COMPUTER RECREATIONS

Wallpaper for the mind: computer images that are almost, but not quite, repetitive

by A. K. Dewdney

Ordinary wallpaper is printed by a rotating cylinder engraved with a design. As the cylinder turns it prints the same design over and over again. Only a computer, however, can reproduce certain richly embroidered patterns I call wallpaper for the mind. These patterns do not repeat themselves, at least not exactly; instead each pattern continually manifests itself in new contexts and configurations, left and right, up and down. From one apparition to the next, what is changed and what is preserved?

The color swatches in my current sample book demonstrate the results of three widely differing techniques. The computer programs responsible for the images range in difficulty from the extremely simple to the merely easy. They come from three readers: John E. Connett of the University of Minnesota, Barry Martin of Birmingham, England, and Tony D. Smith of Essendon, Australia.

Connett's program is based on the circle, but it celebrates the varieties of design based on the square. The apparent conundrum compels me to name it CIRCLE^2. In a nutshell, it applies the formula for the graph of a circle, $x^2 + y^2$, to assign a color to the point that has the coordinates x and y. I shall give the details below. In the meantime you may be boggled, as I was, to discover that there is much more to this wallpaper than a set of concentric circles; specifically, if you back away from the wall, intricate patterns of delicate squares may also emerge [*see illustrations on opposite page*]. There are mysteries here.

It is perhaps not surprising that CIRCLE^2 was inspired by the Mandelbrot set, which I described here in August, 1985. The set is the discovery of Benoit B. Mandelbrot of the IBM Thomas J. Watson Research Center; the riot of form and color that surrounds the Mandelbrot set is based on a single mathematical function, applied repeatedly to its own output for each complex number in a region of the plane. Whenever the iterated value of the function reaches the magnitude 2, the number of iterations needed to reach that magnitude determines the color of the corresponding point.

Connett, who had no color monitor at his disposal, assigned black to points that reach 2 after an even number of iterations and white to points that reach 2 after an odd number of iterations. Creditable images of the Mandelbrot set appeared, but Connett was led to explore other formulas. He selected the formula $x^2 + y^2$ and discarded iteration altogether. His program systematically scans a gridded section of the plane; at each point (x,y) it computes the formula and then truncates the resulting value to an integer. If the integer is even, the point (x,y) is colored black; if it is odd, the point is colored white (left blank).

I fear I have just lost half of my audience. They already understand the program, and they have rushed off to type it into a nearby computer. It is that simple. In algorithmic shorthand CIRCLE^2 is an input section followed by a double loop:

```
input corna, cornb
input side

for i ← 1 to 100
    for j ← 1 to 100
        x ← corna + (side × i/100)
        y ← cornb + (side × j/100)
        z ← x² + y²
        c ← int(z)
        if c is even, then plot (i,j)
```

First the program causes the computer to call for the coordinates (*corna,cornb*) for the lower left corner of the square to be examined. The variable *side* is the length of the side of the square of interest. For example, if the user types -15 and -20 for the corner coordinates and 87 for the side, the program plots a 100-by-100 array of points within a square region of the plane 87 units on its side, whose lower left corner is the point $(-15,-20)$. In my outline of the program I have assumed the iteration limits run from 1 to 100, but they must be adjusted so that the square lies within the boundaries of the output device to be used. On my monitor these limits outline a smallish square on the screen.

The double loop marches through the square grid and for each index pair (i,j) computes the coordinates of the point (x,y) to which the pair corresponds. The loop then squares x and y, assigns the sum of the two squares to the variable z and truncates the sum. The largest integer less than or equal to the sum is stored as the variable c. If c is divisible by 2, the point (x,y) is plotted, presumably as a colored pixel on a monitor or as a black dot in printed output. If c is odd, no point is plotted.

Readers who want to re-create Connett's wallpaper need not get too anxious about whether they have the colors right. Most patterns are equally striking with the colors reversed. Indeed, even more than two colors are possible: instead of determining whether c is even or odd, divide it by the number of colors you want. The remainder after the division can then be assigned to a distinct color. For example, two, three and four colors were chosen to generate respectively the top, middle and bottom images in the illustration on the opposite page.

The smaller the square under examination is, the closer the plane appears to the viewer and the greater the magnification of CIRCLE^2's image is. Unlike the procedure for coloring neighborhoods of the Mandelbrot set, however, Connett's program does not yield an infinite regress of progressively smaller patterns. At high magnifications about the origin (0,0) a set of concentric circles appears. At still higher magnifications there is a large black disk in the middle of the screen: the truncated-integer value of every point in the disk is zero. Then all the screen is blackness.

You can better appreciate the beauty of Connett's wallpaper by reducing the magnification: back away from the wall. The concentric circles dissolve into an intriguing arrangement of primary and secondary circles resembling a moiré pattern. New wallpaper designs appear like magic, seemingly different with every lower magnification. Is there an infinite regress lurking here? It seems a vexing question, but I am confident readers will be able to shed some light on it before I publish the most interesting responses three months from now.

At Aston University in Birmingham, Barry Martin was also inspired by the Mandelbrot set. Martin adopted Mandelbrot's idea of iterating a formula from a numerical seed, but it is there that the similarity ends. Whereas Mandelbrot's patterns emerge from complex numbers, Martin's wallpaper is based on iterations of ordinary real numbers. Moreover, the numerical seeds for the Mandelbrot set are the points, infinite in number, found throughout a region of the plane; Martin's program grows its patterns from only one seed.

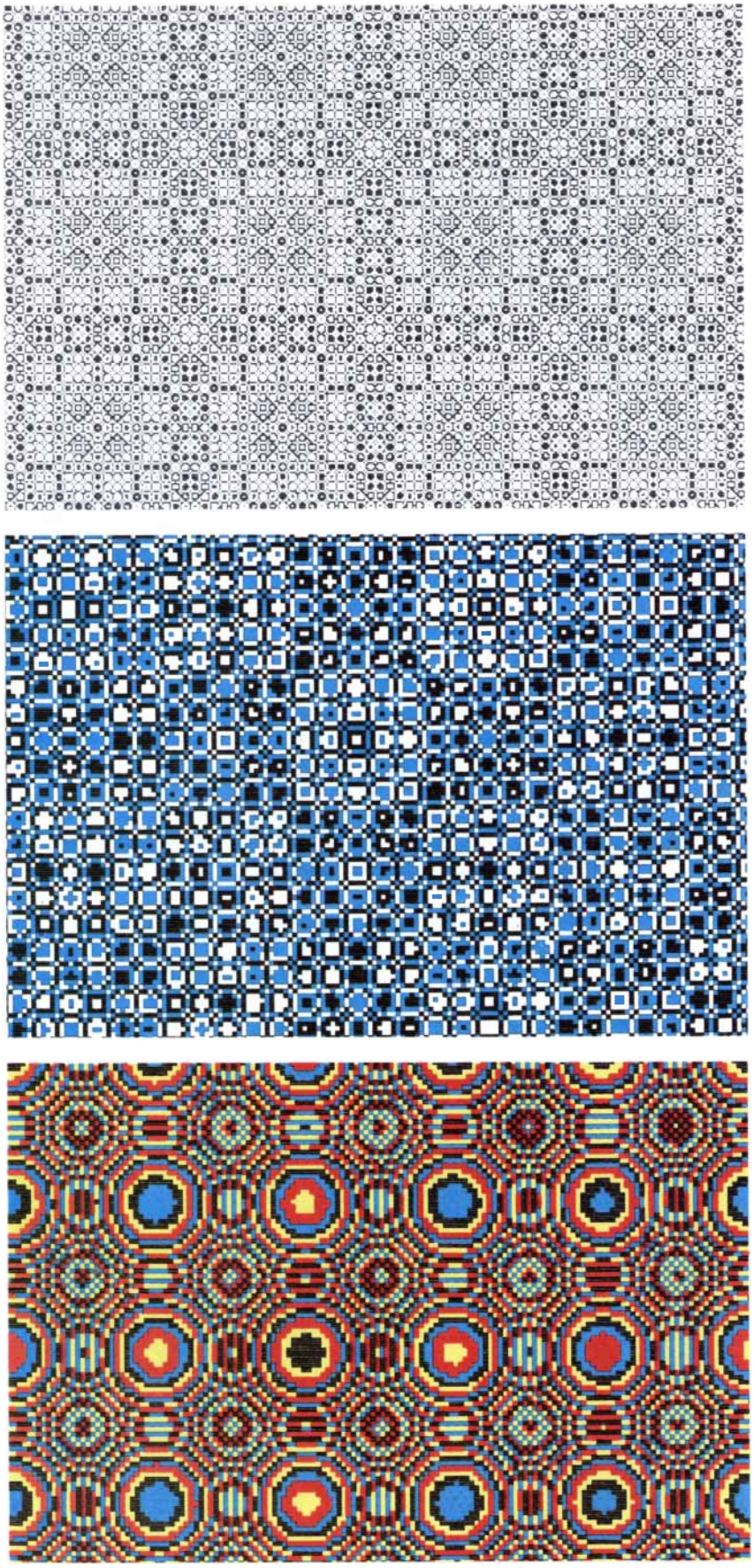
For example, Martin suggests trying the following pair of formulas, which can generate stunning, highly detailed images in many colors [*see the cover of this issue and the illustrations on the next page and pages 17 and 19*].

$$x \leftarrow y - \text{sign}(x) \times [\text{abs}(b \times x - c)]^{1/2}$$
$$y \leftarrow a - x$$

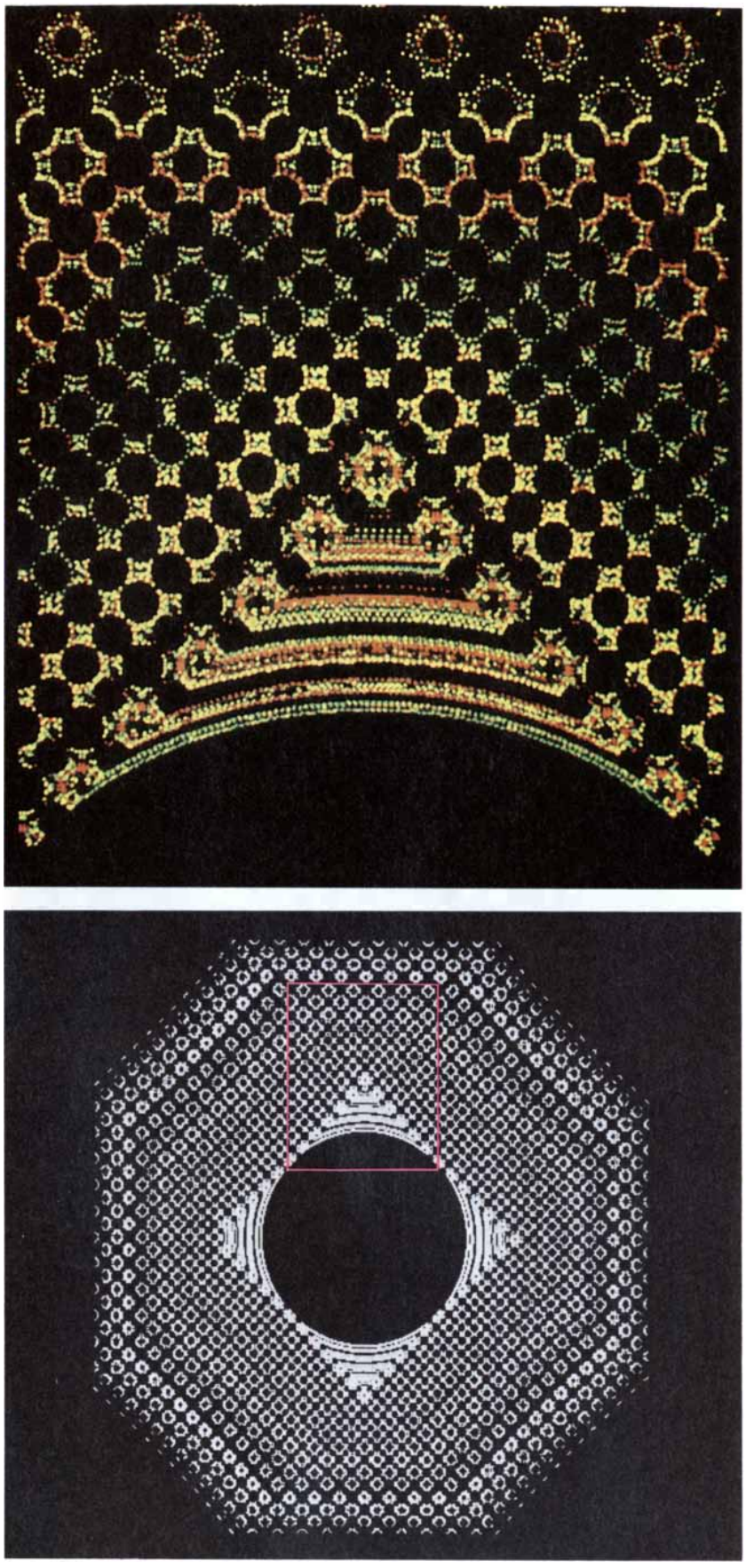
Here the function sign(x) takes the value 1 or -1, depending on whether x is positive or negative; the function abs($b \times x - c$) is the absolute value of the expression $b \times x - c$. The patterns can vary widely according to the values assigned to the letters a, b and c, which are numerical constants in the formula.

The formulas themselves are written in a form of mathematical shorthand: it is understood that one set of values is used for x and y in the formulas to the right of the arrows and a new set of values is computed from them on the left. The new values for x and y then replace the old values to the right of the arrows, and the calculation is repeated. In this way the program I call HOPALONG hops from one point to another. It begins at the point for which x and y are both equal to 0, namely at the origin. The next point might be at the upper right, the one after that at the lower left. A computer draws the points so quickly that one has the impression miniature electronic rain is falling on the screen: hundreds and then thousands of points drop onto the monitor. Soon a pattern begins to emerge. For example, if a is set equal to -200, b to .1 and c to -80, a broadly octagonal pattern is formed [*see bottom of illustration on next page*]. If the pattern is magnified and each point is colored according to the number of hops needed to reach it, the pattern becomes a wonderful cartouche [*see top of illustration on next page*]. For other values of a, b and c new designs appear: try setting a to .4, b to 1 and c to 0 [*see top illustration on page 17*], or set a to -3.14, b to .3 and c to .3 [*see bottom illustration on page 17*].

The algorithm for HOPALONG is al-



Circles and squares modulo 2, 3 and 4, from John E. Connett's program CIRCLE²



"Fractal" wallpaper from Barry Martin's program HOPALONG

most as easy to appreciate as the one for CIRCLE^2:

```
input num
input a, b, c

x←0
y←0
for i←1 to num
  plot (x,y)
  xx←y − sign(x) × [abs(b × x − c)]^1/2
  yy←a − x
  x←xx
  y←yy
```

Another stampede of readers rushes off to type in the program. Your rewards for lingering here are a more detailed explanation of Martin's program and a description of the third kind of wallpaper program.

In order to run HOPALONG one enters the total number of iterations as the variable *num;* one also enters values for *a, b* and *c.* The larger the value of *num,* the finer the detail of the pattern. For example, if *num* is 10,000, the program will plot 10,000 points on the screen, but for some values of *a, b* and *c* that is only the beginning. If *a* is −1,000, *b* is .1 and *c* is −10, the pattern at low magnification resembles the rind of a four-lobed lemon [*see bottom of illustration on page 19*]. When the run of the program is extended from 10,000 points to 100,000 and then to 600,000, the filigree becomes increasingly ornate [*see top of illustration on page 19*].

The algorithm can work as it stands, but it can also be enhanced; for example, one might add a facility for moving off-screen points or compressing off-screen regions into the visible region. If such features are added, three more parameters for determining position and scale must be specified at the start of the program. The body of the main loop must then be modified: just after *x* and *y* are computed the enhanced version of HOPALONG adds the position changes to *x* and *y* and multiplies the result by the scale factor.

The wallpaper analogy was not lost on Martin: "It seems to me we are on the verge of a pattern-generating explosion that has great commercial implications, e.g., we can expect to see 'designer' wallpaper and textiles within the next few years. Patterns will be produced by the customer merely by the selection of a few numbers." Martin is equally sanguine about the implications for mathematical biology. Look once again at the four-lobed lemon. The enlargements show details strongly reminiscent of vascular bundles: could it be the outer rind of a monocotyledon in cross section? About these and other patterns Martin

writes: "Clearly these curious configurations show us that the rules responsible for the construction of elaborate living tissue structures could be absurdly simple."

Readers might enjoy exploring the patterns generated by a different pair of iteration formulas, also suggested by Martin:

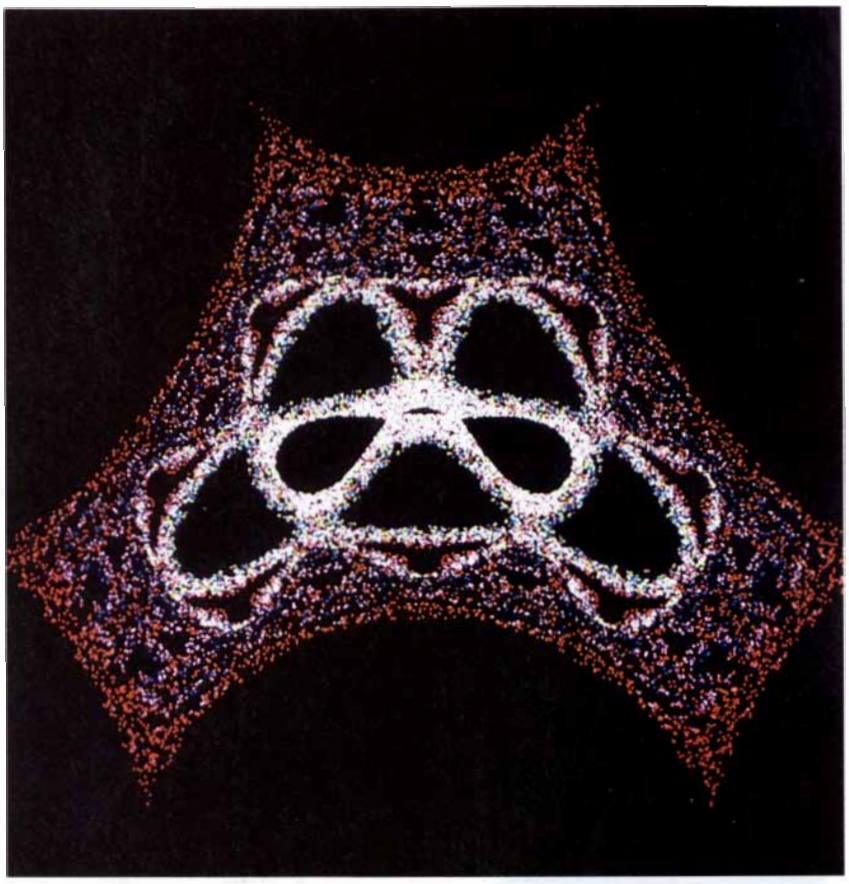
$$x \leftarrow y - \sin(x)$$
$$y \leftarrow a - x$$

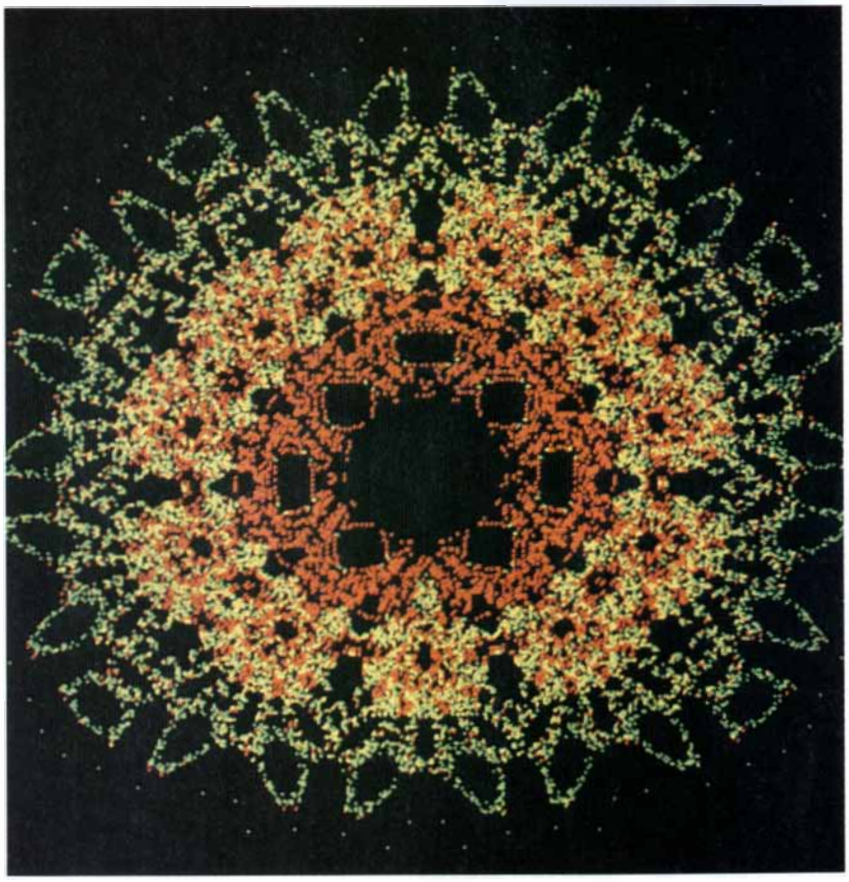
In these formulas only the variable a must be specified. Martin has discovered a most interesting series of patterns when a lies within .07 of the number pi.

The third kind of mental wallpaper must be reserved for rooms devoted to heavy thinking. The patterns range from Persian complexities to Incan riots [*see illustrations at bottom of page 20 and on page 22*]. The methods that lead to them could hardly be less like the techniques described above. Tony D. Smith of PICA Pty. Ltd. in Australia has devised intricate variations of the self-replicating cellular automaton invented in 1960 by Edward Fredkin of the Massachusetts Institute of Technology [see "Computer Recreations," by Brian Hayes; SCIENTIFIC AMERICAN, October, 1983]. There is astonishing potential for pattern in this idea, and Smith has begun to explore an amusing corner in the vast space of possibilities.

What is the Fredkin cellular automaton? Imagine an infinite, two-dimensional grid of square cells. At any given moment each cell is in one of two possible states: living or dead, so to speak. Somewhere an imaginary clock ticks away. The fate of each cell is determined by its four edge-adjacent neighbors: if the number of living neighbors is even for one tick of the clock, the cell will be dead at the next tick regardless of its previous state. On the other hand, if the number of living neighbors is odd, the cell will be alive at the next tick. The same rule is simultaneously applied to every cell on the planar grid.

Fredkin's automaton is closely related to the game of Life, invented by John Horton Conway of the University of Cambridge and often discussed in this department. Fredkin's automaton, however, was discovered earlier than Conway's, and it is much simpler. Moreover, it has an amazing property not shared by Life: any initial configuration of live cells grows through a series of generations (ticks of the clock) into four copies of itself. After several more generations there are 16 copies, 64 copies and so on. The finest wallpaper appears during intermediate gen-





Mandalas on the number 7, generated by Martin's iteration formula

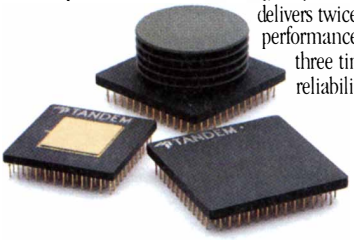
NonStop VLX.™

Tandem technology sets the new standard for large applications in on-line transaction processing.

More transactions per second at a lower cost per transaction than any system in the world.

THE CIRCUITRY'S FAST.

We designed the system in our own laboratory, right down to our own unique VLSI chips. The result is more circuitry in less space. With fewer components than our next largest system, the VLX delivers twice the performance and three times the reliability.



PROCESSORS WITH LARGE APPETITES.

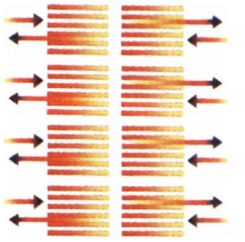
The VLX processors move transactions in 32-bit chunks. They reach into main memory in 64-bit chunks. Because this happens in parallel, more work gets done in less time at a lower cost per transaction.

THE SERVICE IS EASY.

All critical components are field replaceable. When service is required, it's faster. You don't even have to stop an operation to add or replace components.

THE DATA EXPRESSWAY.

In a conventional database, I/O requests must be handled sequentially. This creates queues that slow response time. In the VLX system, there are multiple paths to multiple disks. Data enters and leaves the database simultaneously. No time is wasted, and all disk space gets used.



DIAGNOSTICS FROM A DISTANCE.

An integrated microprocessor allows us to monitor the system environment from anywhere in the world. We can even run stress tests remotely. If a failure does occur, the VLX has the capability to automatically dial out to remote centers anywhere in our worldwide network.

THE SYSTEM KNOWS THE SYMPTOMS.

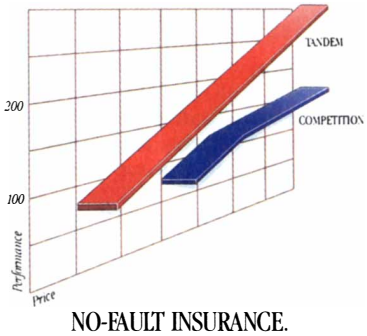
Expert systems software, using fault analysis, directs the problem diagnosis systematically. It also allows us to analyze it and shorten service time even more.

SECRETS ARE SAFE.

We offer software that will protect the security of your data whether it's in the VLX, in another Tandem system or in transmission.

NO GROWING PAINS.

To add power, just add processors. You can grow from a base four-processor system to 16. From there, you can expand in whatever increments you choose, all the way to 255 systems. You never buy more than you need, and you'll never have to rewrite a line of applications code.



NO-FAULT INSURANCE.

Tandem systems achieve fault-tolerance with a unique, parallel processing architecture. There are no idle back-up components. Instead, multiple components share the workload. If one goes down, the others pick up the slack, and application processing is uninterrupted.

HERE TODAY. HERE TOMORROW.

The VLX is compatible with any Tandem system and with all major communications standards — SNA, X.25, MAP and O.S.I. And by acting as a gateway to other vendors' systems, the VLX can link them and enhance their value as well.

WE HAVE EXCELLENT REFERENCES.

Tandem systems are already at work for Fortune 500 companies in banking, telecommunications, manufacturing, transportation, retailing and energy, as well as several branches of the U.S. Government.

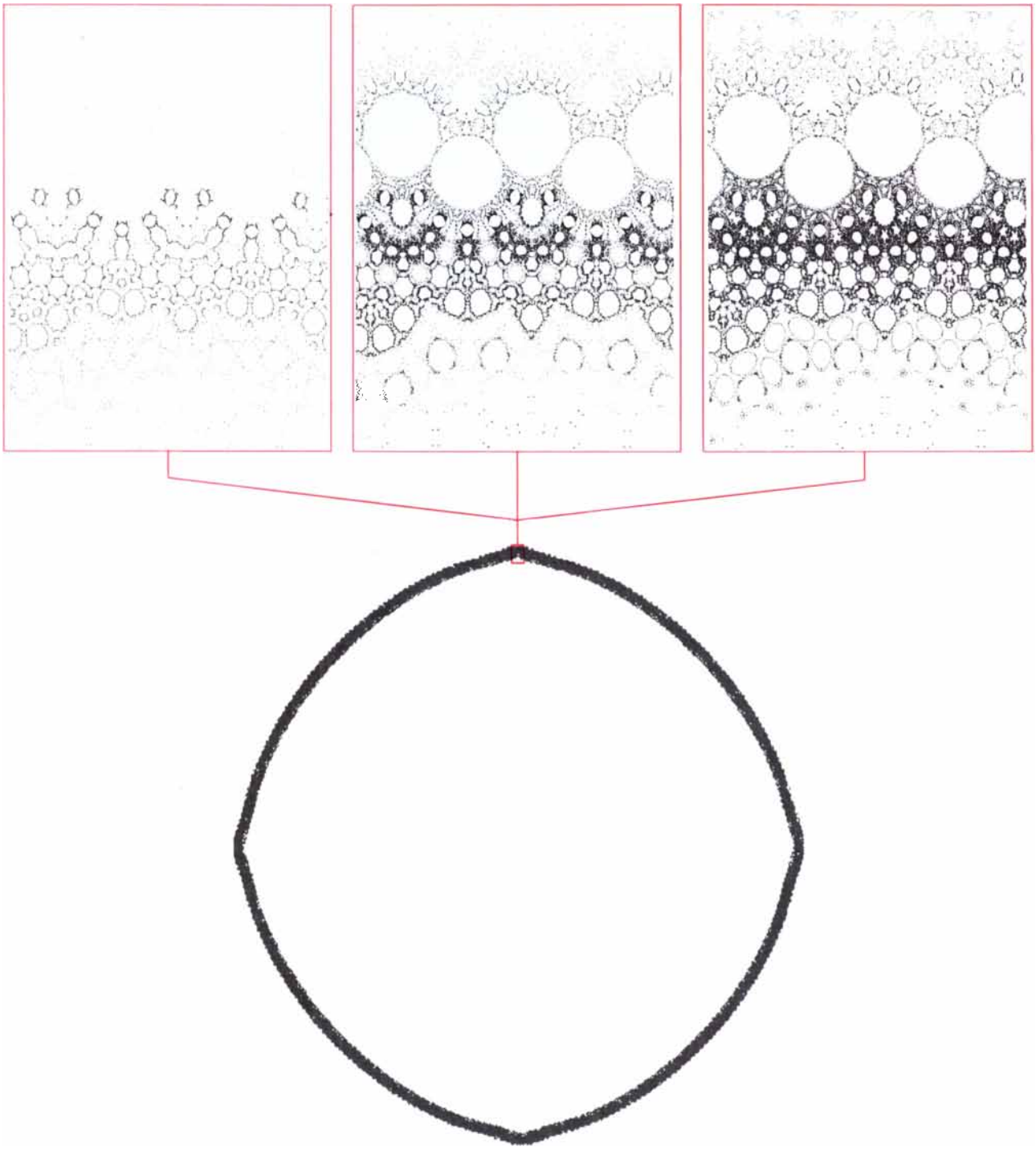
To find out what we can do for you, call 800-482-6336 or write to us. Corporate Headquarters: Tandem Computers, Incorporated, 19191 Vallco Parkway, Loc. 4-31, Cupertino, CA 95014

TANDEMCOMPUTERS

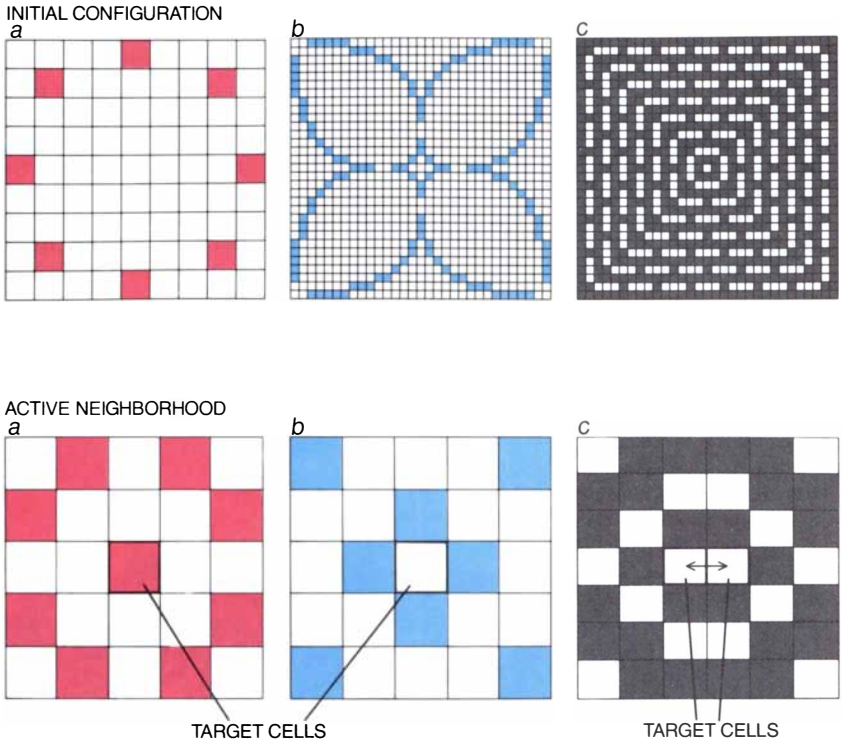
erations: in the generations between the ones in which the original population of live cells is replicated.

Smith's program for printing generalized Fredkin wallpaper is a highly versatile one called PATTERN BREEDER. The rules that determine the fate of a cell in PATTERN BREEDER need not depend solely on the state of the four edge-adjacent cells. In fact, before running the program one can specify the configuration of surrounding cells that will constitute the active neighborhood of each cell. The program then applies the same even-odd rule that was chosen in Fredkin's original automaton. At each tick of the clock if the number of living cells in the active neighborhood is even, the target cell will be dead in the following generation. Otherwise, the cell will be alive.

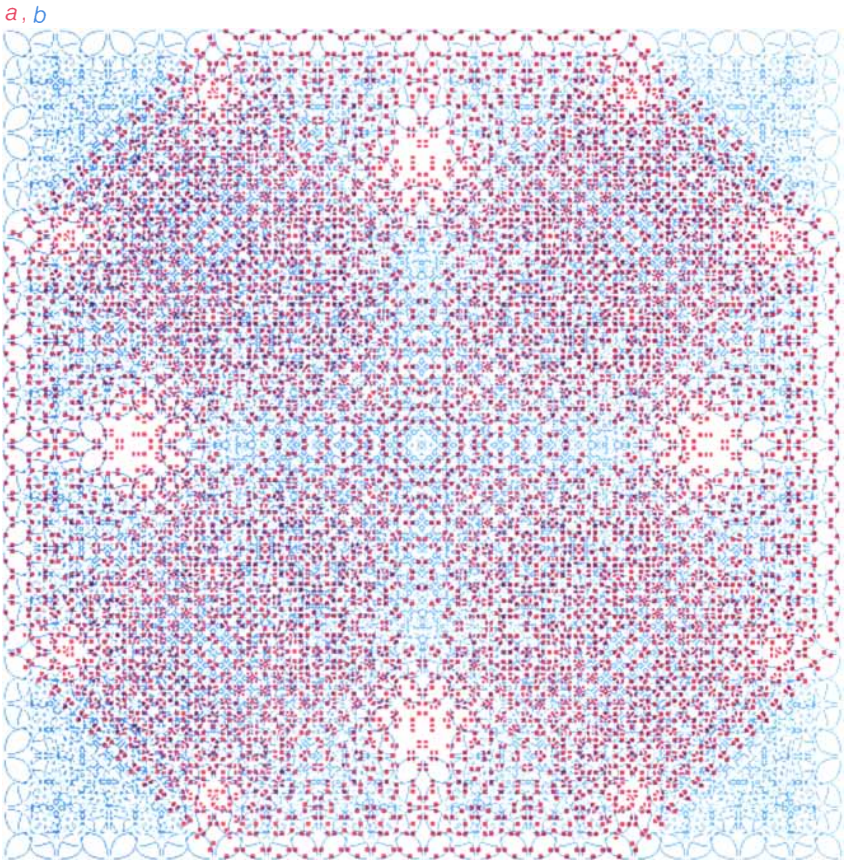
PATTERN BREEDER goes to work on any initial configuration of cells supplied by the user. For example, the initial configuration designated *a* in the top illustration on the next page gives rise to the red part of the pattern shown in the illustration immediately below it: for each target cell and for every stage in the evolution of the pattern the active neighborhood is the same. It is a complex pattern itself, which includes all the colored cells



A model plant stem and its vascular bundles, generated by HOPALONG



Rules for generating cellular automatons in Tony D. Smith's program PATTERN BREEDER



Two images generated by PATTERN BREEDER *and superposed*

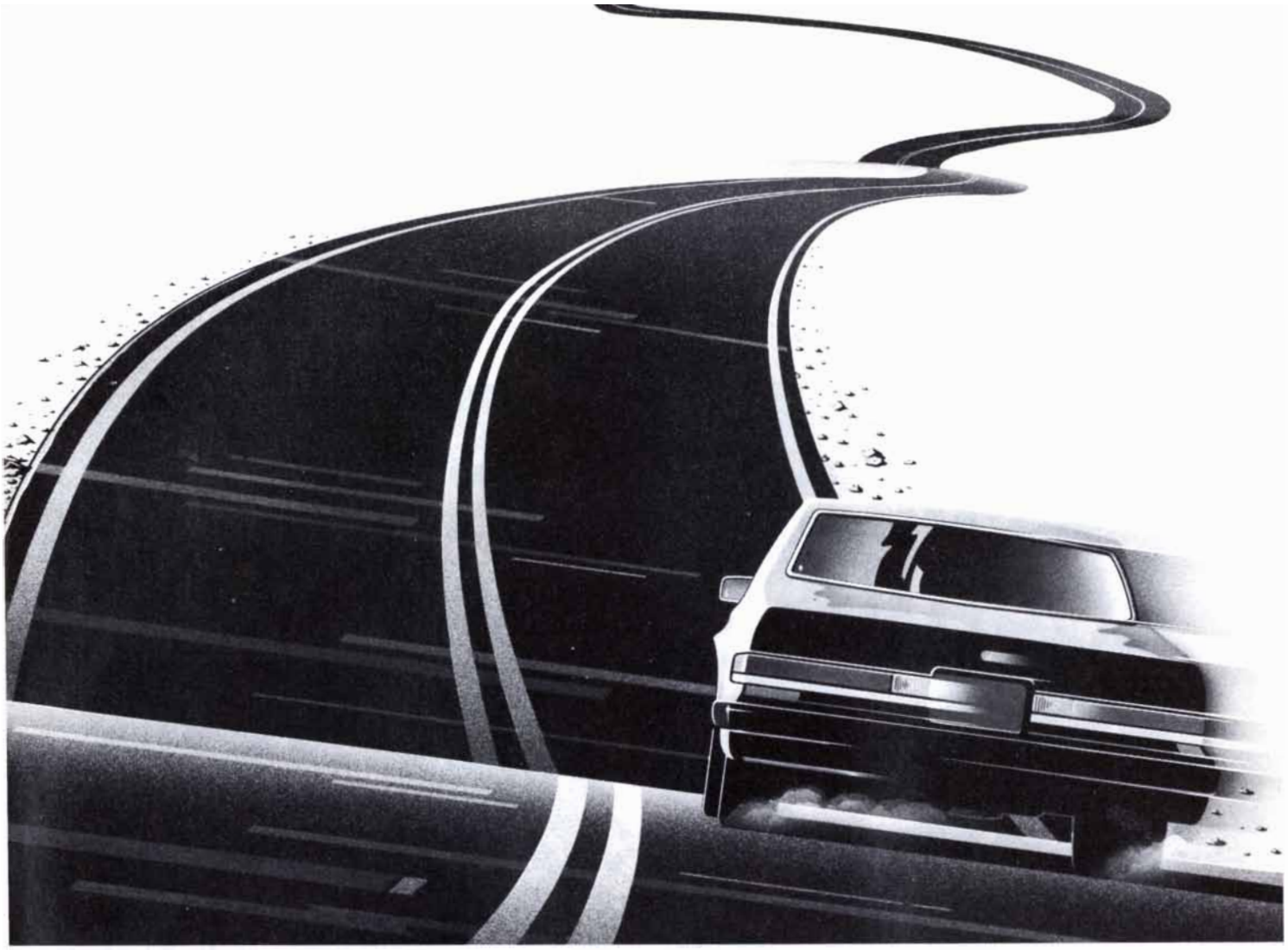
within a 5-by-5 matrix, also designated *a* in the top illustration at the left. Note that the active neighborhood in this case includes the target cell itself. To apply the rule, count the number of live cells that coincide with the cells in the active neighborhood; if the target cell is currently alive, it too is included in the count. Initial configurations and the active neighborhood associated with each of them are also shown for two other images. The ones labeled *b* correspond to the blue part of the pattern in the bottom illustration at the left and the ones labeled *c* correspond to the illustration on page 22. There is an added complexity for active neighborhood *c:* the target cell itself oscillates in succeeding generations from left to right and back again in the center of the neighborhood.

Readers impatient to generate their own magnificent patterns may forgo the algorithmic struggle by writing to Smith at P.O. Box 256, 38 Ardoch Street, Essendon, Victoria 3040, Australia. Smith's program incorporates a library of initial configurations and active neighborhoods from which the user can choose; the program is currently available only for Apple Macintosh computers.

I am not about to outline PATTERN BREEDER in its full sophistication, but I shall describe a simpler program called FREDKIN. Readers with a bit of additional programming acumen can then convert FREDKIN into a more general program with some of the features of PATTERN BREEDER.

```
    input initial pattern
S   for each cell in the screen array
        count ← 0
        for each neighbor of that cell
            if neighbor alive
                then count ← count + 1
        if count even
          then cell ← 0
          else cell ← 1
                plot cell
    input go
    go to S
```

One of the great joys of writing algorithms is that so many levels of description are available. The line between descriptive generality and irresponsible vagueness is a fine one. Readers will note that FREDKIN, as specified above, occupies a slightly more rarefied stratum than the algorithms I outlined previously. For example, the instruction "Input initial pattern" will take several instructions to implement in any practical programming language. Any such instructions would involve a double loop, with two indexes *i* and *j*. Another double loop is concealed in the instruction



All you asked was to go faster and farther. With more power. And less fuel.

In many of the finest American cars, Bosch fuel injection components help deliver precisely the amount of fuel needed for any driving condition.

And those American cars are going faster and farther. Producing up to 20 percent more horsepower. Using less fuel. And reducing exhaust emissions.

Power as you need it.

Bosch multi-point systems inject fuel right at the individual cylinders.

A microcomputer reads thousands of engine operations and driving conditions every second, and dictates the most efficient mix.

So Bosch fuel injection not only gives you more power when you need it. It can boost mileage by up to 15 percent.

Traveling in the best of companies.

Bosch fuel injection arrived in the U.S. in imported cars over 30 years ago. It's since evolved from a curiosity to a high-technology must for today's best performing machines.

Today, every U.S. manufacturer uses Bosch fuel injection systems or components.

Virtually every European model sold in the U.S. is equipped with Bosch fuel injection. And every fuel-injected Japanese model uses systems or components produced under Bosch license or using Bosch patents.

Bosch breaks ground.

Sophisticated problem solving is what Bosch is all about.

We supply every major U.S. and European automaker with an array of advanced products, from high-performance spark plugs to anti-lock braking systems. But our expertise isn't limited to automotive.

We're a leading manufacturer of high-technology products, such as Bosch assembly systems, power tools, packaging machinery, TV studio equipment and video graphics, home appliances, and medical equipment.

In fact, we've pioneered literally thousands of the world's most sophisticated electronic and electro-mechanical advances.

As a result, we're an international corporation with manufacturing facilities in 16 countries. And with sales and service in 137.

But more important are 144,000 Bosch people—9,000 in R&D alone—who are capable of finding the right answer to the toughest challenges.

Including yours.

BOSCH

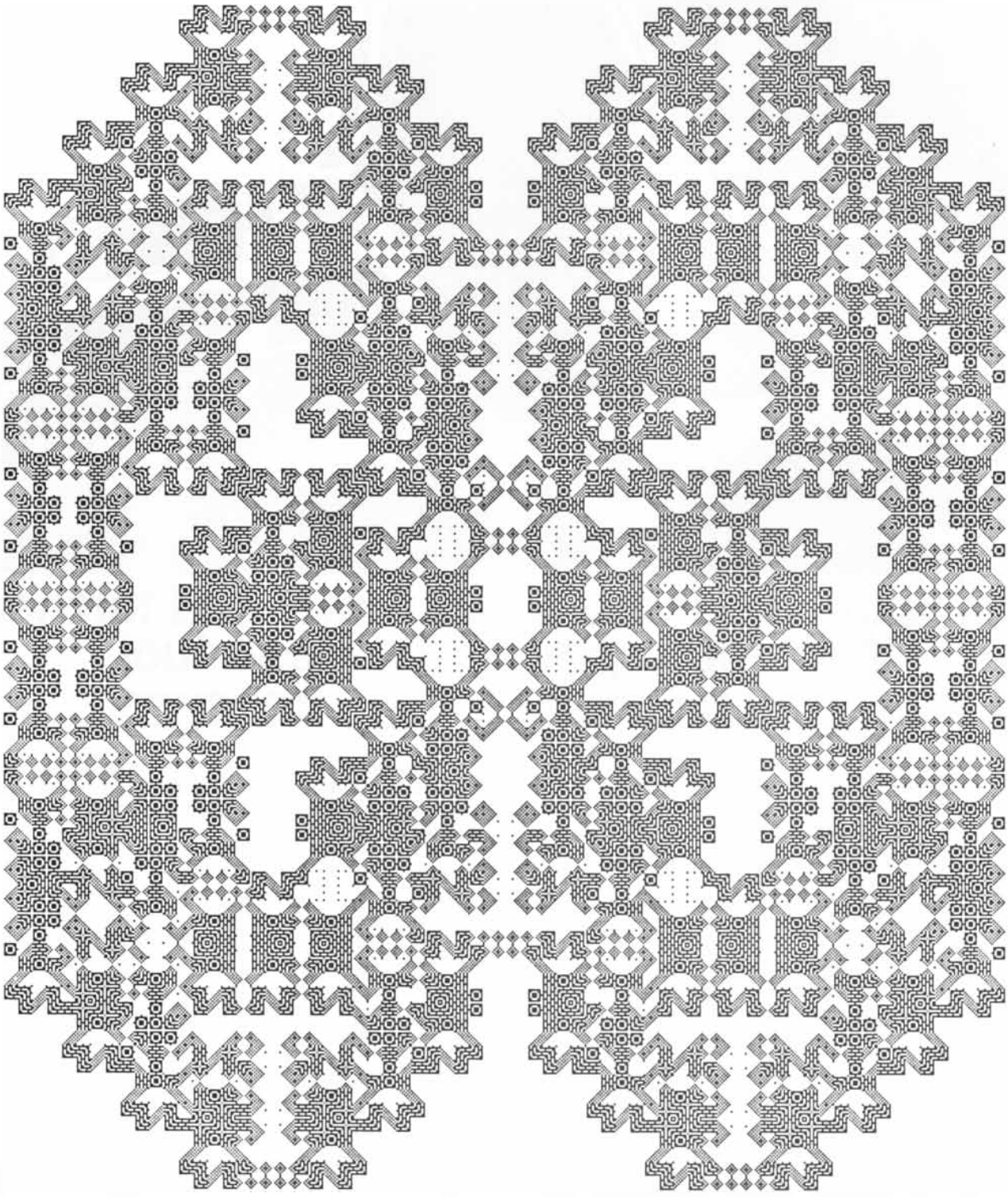
Bringing high technology down to earth.

"For each cell in the screen array." Here the two indexes supply the coordinates of points on the display screen or the printer.

Inside the main program loop FREDKIN simply carries out the rule for the evolution of a given pattern. It counts the number of living neighbors for each cell (i,j); then, if the number is odd, it plots or prints the cell as a single point. The last instruction of the main loop calls for input of the variable *go*. The user may type any number at this stage in order to get FREDKIN to generate the next pattern. In this way the execution of the program can be halted at will if a particularly pleasing pattern appears. Sometimes a bit of trickery and a go-to statement are useful. This strategy is not structured programming, but it works just fine.

I shall be pleased to print, in three months' time, the finest mental wallpaper that readers are able to supply.

c



A Mayan abstraction from Smith's wallpaper program