DRAGON

BY BRUCE R. LAND

Recursive drawing of a dragon curve on the Macintosh

THIS PROGRAM uses the high-resolution Macintosh display and the rapid execution rate of MacFORTH (Creative Solutions Inc.) to produce a recursive drawing of the so-called Harter-Hightway Dragon (described in reference 1). The resulting curve (see figure 1) is interesting to look at because it has features on many different length scales, from the very smallest kinks to the major "body" segments.

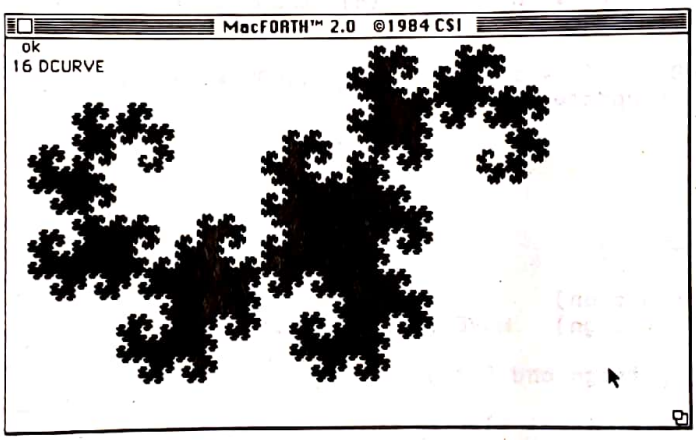

Figure 1: A *16th-order dragon drawn by the* FORTH *program* DRAGON.

The curve is constructed by successive fragmentation of line segments into ever-smaller right angles. This construction lends itself to implementation by recursion.

Recursion is a process by which a task is broken up into smaller tasks that are similar except for, perhaps, a count of how many times the smaller task has been performed. The unique feature of recursion is that a recursive procedure calls *itself* before it finishes execution, so that intermediate results pile up, to be resolved later when an end condition (for instance a count) indicates that the recursion is complete.

To perform recursion in FORTH you must consider two things. The first is that all local variables of the recursive routine must be on the stack, because when a routine calls itself, FORTH (unlike Pascal) does not make copies of variables for each invocation. The second is that FORTH normally assumes that no one would ever want to invoke a verb whose definition is not complete, so a recursive call within a definition is considered an error. This problem is easily handled by defining a verb called RECURS (reference 2) in listing 1 that temporarily tricks the compiler into considering the definition complete. As you can note in listing 1, this verb must always be used in pairs or strange things will happen.

FORTH is a bottom-up programming language in which you start with the primitive operations and build up complex ones. After defining the RECURS verb, the program defines four variables to locate a "drawing turtle." I used a turtle-graphic approach because the code was translated from Apple UCSD Pascal (reference 3). MacFORTH does not have turtle operations, but it is easy to produce the turtle operations TURN and MOVE. TURN accepts an angle (in degrees) on the stack and updates the turtle direction. MOVE causes the turtle to take one step forward in the direction it is facing.

The main work is done by DRAGON, which is the recursively called routine. Each time you enter DRAGON it calls itself twice until a value on the stack, called "level" in

(continued)

Bruce R. Land, Ph.D. (Box 73, RD #1, Trumansburg, NY 14886), is a senior lecturer at Cornell University.

Listing 1: *The* FORTH *blocks for the* DRAGON *program.*

```
Screen #1
( begin  Dragon curve )
CREATE CURVE    ( a FORGETable name)

CARTESIAN OFF

: RECURS  SMUDGE ; IMMEDIATE  ( trick verb for recursion)

VARIABLE ANGLE
VARIABLE XCOOD
VARIABLE YCOOD
VARIABLE STEPSIZE

: TURN (  deltangle-- | turn sign*delta)
  ANGLE +! ;

2 4 THRU


Screen #2
: MOVE ( -- | takes a step in present turtle direction)
  STEPSIZE @  DUP
  ANGLE @  COS *  10000 /  ( r* cos  of theta)  XCOOD @  +
  DUP ( newX)  XCOOD ! ( update X)
  SWAP
  ANGLE @  SIN *  10000 /   ( r* sine theta )   YCOOD @  +
  DUP ( newY)  YCOOD ! ( update Y)
  DRAW.TO    ;


Screen #3
: DRAGON ( sign level-- |  )
  DUP ( level) 0=
  IF  ( at bottom of recursion)
    DROP ( level)  DROP ( sign)   MOVE ( by stepsize)
  ELSE
    OVER  45 * TURN  ( getsign and turn)
    1 ( newsign)
    OVER   1-       ( level=level-1)
    RECURS DRAGON  RECURS

    OVER  -90 * TURN ( getsign & turn)
    -1 ( newsign)  ( edit to +1 for diff curve)
    OVER  1-           ( level=level-1)
    RECURS DRAGON RECURS
    DROP ( input level)  45 * TURN  ( getsign and turn)
  THEN  ;


Screen #4
: DCURVE ( level  --|  )
  ( init pen position)
  PAGE   100 XCOOD !    90 YCOOD !   360 6 *  ANGLE !
  WHITE PENPAT  XCOOD @  YCOOD @  MOVE.TO

  PEN.NORMAL
  1 STEPSIZE !
  1 SWAP ( level) DRAGON

  WHITE PENPAT   4 10 MOVE.TO    PEN.NORMAL ;
```

With this MacFORTH program, you can draw a 16th-order dragon curve in 4½ minutes.

the commands, reaches zero. At that point a step is drawn. A 16th-level dragon curve (the biggest that will fit on the screen) results in 2^{16} separate calls to DRAGON.

The final routine DCURVE initializes the pen position, sets up the stack for DRAGON, invokes DRAGON, and finally moves the pen away from the end of the curve. The program will draw a 16th-order dragon curve in 4½ minutes.

The program has a few limitations. The MacFORTH sine and cosine routines carry only four-place accuracy, so the program runs correctly only for even orders. An odd order causes the turtle to wander off the screen.

To run DRAGON, type N DCURVE, where N is the order desired. At one place in the program listing you'll see a notation that editing will produce an interesting variant. In particular, note that a 16th-order curve will not fit on the screen unless the STEPSIZE constant is unity. ■

[*Editor's note: The source code for* DRAGON *is available for downloading from* BYTE*net Listings. The number is* (617) 861-9764. *It is also available on disk. See page* 346 *for details. You will need* MacFORTH *to run the program.*]

REFERENCES

1. Mandelbrot, B. B. *The Fractal Geometry of Nature*. San Francisco: Freeman, 1982, page 66ff.
2. Smith, A. J. "Another Recursion." *FORTH Dimensions*, vol. 3, no. 6, 1982, page 179.
3. Luehrmann, A., and H. Peckham. *Apple Pascal: A Hands-on Approach*. New York: McGraw-Hill, 1981, pages 335–344.