A Fast and Simple Method for Curve Drawing —A New Approach Using Logarithmic Number Systems—

TOMIO KUROKAWA* and TAKANARI MIZUKOSHI**

A completely new but effective method is introduced for drawing curves on computer graphics. While most curve drawing methods employ "incremental"-type algorithms, the method presented here simply computes a mathematical expression for a curve to generate dot addresses. It employs a logarithmic number system (LNS). To generate the curve of $f(x, y)=0$, for example, the curve expression should be in the form: $y=g(x)$. Then, y is directly computed for each x, using the LNS. Since the arithmetic in an LNS is extremely fast and accurate, the speed and the curve quality are naturally expected to be good. The specific procedure for drawing the curve of $y=g(x)$ is as follows: For each x, (1) convert x (integer) into the LNS by using a lookup table; (2) compute y, using the LNS; (3) convert the resultant y into an integer by using another lookup table; (4) plot the point (x, y). Some software experiments were done on a micro-computer to generate circles and ellipses. They showed that both the speed and the quality are surprisingly good. The former is comparable to or possibly faster than that of the fastest "incremental" algorithm. The latter is also very good, but depends on the specific LNS used. The better the quality desired, the longer the word length required and consequently the more memory required. However a practically high level of quality can be obtained with fairly little memory.

1. Introduction

Generation of curves has been one of the major interests in computer graphics. Much research has been done on drawing curves of circles, ellipses, and so on [1-5].

The standard method of developing efficient algorithms is to make a very close analysis of the curve equation so that the algorithms consist of only the simplest of integer computations. Recently, the algorithms have become very simple and fast.

On the other hand, our logarithmic arithmetic (LA) method is based on computations. To generate the curve of the equation $f(x, y)=0$, for example, we simply compute y directly from the equation. That is to say, we directly compute y of $y=g(x)$ by using a third type of arithmetic called logarithmic arithmetic (LA), which employs a logarithmic number system (LNS). It is like using floating point (FP) arithmetic but without sacrificing the computational speed. It is conceptually and algorithmically simple, and it is very general, being applicable to many types of curves. As long as a curve is expressed as $y=g(x)$, assuming that g is computationally possible, it is always possible to draw it. What is more, the coefficients of the curve equation need not be integers. The LA method also has many other advantages.

2. Logarithmic Arithmetic

In LA, a number is expressed by a binary sequence:

$$sd_0d_1 \ldots d_m \,.\, 1_1 1_2 \ldots 1_n \tag{2.1}$$

where s, d_i, and 1_j are 0 or 1; s and d_0 are the sign of the number and the exponent, respectively; the d-part and 1-part combined represent the exponent of the number; the base is assumed to be a positive constant a (greater than 1); and [.] is the assumed binary point of the exponent. Then, (2.1) indicates the number,

$$\pm a^{d\text{-part} \,.\, 1\text{-part}}. \tag{2.2}$$

In LA, multiplication and division are simple, being equivalent to fixed point addition and subtraction, respectively. Addition and subtraction are a little more complex, but they can be done quickly by using pre-computed look-up tables [6]. To explain LA addition, let a^x and a^y be two numbers, and let a^z be the result of the addition. Then z is expressed as follows:

$$a^z=a^x+a^y=a^x(1+a^{y-x}), \quad \text{then and therefore} \tag{2.3}$$

$$z=x+\log_a(1+a^{y-x}), \quad (x \geq y). \tag{2.4}$$

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 computed quickly. Subtraction can also be done in a similar

*Department of Industrial Engineering, Aichi Institute of Technology, 1247 Yachigusa, Yagusa-cho, Toyota 470-03, Japan.
**Oki Technosystems Laboratory, 3-8-10 Uchiyama-cho, Chigusa-ku, Nagoya, 464, Japan.

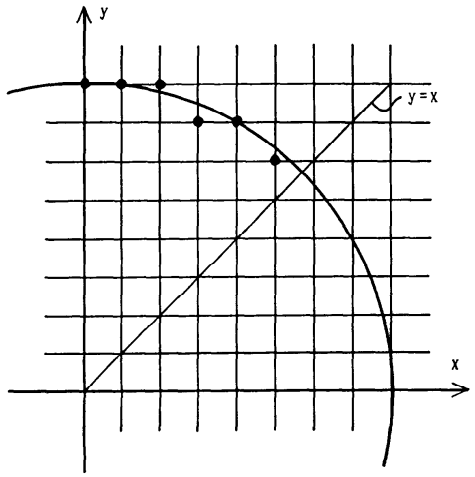

Fig. 1 Curve Dots of an Octant.

fashion [6]. Let

$$a^z=a^x-a^y, \quad \text{then} \tag{2.5}$$

$$z=x+\log_a(1-a^{y-x}), \quad (x\geq y). \tag{2.6}$$

Furthermore, power computations are very simple, unlike in FP arithmetic. For example, the square of a number a^x can be obtained by simply shifting x one bit to the left and the square root by shifting x one bit to the right.

3. Circle and Ellipse Drawing

Consider how to draw a circle expressed by

$$x^2+y^2=R^2. \tag{3.1}$$

It can be done by generating the curve dots shown as the octant in Fig. 1. The entire circle can be obtained by symmetry. To generate the octant dots, we directly compute y in the equation

$$y=\sqrt{R^2-x^2}, \tag{3.2}$$

for each integer of x. This sort of computation is usually done by using FP arithmetic. It is very attractive because it is conceptually simple and flexible to apply for other curves. But it is slow, especially because of the square and square root computations.

LA provides a fast and accurate computational method for real numbers [6–12]. In order to enjoy the features of LA for integer computation, we need to convert integers into logarithmic numbers. To compute y in (3.2), consider the following procedure:

(1) Convert an integer (x) into a logarithmic number by using a lookup table.

(2) Do the required computations in LNS.

(3) Convert the resultant logarithmic number back into an integer by another lookup table.

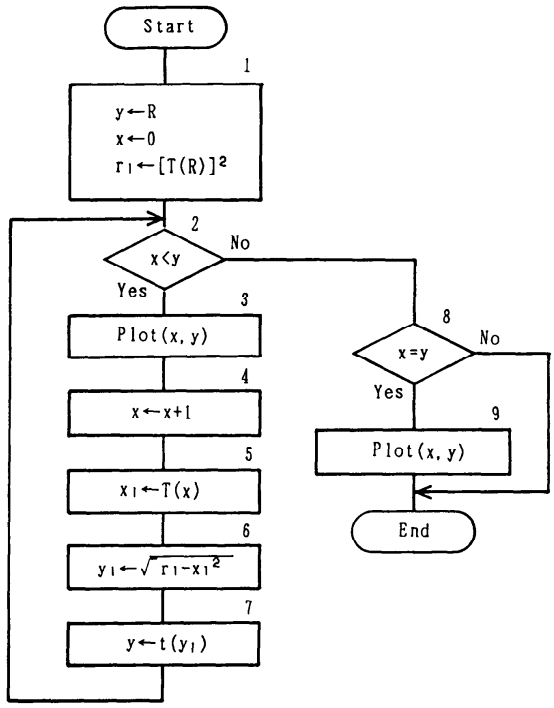

Fig. 2 Algorithm for Circle Dot Generation.

To find y in (3.2), all we need to do is to compute the squares, the subtraction, and the square root and to do number conversions by using the lookup tables. The specific algorithm is shown in Fig. 2. $T(\#)$ means a table lookup conversion from an integer to LNS, and $t(*)$ from LNS to an integer. Both tables are assumed to be pre-computed. R, x, and y are integers, and r_1, x_1, and y_1 are logarithmic numbers of the form (2.1). The overall flow structure is made to resemble that of the octant drawing algorithm [5] so that later comparisons should become easier. The essential part is the computation of y_1 in box 6. It consists of three operations: one square, one subtraction, and one square root. The square and square root operations can be done as one bit shift in LA. The entire algorithm of Fig. 2 they becomes very fast.

An ellipse can be drawn in a similar fashion. Consider the ellipse

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1. \tag{3.3}$$

$y(>0)$ is expressed as

$$y=\sqrt{b^2-(b^2/a^2)x^2}. \tag{3.4}$$

The expression (3.4) is essentially the same as (3.2). There is only one difference, the additional multiplication of (b^2/a^2) and x^2. Multiplication is not a big burden for LA. Figure 3 shows a conceptual dot curve

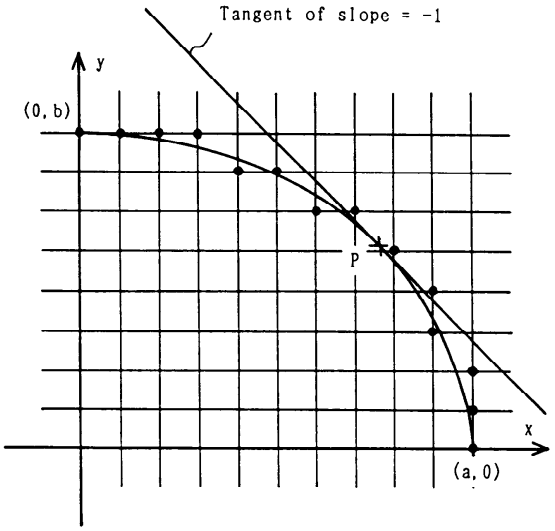

Fig. 3 Curve Dots of a Quarter Ellipse.

of a quarter ellipse. P is the point at which the tangent line of slope $= -1$ touches the ellipse. The curve is to be drawn in two steps so as to avoid creating gaps along it. The first step is from the point $(0, b)$ to P with x the independent variable. The second is from $(a, 0)$ to P with y the independent variable.

The coordinates of P can be obtained as follows. Differentiating Eq. (3.3) with respect to x and setting the derivative dy/dx to -1, we have

$$y=(b^2/a^2)x \tag{3.5}$$

and the tangent point,

$$\left(\frac{a^2}{\sqrt{a^2+b^2}}, \frac{b^2}{\sqrt{a^2+b^2}}\right). \tag{3.6}$$

Figure 4 shows an algorithm for drawing the quarter ellipse. This is essentially the same as the algorithm in Fig. 2; a, b, x, y and M_m are integers, and a_2, b_2, L_m, c_1, c_2, x_1, and y_1 are logarithmic numbers of the form (2.1). There is no significant difference between Fig. 2 and Fig. 4. The non-significant differences are in boxes of 1, 7, 9, and 15. Box 1 is for initial constant computations. The biggest difference is the multiplications of c_2 and x_1^2 in box 7 and of c_2 and y_1^2 in box 15. Since multiplication in LA is quick, it should not affect the computational speed too much. Box 9 also includes some constant computations. The above discussion suggests that there is no significant speed difference between the circle and the ellipse drawing algorithms, as long as one dot address computation is concerned.

Equation (3.4) can be expressed as

$$y=(b/a)\sqrt{a^2-x^2}. \tag{3.7}$$

If Eq. (3.7) is used, the algorithm should be changed accordingly but with no significant difference.

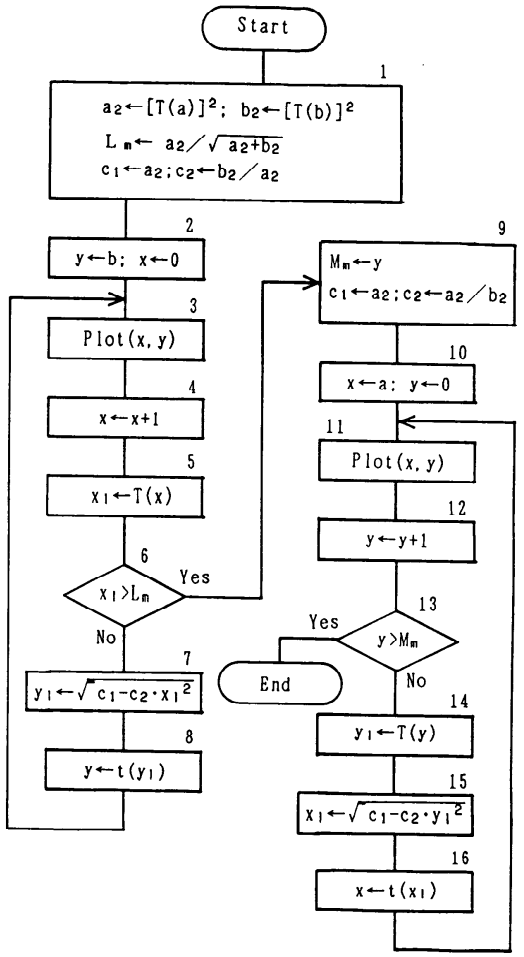

Fig. 4 Algorithm for Ellipse Drawing.

The LA method has some attractive features. For example, the coefficients of R, a, and b can be real numbers but with some sacrifice. Some changes should be made to enable the algorithm to handle such coefficients.

As for Fig. 2, "$y \leftarrow R$" and "$r_1 \leftarrow [T(R)]^2$" of box 1 should be changed so that R can be real. There is more than one way to do this.

One is for the case in which R is a general FP number. In this case y and r_1 may be given by "$y \leftarrow RD(R)$" and "$r_1 \leftarrow \log_a(R^2)$", respectively, where RD means a rounding function and $\log_a$ means a logarithmic function with base a. The two functions can be realized in software or hardware, possibly by using a general FP hardware processor. Even with hardware the speed should become slower as long as FP arithmetic is used.

Another way is for the case in which R is a fixed point number with some fraction below the binary point. Its form is similar to that of (2.1). In this case "$y \leftarrow R$" and

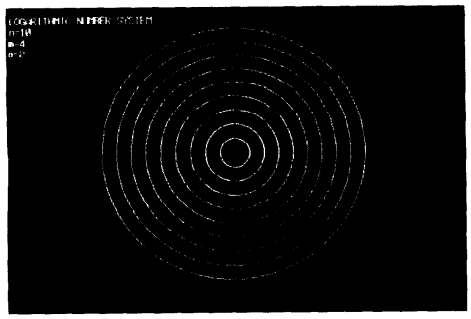

Fig. 5 Circles Generated by LNS.

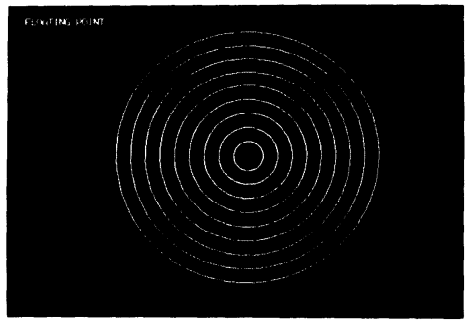

Fig. 7 Circles Generated by FP.

```
14: cir_ln(px,py,pr)
15: int px,py,pr;
16:  {
17:  int ex,ey,er,x,y;
18:  y=pr;
19:  er=cvtil[pr] << 1;
20:  for (x=0;x < y;){
21:       Plot(x,y);
22:       x++;
23:       ex=cvtil[x] << 1;
24:       ey=(er+mtable[er-ex]) >> 1;
25:       y=cvtli[ey];
26:       }
27:  if (x == y){
28:       Plot(x,y);
29:       }
30: }
```

Fig. 6 *C* Program for Generating Circles.

```
09: cir_fp(px,py,pr)
10: int px,py;
11: unsigned pr;
12:  {
13:  int x,y;
14:  y=pr;
15:  pr*=pr;
16:  for (x=0;x < y;){
17:       Plot(x,y);
18:       x++;
19:       y=(int)(sqrt(pr-x*x)+0.5);
20:       }
21:  if (x == y){
22:       Plot(x,y);
23:       }
24:  }
```

Fig. 8 *C* Program for Generating Circles by FP.

"$r_1 \leftarrow [T(R)]^2$" should become "$y \leftarrow RD(R)$" and "$r_1 \leftarrow [K(R)]^2$", respectively, where RD means a rounding function and K means a table for conversion to LNS. With a limited fraction in R, the above functions should not be a big draw-back in either table size or speed.

The same discussion can be applied to the real coefficients of a and b in ellipse drawing.

4. Experiments and Discussion

Some experiments were done on a 16-bit micro-computer. Programs were developed to draw circles and ellipses, and were tested for quality and speed.

4.1 Quality Test of Circles

A set of circles was drawn to see how it looked. The quality of the circles generated by the LNS method was evaluated by comparison with other methods of circle drawing.

Figure 5 shows the circles thus drawn by using LNS of $a=2$, $m=4$, and $n=10$, which can be realized in a 16-bit word system. Those circles have radii $R=180$, 160, 140, . . . , 20. They look very good. The specific program is shown in Fig. 6, which is written in *C* language. This program is easy to read, because it precisely follows the flow of Fig. 2. "Plot" is the procedure for plotting. All the lookup tables are pre-computed by rounding instead of truncating. The LA used in the program does not necessarily follow precisely the original procedure of LA [6]. For example, the sign of the logarithmic number s in (2.1), is omitted because clearly none of the values used in drawing a circle are negative. Incidentally, "*px*" and "*py*" are the center coordinates of the display, and "*pr*" is the radius R.

To find how good the circles are relative to these drawn by another method, we can draw supposedly identical circles by using the FP arithmetic function of an 80287 (FP, 64 bit). Figure 7 shows the result. The program, written in *C*, is shown in Fig. 8. The control flow is exactly the same as that of Fig. 6. The two sets of circles in Figs. 5 and 7 look indentical. There are, however, some differences. As long as FP is used, almost error-free computation is expected to generate the integer address coordinates. Every coordinate generated by LNS is compared with every one generated by FP. There are 634 dots generated in drawing all the nine octants of $R=180$, 160, . . . , 20. Out of those, 613 dots are exactly the same and 21 dots have a difference of 1 in the two sets of circles.

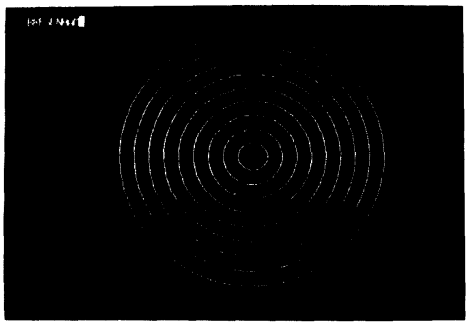

Fig. 9 Circles Generalted by BM.

```
87: cir_br(px,py,pr)
88: int px,py,pr;
89:  {
90:  int x,y,d;
91:  y=pr;
92:  d=3-(pr << 1);
93:  for (x=0;x < y;){
94:       Plot(x,y);
95:       if (d < 0)
96:            d+=(x << 2)+6;
97:       else{
98:            d+=((x-y) << 2)+10;
99:            y--;
00:            }
01:       x++;
02:       }
03: if (x == y){
04:       Plot(x,y);
05:       }
06:  }
```

Fig. 10 C Program for Generating Circles by BM.

Another quality comparison was made with the well-known Bresenham-Michener (BM) "incremental" method [5]. It was found that the circles generated by FP and BM had no differences insofar as the nine circles are concerned. The circles generated by the "incremental" method are shown in Fig. 9, and the program used is shown in Fig. 10.

4.2 The Relation between Types of LNS and the Circle Quality

In LA, conversion or computational errors are expected to be strongly related to the type of specific number system [8, 9, 13] especially n in (2.1). More experiments were done to find the relation between n and the error. Using the number systems ($a=2$, $m=4$, $n=4$ to 18), 186 octants with radii between $R=5$ and $R=190$ were computed. The number of disagreements with the results obtained by FP and the number of jumps (gaps on the curves) were counted. The results are shown in Table 1: "n" as in (2.1), "disag" is the number of disagreements; "jump" is the number of gaps; "N" means the total number of dots computed in each LNS. The table shows that for $n=17$ and above, the results of

Table 1 Relation Between Error and LNS Type (LNS: $a=2$, $m=4$, $n=4$ to 18; 186 Octants of Radius=5 to 190).

n	disag	jump	N
4	10559	1238	12840
5	8696	1733	12815
6	6259	1085	12797
7	3079	164	12781
8	1540	27	12776
9	773	8	12776
10	443	1	12776
11	205	0	12775
12	103	0	12777
13	55	0	12776
14	40	0	12776
15	12	0	12776
16	2	0	12776
17	0	0	12776
18	0	0	12776

Table 2 Comparison of Operational Constituents.

Operation	LNS	Bre-Mich
Inc/Dec	1	1.41
Compare	—	1
Shift	2	1
Add/Sub	2	2.41
Memory ref.	3	—

LNS and FP are in complete agreement and that LNS has no jumps for $n=11$ and above. It shows that as n increases, the quality of the computation also rapidly increases.

Like any other number system, a LNS also has its maximum and minimum (absolute value) expressible numbers. According to (2.1), the maximum is expressed as

$$a^{2^m-2^{-n}} \approx a^{2^m}. \tag{4.1}$$

Therefore, as long as R^2 is less than (4.1), a normal LNS computation is expected. But the accuracy should depend on n [8] with the base a fixed [9]. As for the LNS ($a=2$, $m=4$, $n=10$), the maximum is close to 2^{16}. Thus the radius R should be less than 256. If m is increased by one, that is, $m=5$, the maximum should be close to 2^{32}. Accordingly, the radius can be almost as large as 2^{16}. An LNS of $a=2$, $m=5$ and $n=15$, which can be realized by a 22-bit word system (often used for FP arithmetic as in DSP), was tested for computing coordinate addresses of a large octant of radius=2000. The disagreement count (with difference 1) with FP was only 24 out of 1414 dots and no jump occurred. The remaining dots were in complete agreement.

4.3 Speed Comparison

Let us do a kind of speed comparison—of the basic operations necessary to generate a dot—with the BM method. Table 2 shows what and how many such operations constitute each of the algorithms of LNS and BM, which is considered to be one of the fastest. The two

may be considered comparable. The memory reference of LNS, however, seems to be a burden. The data shown are derived from the programs of Figs. 6 and 10 especially inside the "for-loop." The procedure calls of "Plot" are ignored. Every possible effort is made to shorten the execution time of each program. Take some examples. In the BM algorithm, shift operations are used for multiplications by 2 and 4, and increment or decrement operations for addition or subtraction of 1, respectively. The fractional data of BM in the table comes from a simple geometrical fact. That is, "$(d<0)$" of Fig. 10 is true ($\sqrt{2}-1$) times and false ($2-\sqrt{2}$) times for each dot generation.

We also did some actual experimental speed comparisons with the two methods of FP and BM, using a micro-computer. The test conditions were as follows:

1) The three source programs shown in Figs. 6, 8, and 10 were used, but without the procedure calls of "Plot." The reason for omitting the calls is that the programs are to compare the address computation time, not the display time. Incidentally, the procedure calls of the plotting should actually take a considerable part of the execution time. An FP program is also made to be as fast as possible. FP arithmetic is used only for the procedure call of "sqrt" and the addition of 0.5 for rounding. Other wise, integer arithmetic is used.

2) One of the most popular personal computers with the micro-processor (an 80286 of 8 MHz) and its FP co-processor (an 80287 of 10 MHz) was used.

3) A microsoft *C* compiler was used with the options of the optimizer ($0x$) and the co-processor options (FPi87).

4) The time taken to compute the dot addresses of 10,000 octants of radius $R=190$ was measured for each of the programs, thus calling each program 10,000 times.

The results are shown on the "*C*" row of the "1/8 Circle" in Table 3. "Bre-Mich" means the BM method; "LNS_1" means the LNS method. The fastest was BM, which took only 9 seconds. The next fastest was LNS, followed by FP, which took 3 minutes and 14 seconds. LNS, though it was only the second fastest, is considered to be very fast, because there are three memory reference operations in the LNS program, which usually take several times longer to execute than operations such as "shift" or "increment." The sentences of "shift" or "increment" are considered to be register operations in machine instructions. In fact, this was confirmed by the assembly program list, and they were executed very quickly. The memory references are those of array operations of "cvtil[x]", "mtable[er-ex]", and "cvtli[ey]" inside the "for-loop" of the LNS program. The fact that the execution time of memory reference operations was longer than that of register operations was probably because the actual physical memory exists outside the CPU chip. In the micro-computer used for this experiment, the memory is on different printed boards.

```
14: cir_ln(px,py,pr)
15: int px,py,pr;
16:  {
17:  int ex,ey,er,x,y;
18:  y=pr;
19:  er=cvtil[pr] << 1;
20:  for (x=0;x < 134;){
21:       Plot(x,y);
22:       x++;
23:       ex=cvtil[x] << 1;
24:       ey=(er+x+ex) >> 1;
25:       y=x;
26:       }
27:  if (x == y){
28:       Plot(x,y);
29:       }
30: }
```

Fig. 11 Program in which Memory Reference Is Replaced by Register Operation.

Table 3 Speed Comparison.

		Bre-Mich	FP	LNS_1	LNS_2
1/8	*C*	9	194	15	8
Circle	*A*	7	87	11	8
1/4	*C*	—	469	34	—
Ellipse	*A*	—	246	26	—

If the execution time of the memory reference is shortened by several hundred percent, then the execution of the LNS program is expected to be much quicker. We did some experiments to measure the excution time; the program in Fig. 11 was written for that purpose. Two of the three array references are replaced by "supposed" register operations, so that the memory references become quicker. These new operations are on lines 24 and 25. The resulting program is nonsense, but is intended to measure the excution time. The measured execution times are shown in the column of Table 3 labeled "LNS_2". The data says that the LNS program has become as fast as the BM program. Row "*A*" in the table shows the execution time of the assembler programs written by the authors. Since they are lengthy, and are therefore not shown, the data is intended only for reference. Incidentally, "*C*" in the table stands for the *C* compiler with the optimizer.

The above operation replacements, though indirect and not rigid, imply a program execution situation closer to one in which the operations of the array lookup are realized within a processor's chip hardware. The LNS processor [12] has this kind of on-chip memory and is capable of tremendous speed computations. Many DSP chips have such a memory on the chip (though not a very large one) and an access time as fast as register operations [14].

RISC is another kind of computer, based on what may be an opposite philosophy. It is probably true that the LNS circle-drawing algorithm is much slower than the "incremental" one if run on a RISC. RISC is, in a

sense, considered to be a computer in which a small amount of memory (number of registers) is made very fast and a large amount of memory left comparatively slow.

However, drawing curves by using an LNS should be rather compared with doing so by using FP arithmetic, since it is based on the computational method.

4.4 Lookup Table Size

The size of the lookup table of LA could become impossibly large. With no reduction [15, 16], the addition and subtraction tables have a total of 2^{m+n+2} entries. Incidentally, the addition table is not used for drawing circles. In addition to the above tables, we require two conversion tables for converting integers to LNS and vice versa. The first table contains N entries if the integers are from 1 to N. It should normally be small. The second table may have as many as 2^{m+n+1} entries if there are no reductions. With an LNS ($a=2$, $m=4$, $n=10$), when the input integers are given in 8-bit code, $2^{16}+2^{15}+2^{8}$ (16-bit) words are necessary. This is not too big. For a 22-bit word LNS ($a=2$, $m=5$, $n=15$), the add/sub tables combined should have 2^{22} entries, and the table for converting LNS to integers should have 2^{21} entries, resulting in about 8 MB.

However, studies [15, 16] have been made of ways to reduce the size of the add/sub tables. One method [16] compresses the add/sub tables by using the fact that the table functions:

$$F(N)=\log_2(1+2^{-N}) \quad \text{and} \tag{4.2}$$

$$G(N)=\log_2(1-2^{-N}), \quad (N\geq 0) \tag{4.3}$$

have essential zeros for N above a certain value. The zero portions occupy the majority of the two tables. The percentage of such parts depends on the base a and n. Accordingly, a simple test in $F(N)$ and $G(N)$ lookup would discard such portions of the tables. For an LNS in which $a=2$, $m=5$, and $n=15$, eliminating the zero part reduces the number of entries to 25.8 percent of the original table. Further reduction is possible if portions of one, two, and so on are discarded [16]. Another method allows additional reduction [15].

The table for LNS to integer conversion can also be reduced. For drawing circles, y in Eq. (3.2) is bounded as

$$R/\sqrt{2}\leq y\leq R. \tag{4.4}$$

It is therefore sufficient to prepare a table of 2^{n-1} entries. However, this is just for circle drawing and not for general use. If the lower and upper bounds of the resultant LA value (before LNS-to-integer conversion) are known as is generally the case, the required table size can be determined. Then, letting x be the LNS exponent, the value 2^x is bounded as

$$t\leq 2^x\leq u. \tag{4.5}$$

Then, if t is zero, the number of entries in the table becomes

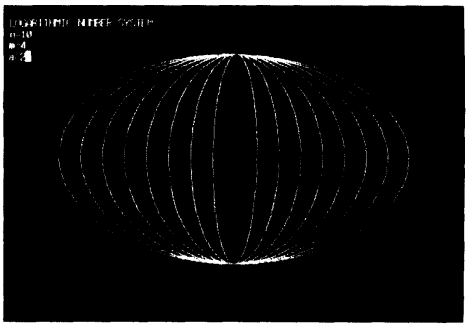

Fig. 12 Ellipses Generated by LNS.

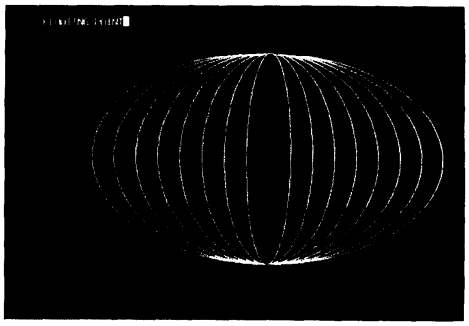

Fig. 13 Ellipses Generated by FP.

$$2^{n+m}+\mathrm{INT}(2^n\log_2 u)+1, \tag{4.6}$$

where INT means truncation at the binary point. If the zero part is to be eliminated, then

$$1/2\leq 2^x\leq u. \tag{4.7}$$

Accordingly, the size becomes

$$\mathrm{INT}(2^n\log_2 u)+2^n+1. \tag{4.8}$$

If the above is applied to an LNS in which $a=2$, $m=5$ and $n=15$, only about 17% of the original table size of 2^{21} is required.

Obviously the table for integer-to-LNS conversion is not usually large in comparison with the above two tables. As a result, the overall storage requirement could be easily reduced by 75% or more with a little loss in computational speed.

4.5 Ellipse Drawing

Some ellipses were also drawn by using an LNS of $a=2$, $m=4$, $n=10$, as shown in Fig. 12. The algorithm shown in Fig. 4 was applied. The resulting ellipses look good. For comparison, the supposedly equivalent ellipses were also drawn by using FP, as shown in Fig. 13. Nothing looks different.

Table 3 gives some data for comparing the speed of the two methods in drawing ellipses. The same conditions are applied for measuring the speed as in circle drawing. Dots of 10,000 quarter (not one-eighth for circles) ellipses of $a=b=190$ (actually the circle drawn

previously) are computed. The measured data suggest that dots can be generated for ellipses almost as fast as for circles. It took 15 seconds to generate 1/8 of a circle and 34 seconds for 1/4 of an ellipse. There should be only two second difference for 1/8 of a circle and 1/8 of an ellipse. This should be due to the additional multiplication of (b^2/a^2) and x^2 of Eq. (3.4), as previously explained.

5. Concluding Comments

A new method has been presented for drawing curves by using logarithmic number systems. It was shown that high-quality curves can be drawn by experimentally drawing circles and ellipses. It was also shown that this method is comparable in speed to one of the fastest "incremental" methods.

Since the curves are drawn by computational means called LA and data conversion, the method is very general and easy to apply to many types of curves. That is, as long as a curve is expressed as a function, such as $y=g(x)$, that consists of addition/subtraction, multiplication/division, and power computation, the computation of y (dot address generation) is always possible and can be donc very quickly. The method has many attractive features:

1) **Parallel computations**: Since every dot generation is independent, parallel computation is possible. It is conceptually possible to generate more than one dot simultaneously in parallel, using multiple LA processors. In the "incremental"-type algorithm, every dot is dependent on the previously generated dot, and therefore, parallel processing is not directly possible.

2) **Flexible starting point**: Any dot can be a starting point for drawing. That is, for $y=g(x)$, x can have any value at the start of drawing. The "incremental" algorithm usually assumes a special point, that is, an exact mesh point. In many cases it must be a very special point.

3) **Real number coefficients**: In LA, computations are naturally in real numbers. The coefficients of curve equations need not be integers. For circles and ellipses, the coefficients of a and b may be real numbers. This feature provides flexibility in that it can easily cope with many ways of defining curves.

4) **High precision**: LNS computations are far more accurate than FP computations with the same number of bits for a word and equivalent dynamic ranges [8, 9]. One reason for this is that there are no computational errors due to multiplications and divisions in LA. Thus there are no errors in squares. In FP, such computations generally cause errors.

In computations of addition and subtraction, errors exist for both FP and LA. But the error range in LA is smaller. Suppose FP numbers are expressed with $n+m+2$ bits, that is, an n-bit fraction, an $(m+1)$-bit exponent, and a 1-bit sign (compare with (2.1)). The dynamic range (the largest positive number divided by the smallest positive number) for LA of Eq. (2.1) is expressed as

$$r_l=2^{(2^{m+1}-2^{-n})}. \tag{5.1}$$

That for FP is

$$r_f=(1-2^{-n})2^{2^{m+1}}. \tag{5.2}$$

These are essentially equal, but that of LNS is a little larger. The relative error e (=the absolute error divided by the error-free result) range for LA addition and subtraction is expressed as

$$2^{-2^{-n-1}}-1\le e_l\le 2^{2^{-n-1}}-1. \tag{5.3}$$

The error range of the FP addition/subtraction, multiplication/division and square/square root is

$$-2^{-n}\le e_f\le 2^{-n}. \tag{5.4}$$

The error range of the FP is about 2.89 times larger [8, 9].

In an LNS, numbers are arranged in such a way that the difference between two adjacent numbers is proportional to their magnitude. In FP, however, the difference is usually equal, but it is doubled each time the exponent is incremented by one, and FP requires normalization at the end of each computation. This means that there are some redundancies in the number coding. The number arrangement of an LNS is more smooth and rational.

These are the reasons why LA computation is more precise than that of FP. Generation of dots by the described method is still more accurate even with number conversions. This accuracy holds for any LA computation. Therefore, it should be possible to draw curves other than circles or ellipses precisely as well.

5) **High-speed Computation**: The Computation speed is much higher than that of FP with equivalent conditions [12], even with number conversions. The speed of the LA method in drawing circles compared with that of the "incremental" method should be noted. With so many outstanding merits, the speed of the LA method is still comparable with that of the fastest existing "incremental" altorithm.

6) **Generality**: The method of the curve dot address generation is very general. Since LA is a valid computation, there is no reason that the computation could not be applied to drawing curves other than circles and ellipses. As long as a curve is defined as a function form, $y=g(x)$, and g is computable in the LNS, it is always possible to generate the curve (as a matter of fact, it need not be a curve). Other applications have also been reported [7–19].

7) **Easy usage**: Assuming the existence of an LA processor (software or hardware), the curve dot generation software can be easily developed without a deep knowledge of geometry or the algorithm.

In summary, the proposed method is like using FP arithmic but fast. The one problem is memory space. The greater the accuracy desired, the more memory is re-

quired—the memory space is doubled by each 1-bit increase in the word length. A word length of, say, 64 bits, which is normally implemented in FP arithmetic, requires an impossibly large memory for current technology. Even for a 32-bit word length it would be very hard to use.

5.1 Miscellaneous Discussions

1) **Problem 1: Draw an nth-order polynomial $y=f(x)$:**

$$y=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0. \tag{5.5}$$

In DDA, it is possible to generate dot addresses by successive additions of differences. The number of additions needed to generate one dot is

$$\sum_{i=1}^{n} i=n(n+1)/2. \tag{5.6}$$

For example, take

$$y=a_3x^3+a_2x^2+a_1x+a_0.$$

The following equations show how a dot is computed:

$$\begin{aligned} y(x)&=y(x-1)+y'(x-1)+y''(x-1)/2+a_3,\\ y'(x)&=y'(x-1)+y''(x-1)+3a_3,\\ y''(x)&=y''(x-1)+6a_3. \end{aligned} \tag{5.7}$$

The number of the additions is six for one dot. The LA method requires essentially:

$3n$ (fixed-point) additions,
n memory references, and
2 conversions (2 memory references),

if the following order of computations is used:

$$(\cdots((a_nx+a_{n-1})x+a_{n-2})x+\cdots+a_1)x+a_0. \tag{5.8}$$

For small n, the DDA method requires smaller numbers of additions and should be faster. For larger n, the LA method should be faster. If, however, the polynomial is given in fewer terms, the LA method becomes faster. As for $y=x^3$, one shift, one addition, and two number conversions can generate one dot address for the LA method. For DDA, the computations of Eq. (5.7) are still required. LA should therefore be much faster than DDA in this case.

Problem 2: Is the LA method superior to the existing number system method using square and square root tables? First, the FP number system should be considered. For FP, the speed of square and square root operations definitely becomes fast with lookup tables. Circle drawing (in which the required operations are square, square root, and addition) should be quick, but only for those two operations. Multiplication, division, and other arithmetic operations will not become faster. Therefore it will not usually work effectively except for circle drawing. In addition, as mentioned previously among the features of the LA method, the computational error becomes larger for FP.

Next, consider a fixed-point number system. In this case, in addition to the multiplication/division problem, the dynamic range of the number system causes a problem. For simplicity, consider an fixed-point (integer) number system with n-bit words and positive numbers only. The largest number is 2^n-1, and its square root approximately $2^{n/2}$. Thus the largest radius is $2^{n/2}$ for circle drawing. For $n=18$ the radius should be less than 512. This is not large in relation to the word length. In general, the intermediate result often becomes very large or very small (in absolute value) and fixed-point arithmetic cannot handle such situations easily. Incidentally, multiplication and division are also slow for integers.

References

1. DANIELSSON, P. E. Incremental Curve Generation, *IEEE Trans. Comput.*, **19**, 9 (Sep. 1970), 783-793.
2. JORDAN, B. W., Jr., LENNON, W. J. and HOLM, B. D. An Improved Algorithm for the Generation of Nonparametric Curves, *IEEE Trans. Comput.*, **22**, 12 (Dec. 1973), 1052-1060.
3. BRESENHAM, J. A Linear Algorithm for Incremental Digital Display of Circular Arcs, *Comm. ACM*, **20**, 2 (Feb. 1977), 100-106.
4. SUENAGA, Y., KAMAE, T. and KOBAYASHI, T. A High-Speed Algorithm for the Generation of Straight Lines and Circular Arcs, *IEEE Trans. Comput.*, **28**, 10 (Oct. 1979), 728-736.
5. FOLEY, J. D. and VAN DAM, A. Fundamentals of Interactive Computer Graphics, Addison-Wesley (1982), 441-445.
6. KINGSBURY, N. G. and RAYNER, P. J. W. Digital Filtering Using Logarithmic Arithmetic, Electron. Lett., **7**, 2 (Jan. 1971), 56-58.
7. LEE, S. C. and EDGAR, A. D. The Focus Number System, *IEEE Trans. Comput.*, **26** (Nov. 1977), 1167-1170.
8. KUROKAWA, T. Error Analysis of Digital Filters with Logarithmic Number System, Ph.D. Dissertation, Univ. of Oklahoma, Norman (1978).
9. KUROKAWA, T., PAYNE, J. A. and LEE, S. C. Error Analysis of Recursive Digital Filters Implemented with Logarithmic Number Systems, *IEEE Trans. Acoust., Speech, Signal Processing*, **28**, 6 (Dec. 1980), 706-715.
10. SICURANZA, G. L. On Efficient Implementations of 2-D Digital Filters Using Logarithmic Number Systems, *IEEE Trans. Acoust., Speech, Signal Processing*, **31**, 4 (Aug. 1983), 877-885.
11. LAMAIRE, R. O. and LANG, J. H. Performance of Digital Linear Regulators Which Use Logarithmic Arithmetic, *IEEE Trans. Automat. Contr.*, **31**, 5 (May 1986), 394-400.
12. TAYLOR, F. J., GILL, R., JOSEPH, J. and RADKE, J. A 20-Bit Logarithmic Number System Processor, *IEEE Trans. Comput.*, **37**, 2 (Feb. 1988), 190-200.
13. KUROKAWA, T. Errors in Conversion from Integers to Logarithmic Number Systems (in Japanese), Denki Kankei Gakkai Tokaishibu Rengotaikai (Oct. 1989), 619.
14. Texas Instruments, TMS320C25 Digital Signal Processor Users' Manual, Texas Instruments (1988).
15. FREY, M. L. and TAYLOR, F. J. A Table Reduction Technique for Logarithmically Architected Digital Filters, *IEEE Trans. Acoust., Speech, Signal Processing*, **33**, 3 (June 1985), 718-719.
16. EDGAR, A. D. and LEE, S. C. Focus Microcomputer Number System, *Comm. ACM*, **22**, 3 (Mar. 1979), 166-177.
17. KUROKAWA, T. and MIZUKOSHI, T. A Fast Transformation of Pictures with Interpolations—An Application of Logarithmic Number Systems (in Japanese), *Proceedings Graphics and CAD Symposium of IPS Japan* (Nov. 1989), 217-226.
18. KUROKAWA, T. and MIZUKOSHI, T. Fast Curve Drawing Using Logarithmic Number Systems, (in Japanese), *Proceeding 20th Joint Conf. Imaging Tech. Japan* (Dec. 1989), 47-50.
19. KUROKAWA, T. An Effective Way to Use Logarithmic Arithmetic, Particularly for Signal Processing (in Jaspanese), Autumn National Convention Records of IEIEC of Japan, 1 (Sep. 1989), 133.

(Received August 31, 1989; revised July 6, 1990)