High speed floating-point(FP) units are essential for 3D graphics geometry engines because a large number of FP calculations are needed to obtain high quality images. Therefore, hardware accelerators for 3D graphics algorithms have been actively researched [1][2][3][6].

The 3D graphics pipeline begins with scene modeling and geometry stages which are floating-point intensive, and ends with integer-intensive rendering. The geometry stage performs transformation, clipping, and lighting which are based on FP operations for each polygon and thus it needs increasingly more FP calculation capability as the number of polygons increases. The geometry stage is supported by CPU's such as Pentium, while the rendering is done by graphics cards which have 3D accelerators such as Voodoo, RIVA, and etc. Because the rendering requires a lot of pixel operations, the graphics cards listed above are so highly adapted for the rendering that the performance is over 60-80M pixel/s. Since CPU's or media processors responsible for geometry engine cannot keep the pace with these cards, geometry engines can be the bottleneck that slows down overall 3D graphics performance. The 3DNow! technology is an example of technology employed in general-purpose processors to alleviate the geometry engine bottleneck[4].

The most frequent operation in transformation and clipping is 4×4 matrix multiplication used to translate or rotate images. For the matrix multiplication requiring 4 multiplications and 3 additions, there have been many researches devoted to the fast calculation [1][5]. As an example, a scheme that executes a FP multiplication and a FP addition in parallel is used in [1].

Assuming one vertex per one polygon, Fig. 1 shows the number of operations required for one polygon to do transformation, clipping, lighting and projection if we use only FP add, FP mul, and FP compare instructions. The power, divide or square root operations needed to calculate lighting values were implemented in software using FP add, FP sub, and FP compare instructions. The operation count used in lighting is over a half of the total number of operations performed in the geometry engine.

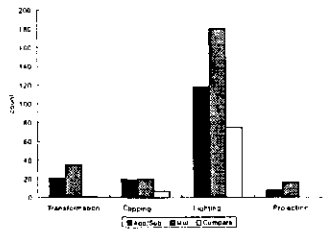


Fig. 1. Operation count in geometry stages.

In the lighting stage, the color value of every vertex should be computed for each light source. Each color component of a vertex is calculated by Phong illumination model [7] shown in Eq. 1.

$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + \sum_i f_{att_i} I_{\lambda_i} [k_d O_{d\lambda} (\overline{N} \bullet \overline{L_i}) + k_s O_{s\lambda} (\overline{R_i} \bullet \overline{V})^{S_{rm}}], \quad (1)$$

where I_λ is the light intensity for one of three color indices : R,G,or B. k,O_d and O_s are a reflection coefficient, a diffuse color and a specular color, respectively. N is the normal vector of the current vertex and $\overline{L_i}$ is the i-th light source's vector from the current vertex. $\overline{R}$ is the reflected light's vector and $\overline{V}$ is the viewer's vector. These vectors are all normalized.

In Eq.1, the first term means an ambient color which models intrinsic intensity and the second term is a diffuse color which exhibits the brightness of an object. The third term is a specular color which exhibits the shininess of the object. The shininess is dependent on the angle, α, between the reflected light vector $\overline{R}$ and the viewer's vector $\overline{V}$, that is the inner product $\overline{R} \bullet \overline{V}$. Phong illumination model assumes that maximum specular reflectance occurs when the angle α is zero and falls off sharply as α increases. This rapid falloff is approximated by $(\overline{R} \bullet \overline{V})^{S_{rm}} = (cos\ \alpha)^{S_{rm}}$, where S_{rm} is the material's *specular reflection exponent*. This effect is shown in Fig. 2.

The specular reflection requires the power operation of two FP numbers as well as the normalization of vectors. The power operation can be calculated by using the machine instructions provided that the power operation is supported in the machine as instructions or by using a software mathematical library. In AMD or Pentium, there

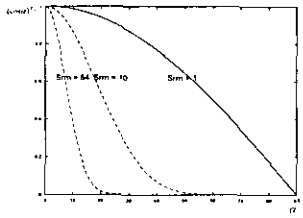


Fig. 2. $(cos\ \alpha)^{S_{rm}}$ used in the Phong illumination model.

are two instructions which calculate $log_2 x$ and 2^x where x is a FP number. With these instructions, $(cos\ \alpha)^{S_{rm}}$ can be computed as in Eq. 2, and it takes about 150 cycles. Since it must be computed for $(vertex\ number) \times (\sharp\ of\ light\ sources)$ times, the number of cycles consumed for the operation is huge.

$$cos^{S_{rm}}\alpha = 2^{S_{rm} \times log(cos\ \alpha)} \tag{2}$$

Eq. 2 can be computed using a software algorithm which uses normal integer and bit-operation instructions. $pow(a, b)$ function of SPARC takes about 140 cycles.

This paper describes a special hardware that computes the power operation of two FP numbers in 4 cycles with small loss of accuracy. The hardware can be easily implemented by expanding a conventional floating-point multiplier. This paper is organized as follows. In section II, the approximation algorithm will be presented and the hardware implementation will be shown. In section III, the measurement of the runtime used in the lighting will be compared. The loss of accuracy caused by our approximation algorithm is so small that human eyes cannot find any differences between the image generated by SPARC and the other one by the proposed approximation algorithm.

II. Approximation of Power operation

As in Eq. 2, $(cos\ \alpha)^{S_{rm}}$ can be computed by the logarithm and exponential operations. We have used a piecewise linear approximation for the $log_2(cos\ \alpha)$ [8].

$$\begin{aligned} & log_2(cos\ \alpha) \qquad (3) \\ & = log_2(2^N(1+x)) \quad (\ N : exponent,\ 1+x : mantissa\ ) \\ & \cong \begin{cases} N + 1.25x & 0 \le x < 0.25 \\ N + x + 0.0625 & 0.25 \le x < 0.75 \\ N + 0.75x + 0.25 & 0.75 \le x < 1.0 \end{cases} \qquad (4) \end{aligned}$$

where N is an unbiased integer exponent of $cos\ \alpha$, and x is the fractional part that does not include the hidden bit. In other words, x represents the lower 23 bits in a single-precision FP number as shown in Fig. 3.

The addends in Eq.4 are easily generated: $1.25x = x + (x >> 2)$ and $0.75x = x - (x >> 2)$. It is quite simple to determine whether $x < 0.25$, $0.25 \le x < 0.75$, or $x \ge 0.75$. If x's 2 MSB's are "11", x is greater than or equal to 0.75. If x's 2 MSB's are "10" or "01", x is between 0.25 and 0.75. Otherwise, x is less than 0.25. A circuit for finding the range of x is shown in Fig. 4.

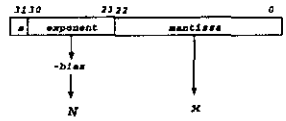


Fig. 3. "N" and "x" fields in $cos\ \alpha$.

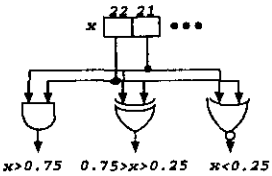


Fig. 4. Determination of the range of x.

As an example, let us consider the case of $x < 0.25$. Its approximated $log(cos\ \alpha)$ can be calculated as in Eq. 5. Since $cos\ \alpha$ is between 0 and 1, its exponent is always smaller than the bias, and the sign of the approximated value is negative.

$$\begin{aligned} N + 1.25x &= N + x + (x >> 2) \\ &= -(\ (bias - exponent) - x - (x >> 2)\ ) \\ &= -(\ (bias - (exponent + x)) - (x >> 2)\ ) \\ &= -(\ (bias + \bar{}(cos\ \alpha) + 1) + \bar{}(x >> 2) + 1\ ) \\ &= -(\ (bias + 2) + \bar{}(cos\ \alpha) + \bar{}(x >> 2)\ ) \qquad (5) \end{aligned}$$

The other cases when $0.25 \le x < 0.75$ and $x \ge 0.75$ can be formulated similarly, and the resulting equations for $log_2(cos\ \alpha)$ are summarized in Eq. 6.

$$log_2(cos\ \alpha) \cong \begin{cases} -(\ (bias + 2) + \bar{}(cos\ \alpha) + \bar{}(x >> 2)\ ) \\ \qquad for \quad 0 \le x < 0.25 \\ -(\ (bias + 1 - 0x00200000) + \bar{}(cos\ \alpha)\ ) \\ \qquad for \quad 0.25 \le x < 0.75 \\ -(\ (bias + 1 - 0x00080000) + \bar{}(cos\ \alpha) + (x >> 2)\ ) \\ \qquad for \quad 0.75 \le x < 1.0 \end{cases} \tag{6}$$

In Eq. 6, three 32 bit additions are required. It can be implemented by a CSA(carry save adder) and one 32 bit carry select adder. The CSA accepts three operands and generates a carry and a sum. The final 32 bit adder adds the carry and the sum and generates a final 32 bit result. A hardware implementation for the log approximation unit is shown in Fig. 5, where "const" means the constant in Eq. 6.

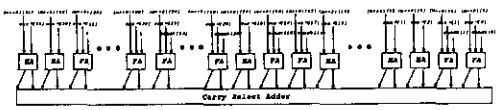


Fig. 5. Log approximation unit.

The output of the log approximation unit is a 32 bit fixed point number and it is shifted by 7 bits to make a 24 bit

fixed point number. S_{rm} is saved in a special register as a fixed point number whose binary point is at the 16th bit. Upper 8 bits are an integer part and lower 16 bits are a fractional part. The approximated $log(cos\ \alpha)$ and S_{rm} is multiplied by the 24-bit multiplier in the FP multiplier unit. When this multiplier is used for the FP multiplication, it multiplies mantissas of two operands, but when used for log approximation it multiplies two fixed point numbers. The 24-bit multiplier is composed of 2 stages. The first stage is a CSA tree which has a Booth encoded Wallace tree structure and the second is a carry select adder which adds the sum and the carry generated by the CSA tree. After the multiplication, the result is shifted by 9 bits to generate a 23 bit mantissa. If the integer part of the result is over 8 bit, an overflow occurs, the result becomes the maximum number which the result can represent.

The approximation equation for 2^x is shown in Eq. 7.

$$\begin{aligned} 2^{S_{rm} \times log(cos\ \alpha)} &= 2^{-n-y} \\ &= 2^{-(n+1)+(1-y)} \quad (n : integer, 0 < (1-y) < 1) \\ &\cong 2^{-(n+1)}(1 + \eta) \quad (0 < \eta < 1, \eta = (1-y)) \end{aligned} \tag{7}$$

Because $n+1$ is an unbiased exponent, it is required to add the bias to the integer part to generate a FP number. Using the 24×24 multiplier in the FP multiplier unit, the circuitry for log approximation and exponential approximation can be merged into the FP multiplier. A block diagram of the merged FP multiplier is shown in Fig. 6, which is called "Fastpow" unit.

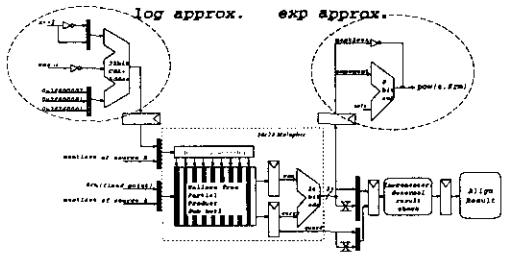


Fig. 6. Floating Point multiplier with "Fastpow" unit.

As $(cos\ \alpha)$ increases, $(cos\ \alpha)^{S_{rm}}$ increases monotonously. If the proposed approximation equation doesn't monotonously increase as $cos\ \alpha$ increases, the image using this approximation equation can be looked as significantly different. A brighter point in the original image can be seen as a darker point compared to the neighbour points. The monotonous increase of the approximation equation 4 and 7 used in "Fastpow" unit can be easily proved. Therefore, the image is almost the same as the original image if the error is not so large.

III. Results

To prove the importance of the lighting stage in 3D graphics geometry engine, we measured the runtime taken by the lighting stage for several applications that draw simple 3D objects having specular reflection. The runtime percentage of the lighting stage in a geometry engine is shown in Fig. 7. About 30% of total runtime is consumed in the lighting stage.

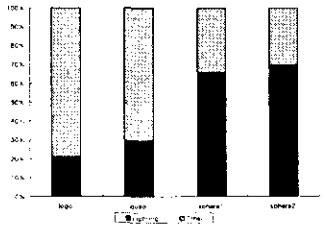


Fig. 7. The runtime percentage of the lighting stage in geometry stage.

Fig. 8 shows the type of operations used in the lighting stage, which was measured using an openGL compatible library. Although the operation count of power operation

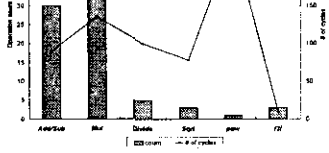


Fig. 8. Operation count and cycle count required in the lighting in the case of TI's TMS320C67x.

and square root operation is relatively small compared to that of addition or multiplication, the cycle count of power and square root operations is larger than that of addition or multiplication.

The method to calculate the specular term can be classified as follows.

- Instructions to compute log, exponential operation.
- Software implementation.
- Table lookup method.

If instructions are used, about 160 cycles are used for $log_2 x$ and 2^x instructions in Pentium or AMD's K6. If a software algorithm is used, about 140 cycles are required in SPARC and over 200 cycles are required in TI's TMS320C67x. With a table lookup method, about 512 values of $cos\ \alpha$ are required and the differences of adjacent $cos\ \alpha$ values are saved in the tables for interpolation. Besides an additional memory to save the tables, the table must be updated at any time a new object appears. Since large overheads are inevitable in the specular term calculation, applications that need fast scene update such as 3D games have not used the specular reflection.

Our "Fastpow" unit takes just 4 cycles to compute the power of FP numbers. Therefore, the specular term calculation is very fast compared to the other methods. The hardware overhead is 32 CSA's, one 32-bit carry select

adder and one 8-bit adder. The cycle counts of "Fastpow" unit for the various operations are compared to the other processors in Fig. 9. Our FP unit uses Newton-Raphson method to compute sqrt or divide operations like TI's TMS320C67x. Therefore, the cycle count of our FP unit for the square root or divide operation is almost the same as the TI's TMS320C67x. Two FP instructions were used to compute the power operation for Pentium and a software mathematical library was used for TI's TMS320C67x. For the power operation, the cycle count required in "Fastpow" unit is roughly reduced to 1/40 of the others.

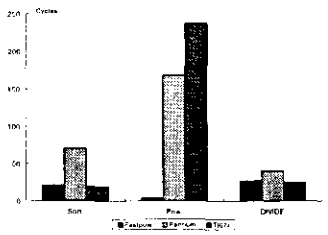
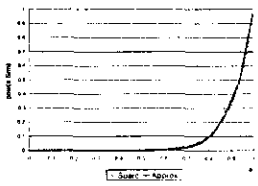
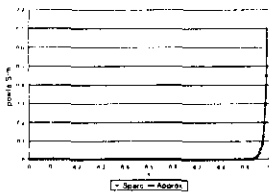


Fig. 9. Cycle count of FP units to compute various operations.

Fig. 10 compares the power value generated by the *pow*() function of SPARC and the value from the "Fastpow" unit for $S_{rm} = 10.16$ and $S_{rm} = 80.01$. The approximated value is very similar to the value generated by SPARC's *pow*(). The range of Y-axis is 0 to 1 because *cos* α's value is between 0 and 1. In our measurement, the mean absolute error between the SPARC's value and the "Fastpow"'s value was 0.000018 and the distribution of error was 0.000158.



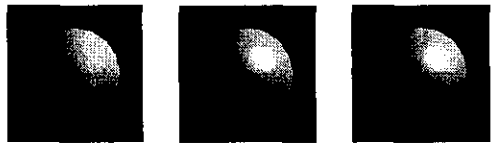
(a) S_{rm}=10.16



(b) S_{rm}=80.01

Fig. 10. *cos* $\alpha^{S_{rm}}$ value generated by SPARC and "Fastpow".

To inspect the effect of error, we have used our algorithm instead of the *pow*() function in 3D API. The resulting image is compared in Fig. 11. Image (a) does not use the specular term and thus the highlight does not appear in the upper right corner. Image (b) uses the specular term whose power value is calculated by the SPARC's *pow*() function. Image (c) uses our approximation algorithm of the Fastpow. As shown in the figure, image (b) and (c) don't show any difference.



(a) Without specular term.

(b) With specular term using SPARC's *pow*().

(c) With specular term using fastpow.

Fig. 11. Image with different method of specular calculation.

IV. Conclusion

In this paper, we proposed the "Fastpow" unit which accelerates the calculation of the lighting value in 3D graphics. "Fastpow" unit calculates the power value of two floating-point numbers in 4 cycles with small loss of accuracy while it takes over 150 cycles in other processors. "Fastpow" is easily merged into a conventional floating-point multiplier and its hardware overhead is only a 32-bit CSA, one 32-bit adder and one 8-bit adder. We have also shown that the error is so small that there is almost no difference between two images generated by using the SPARC's *pow*() function and "Fastpow" unit, respectively.

References

[1] O. Nishii *et al.*, *"A 200MHz 1.2W 1.4GFLOPS Microprocessor with Graphics Operation Unit,"* *ISSCC Digest of Technical Papers*, Feb. 1997, pp. 402-403.

[2] Hiroshi Makino *et al.*, *"A 286 MHz 64-b Floating Point Multiplier with Enhanced CG operation,"* *IEEE J. Solid-State Circuits*, Apr. 1996, pp. 504-512

[3] J. Shipnes, *"Graphics processing with the 88110 RISC microprocessor,"* in *Dig. of Papers, IEEE Proc. COMPCON*, '92, pp. 169-174.

[4] *"3DNow! Technology, Delivering Leading-Edge 3D Graphics and Multimedia Performance for the New Era of Realistic Computing,"* Advanced Micro Devices, INC., May 1998.

[5] J. Grimes, *"The Intel i860 64-bit Processor:A General-Purpose CPU with 3D Graphics Capabilities,"* *IEEE Computer Graphics and Applications*, Vol. 9, No .4, July 1989, pp.85-94.

[6] J. H. Clark, *"The Geometry Engine: A VLSI Geometry System for Graphics,"* *Computer Graphics*(Proc. Siggraph), Vol. 16, No. 3, July 1982, pp.127-133.

[7] Bui-Tuong, Phong, *"Illumination for Computer Generated Pictures,"* *CACM*, 18(6), June 1975, 311-317.

[8] M. Combet, H. Van Zonneveld, and L. Verbeek, *"Computation of the Base Two Logarithm of Binary Numbers,"* *IEEE Trans. Electron. Comput.*,, June 1975, 863-867.