Bulletin of Aichi Institute of Technology
Vol.26 Part B March 1991

Logarithmic Number Systems for Picture Generation

対数表現数値システムによる画像生成

Tomio KUROKAWA+ and Takanari MIZUKOSHI++

黒河富夫＋ 水越剛成＋＋

Abstract Logarithmic arithmetic (LA) is a very fast computational method for real numbers. And its precision is better than a floating point arithmetic of equivalent word length and range. This paper shows a method to use LA in computer graphics---picture generation of almost any kind. Various experiments are done---from curve drawing to 3D image generation. The results are all excellent for quality and speed.

1. Introduction

LA is a third type of arithmetic which uses a number system called logarithmic number system (LNS). Its fast addition/subtraction method was first introduced by Kingsbury et al.[1] In LNS, a number is expressed by a binary sequence[2]:

$$sd_0d_1...d_m.l_1l_2...l_n, \quad (1)$$

where s, d_i, and l_j are *0* or *1*; s and d_0 are the sign of the number and the exponent, respectively; the d-part and l-part combined represents the exponent of the number; the base is assumed to be a constant a (greater than 1); [.] is the assumed binary point of the exponent. Then, the sequence (1) indicates the number,

$$\pm a^{d\text{-part}.l\text{-part}}. \quad (2)$$

Due to the number expression form, many computations become simple. Let a^x and a^y be two numbers expressed as in the form of (2). In LA, multiplication/division, square/square root, addition and subtraction can be done as indicated by the expressions (3), (4), (5) and (6), respectively.

$$a^x * a^y = a^{x+y};\ a^x / a^y = a^{x-y}. \quad (3)$$

$$(a^x)^2 = a^{2x};\ (a^x)^{1/2} = a^{x/2}. \quad (4)$$

If $a^z = a^x + a^y = a^x(1 + a^{y-x})$,

then $z = x + \log_a(1 + a^{y-x})$, $(x \geq y)$. (5)

If $a^z = a^x - a^y$,

then $z = x + \log_a(1 - a^{y-x})$, $(x \geq y)$. (6)

As shown in (3), LA multiplication/division is equivalent to fixed point number addition/subtraction; square/square root, as (4) shows, becomes equivalent to shift operation. If $\log_a(1+a^{y-x})$ is pre-computed as a look-up table with the table address $y-x$, then z can be obtained quickly as (5) shows. Subtraction can be done in a similar fashion with $\log_a(1-a^{y-x})$, the additional lookup table.

2. Method of Fixed Point Number Computation in LNS

LNS is very effective if it is used as in Fig. 1.

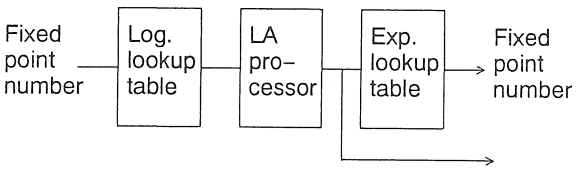

Fig.1 Method of fixed point number computation using LA.

+ Department of Industrial Engineering, Aichi Institute of Technology (Toyota-shi)
++ Oki Technosystems Laboratory (Nagoya-shi)

It is for fixed point number computations. This method is very simple but effective. In computing fixed point numbers, the numbers are first converted to logarithmic numbers (LN); then computations are done in LNS; and whenever necessary, the results are converted back to fixed point numbers. Those conversions of integer to LNS and vice versa can be done by look-up tables (LUT). Accordingly the processing should be very quick.

Digital pictures are usually defined as a two variable function f(x,y), where f is the pixel intensity and x and y are coordinate addresses. In many cases, all of f, x, and y are fixed point numbers (usually integers). Then f(x,y) can be generated using the above method. For curve drawing, the dot generation can be done by computing y=g(x) or x=h(y), where g and h are curve functions.

3. Experiments

Using the above method, some experiments were done to evaluate the quality and the speed of curve drawing, geometrical picture transformation, ray tracing, and fractal image generation. The comparisons were done on a personal computer between two kinds of computer programs: one is the pure software which employs LNS, the other is the software which uses a floating point hardware chip (FP). The specific conditions are as follows:

1) Used computer: a personal computer, PC-98XL with 80286:8Mz CPU.
2) LNS program: 16 bit LNS implemented by C language. There are two type of programming. One is called "procedure call" (subroutine call); the other is "non-procedure call". In "procedure call" programming, arithmetic functions of addition/subtraction, multiplication/division and square/square root are realized in procedures. The computations are done by calling them. This programming is used for polynomial curve, geometrical transformation, fractal image and ray tracing. "Non procedure call" is a programming in which LNS arithmetics are realized without using procedures. This programming is used for circle and ellipse drawing by LNS.
3) FP program: 80287,10Mz (PC-98XL-03) processor used in C program, 32 or 64 bit word FP is used.

Curve drawing: Figure 2 shows the circles drawn by the method. They are obtained by computing the following with each integer of x:

$$y = \sqrt{R^2 - x^2}. \tag{7}$$

Figure 3 is the circles by FP. There are no difference for a look. An ellipse can be generated by computing

$$y = (b/a)\sqrt{a^2 - x^2}. \tag{8}$$

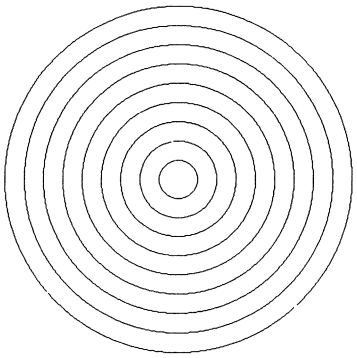

Fig.2 Circles generated by LNS (16 bit: a=2, m=4, n=10). R = 20, 40, 60, ..., 180.

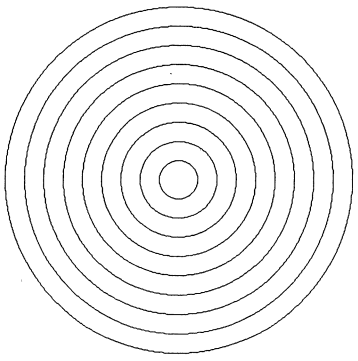

Fig.3 Circles generated by FP (64 bit). R = 20, 40, 60, ...,180.

Figure 4 shows the ellipses drawn by the method. And Fig.5 is the curve of the polynomial:

$$y = x^5 + 2x^4 - 7x^3 - 8x^2 + 12x, \text{ and}$$

$$py = p(((((1.0t + 2.0)t - 7.0)t - 8.0)t + 12.0)t + 0.0), \quad (9)$$

where x and y are scaled by 50 and 10, respectively, that is, p=10, t = s/50 = x, and s is integers. The scaling is to draw the curve of Fig.5 and to avoid the overflow within the number system. The computation order is by (9). Gaps are interpolated by straight lines if they occur. The specific LNS used is of a=2, n=4 and m=10, which is of 16 bit word, for all the three kinds of curves. The experimental speed comparisons between LNS and FP are shown on Table 1. Note that the circle and ellipse drawing (non-procedure call) by LNS is drastically faster than FP method. The circle is mostly by integer computations and by 64 bit FP square root procedure (a vendor's library); the ellipse and polynomial are by 64 bit FP hardware.

The object program sizes of circle drawing are 104 bytes for LNS and 74 bytes for FP, both very small.

Geometrical transformation of pictures: Bi-level picture (size 100 x 100) of an alphabet "K" is used for the affine transformation with scaling of 6x6 and rotation of 30° (see Fig.6 and 7). Three

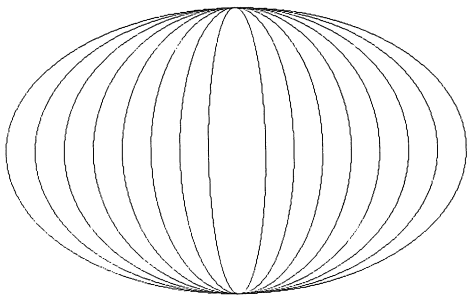

Fig.4 Ellipses generated by LNS (16 bit: a=2, m=4, n=10). b = 150; a = 30, 60, 90, ..., 240.

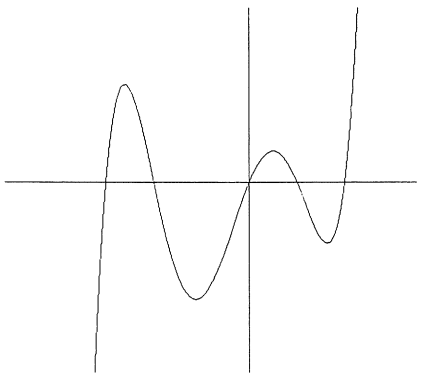

Fig.5 Polynomial curve by LNS (16 bit: a=2, m=4, n=10). $y = x^5 + 2x^4 - 7x^3 - 8x^2 + 12x$.

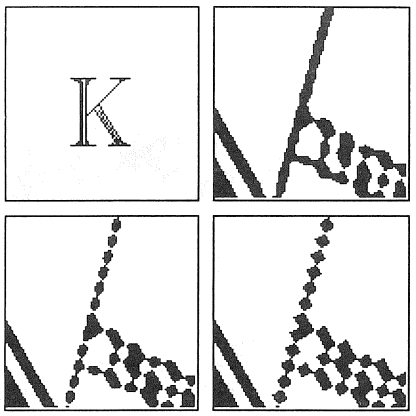

Fig.6 Affine transformation of a bi-level picture using LNS (16 bit: a=2, m=5, n=9).

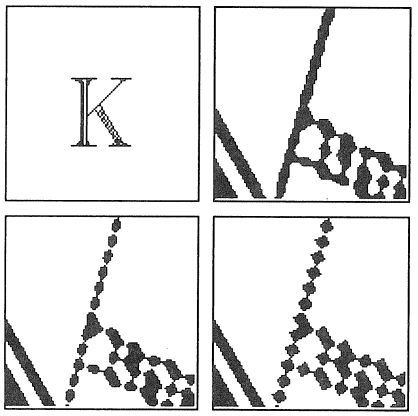

Fig.7 Affine transformation of a bi-level picture using FP (32 bit).

*Note: Upper left (original); Upper right(cubic convolution); Lower left (bi-linear); Lower right (nearest neighbor) for Fig.6 and 7 also for 8 (on next page).

Table 1 Experimental speed comparison between LNS and FP for curve drawing. Data is in seconds per one dot generation. LNS circle and ellipse are by "non-procedure call" and LNS polynomial is by "proce dure call".

	LNS	FP
circle	1.12×10^{-5}	1.45×10^{-4}
ellipse	1.27×10^{-5}	1.85×10^{-4}
polynomial	5.31×10^{-4}	5.47×10^{-4}

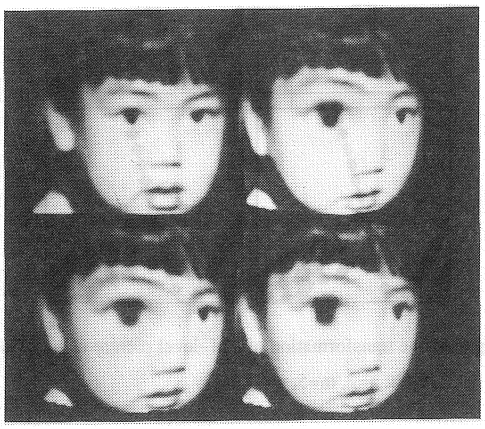

Fig.8 Non-linear transformation of a gray level picture using LNS (16 bit: a=2, m=5, n=9).

Table 2 Experimental speed comparison between LNS and FP (hardware, 32 bits) for geometrical transforma tion of pictures. A picture of size 100x100 is mapped to a picture 100x100. LNS is by "procedure call".

	Affine Trans.		Non-linear	
	LNS	FP	LNS	FP
Nearest neighbor	2	3	3	5
Bi-linear	6	7	7	9
Cubic conv.	35	35	36	38

interpolation methods, "nearest neighbor," "bi-linear," and "cubic convolution" are tried for LNS and FP. The resulting pictures by LA and by FP look alike with no particular difference. The second geometrical transformation is non-linear[3]. A gray level picture (a child head, size 100x100) is used for the mapping:

$$x_0 = a_1 x_1 (1+b_1\sqrt{x_1^2+y_1^2}),$$
$$y_0 = a_2 y_1 (1+b_2\sqrt{x_1^2+y_1^2}). \quad (10)$$

Constants of $a_1 = a_2 = 0.5$ and $b_1 = b_2 = 0.02$ are used. The results are shown in Fig.8. Table 2 shows the speed comparison between LNS and FP (hardware, 32 bits) for the above two kinds of mapping.

Fractal image and 3D computer graphics: Further experiments are done for picture generations of fractals and ray tracing. Figure 9 is a fractal image and Fig.10 is a ray tracing picture, both made by the method (LNS of a=2, m=5 n=9). Figure 11 is generated by FP (hardware, 32 bits) method, which is presumably the same picture as Fig.10. There are not big differences between the two. All the three pictures were originally with colors. As for the speed comparison, LNS (Fig.10) took 5 minutes and 44 seconds and FP method (Fig.11) took 5 minutes and 56 seconds, both excluding the display time. Although the programming with LNS is with "procedure call", which are time consuming at run time, it is as fast as the FP method. The speed of the fractal image generation with "procedure call" is equivalent to that of FP also.

Lookup table size: The size of lookup table is 2^{m+n+2} entries for LA addition and subtraction combined with no reduction. This size could be reduced to 1/4 or more very easily[4]. For the conversion, the size of LUT (from LNS to integer) is 2^{m+n+1} with no reduction. This can be reduced, too, depending on the output range. LUT (from integer to LNS) should be small; 1000 entries could be enough for most cases.

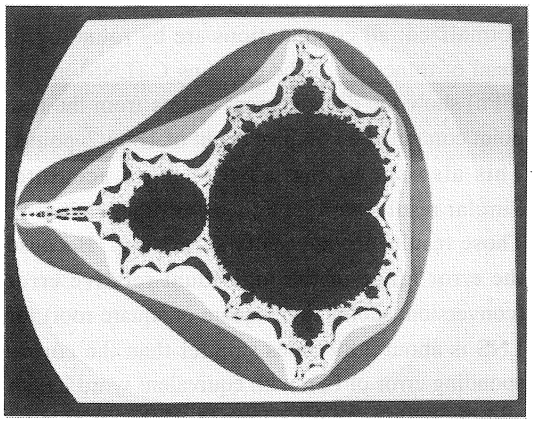

Fig.9 Fractal image by LNS (16 bit: a=2,m=5,n=9).

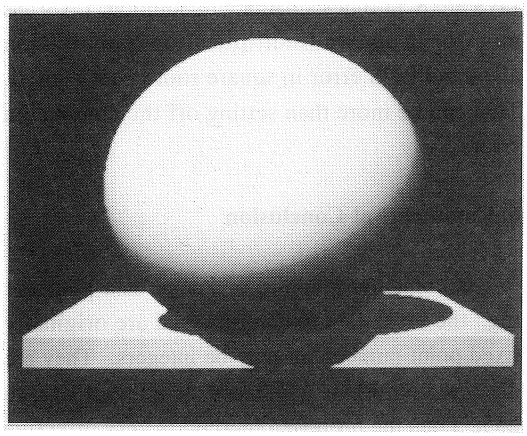

Fig.10 Ray tracing by LNS (16 bit: a=2,m=5,n=9).

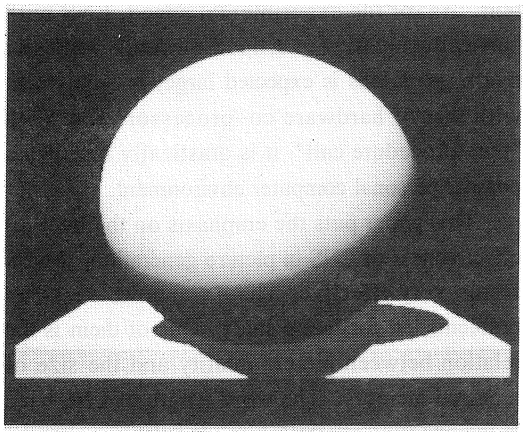

Fig.11 Ray tracing by FP (32 bit).

Speed of single computations and the polynomial: The speed comparison of individual computations between LNS procedures (library) and FP (32 and 64 bits) computations is shown in Table 3. LNS software is faster than FP hardware in division, square, and square root. For the addition and the subtraction, the order is reversed. The FP software is much slower. Note that the square root of 32 bit FP hardware is slower than that of 64 bit FP hardware.

The above results (16 bit LNS and 64 bit FP) roughly agree with the data (polynomial case) of Table 1. According to the results of Table 3 and (9): five additions, six multiplications and one division, however, 64 bit FP hardware polynomial, which should take about $4.9\text{x}10^{-4}$ seconds for a dot without the overhead such as the loop, should be slightly faster than that of 16 bit LNS, which should take about $5.3\text{x}10^{-4}$ seconds. It differs a little from the results of Table 1. It is probably due to the fact that the actual LNS software is so implemented as to be capable of handling zeros, and the addition of zero is faster; it takes only $1.7\text{x}10^{-5}$ seconds instead of $5.6\text{x}10^{-5}$. The addition of zero exists in (9). The polynomial with 32 bit FP (hardware), which should take $4.1\text{x}10^{-4}$ seconds for a dot, should be somewhat faster than LNS software.

5. Error Size on Circle Drawing

Since the word size to represent a number is limited, the conversion or computation error cannot be avoided. LNS is no exception. LNS (n bit fraction) and FPn (FP with n bit fraction) are compared in error size for circle drawing. Let y_l and y_f be the computational result (just before final conversion to integers) of (7), for LNS and FPn respectively. And let y be the true result (computed by 64 bit FP). Then the errors e_l for LNS and e_f for FPn are expressed by

$$e_l=y_l-y, \qquad e_f=y_f-y. \tag{11}$$

With above errors, the error to signal ratios

$$\sqrt{\sum e_l^2 / \sum y^2}, \text{ and } \sqrt{\sum e_f^2 / \sum y^2} \quad (12)$$

are computed for both of LNS and FPn for the octant of R=180. The results are shown on column A and B of Table 4. The error to signal ratios of FPn are about two times larger than those of LNS. LNS is defined by a=2, m=4 and n=10 to 15; and FPn is defined by a=2 m=4 n=10 to 15 with m+n+2 bit (same as LNS) floating point number system as

$$f \times a^e,$$

where f (n+1 bits) and e (m+1 bits) both in 2'complement form for negative numbers and f normalized; all computations are by rounding in stead of truncation. The columns C (LNS) and D (FPn) show the disagreement counts from the true point computed by 64 bit FP, out of $180/\sqrt{2}$ points. This also shows that LNS is more accurate. Similar results were obtained for other radius R. Those results are not really surprising. Because the error range of the individual relative error (conversion, subtraction, square or square root) for LNS is about 2.89 times smaller than the corresponding error of FPn with equivalent word length and dynamic range[2]. That is, if the individual relative error distributions are uniform for both number systems, the relative error variances of LNS are 8.35 times smaller. In addition, LNS has no error in square (multiplication and division also) and LNS error in square root is very small. That can be more than setting off the conversion error.

Table 3 Speed comparison for single computations between LNS and FP. Time unit is in seconds.

	16 bit LNS	32 bit FP (80287)	64 bit FP (80287)	FP soft 32 bit
a+b	5.6×10^{-5}	3.3×10^{-5}	3.8×10^{-5}	1.8×10^{-4}
a−b	5.8×10^{-5}	3.3×10^{-5}	3.8×10^{-5}	2.0×10^{-4}
a*b	3.5×10^{-5}	3.4×10^{-5}	4.2×10^{-5}	1.9×10^{-4}
a/b	3.5×10^{-5}	4.4×10^{-5}	4.9×10^{-4}	2.2×10^{-4}
a^2	1.7×10^{-5}	3.3×10^{-5}	3.8×10^{-5}	1.9×10^{-4}
$\sqrt{a}$	1.4×10^{-5}	1.7×10^{-4}	1.2×10^{-4}	9.6×10^{-4}

Table 4 Error size comparison for circle drawing. LNS and FPn used are of a=2, m=4 n=10 to 15. The radius of the circle is 180.
A is the error to signal ratio for LNS;
B is that for FPn;
C is the number of disagreement from the true point for LNS;
D is that for FPn.

n	A	B	C	D
10	1.97×10^{-5}	3.95×10^{-5}	5	23
11	9.01×10^{-6}	1.89×10^{-5}	5	11
12	4.83×10^{-6}	9.80×10^{-6}	1	8
13	2.34×10^{-6}	5.06×10^{-6}	1	1
14	1.15×10^{-6}	2.75×10^{-6}	0	1
15	5.54×10^{-7}	1.22×10^{-6}	0	1

6. Comments at Conclusion

In computer graphics or image data processing, the most data to be processed are originally fixed point numbers or may be integers. They are discrete coordinate addresses or integers of 0 to 255. Thus the proposed method is very effective in very wide area of computer graphics---image generation of almost any kind. The picture quality and the processing speed are surprisingly excellent. As for the experimental speed, LNS with "procedure call" (overhead of stack, call and return operations is expected large) is equivalent with that of hardware co-processor; and with "non-procedure call" it is drastically faster in a popular personal computer environment.

This paper puts the emphasis on the over all effectiveness of LNS in picture generation. And it seems very effective. There are, however, some problems left for future study. One of them is the relation between picture quality and the size of required memory. The word length of LNS used in the experiments is 16 bits. It is known that the longer the word, the more accurate the computations become, thus the better picture quality is

obtained. But the word length of LNS cannot be extended unlimited because the required memory size (look-up tables) is generally doubled with each one bit extension. Experimental accuracy comparison is made between LNS and FP for circle drawing. The result is that LNS is more accurate for equivalent word length and range.

Further study is desired to investigate how much memory is required for what kind of picture quality.

References

(1) N. G. Kingsbury and P. J. W. Rayner: "Digital Filtering Using Logarithmic Arithmetic," Electron. Lett., 7, 2, pp.56-58 (1971).

(2) T. Kurokawa, J. A. Payne and S. C. Lee: "Error Analysis of Recursive Digital Filters Implemented with Logarithmic Number Systems," IEEE Trans. on ASSP, ASSP-28, pp.706-715 (1980).

(3) M. Shiono: "Generation of Various Hand-written Style Character Patterns Using Nonlinear Distortion," (in Japanese), IPSJ SIG Reports, 89-CG-41-8 (1989).

(4) A. D. Edgar and S. C. Lee: "Focus Microcomputer Number System, Comm. ACM, vol.22, pp.166-177 (1979).

(Received March 20, 1991)