Pergamon

Comput. & Graphics, Vol. 20, No. 2, pp. 325–332, 1996
Copyright © 1996 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0097–8493/96 $15.00 + 0.00

0097–8493(95)00133–6

Chaos & Graphics

STRANGE ATTRACTOR SYMMETRIC ICONS

J. C. SPROTT

Department of Physics, University of Wisconsin, Madison, WI 53706, U.S.A.
e-mail: sprott@juno.physics.wisc.edu

Abstract—Aesthetically appealing patterns are produced by searching a class of nonlinear maps for chaotic solutions and then transforming the coordinates so that the resulting strange attractors are contained within a sector of a circle, which is then replicated at various angles to produce images that combine the symmetry of the resulting displays with the fractal nature of the underlying attractors. Sample images, computer code in BASIC, and suggestions for enhancing the quality of the images are given. Copyright © 1996 Elsevier Science Ltd

Dynamical systems modeled by nonlinear discrete maps and continuous flows frequently have chaotic solutions characterized by sensitive dependence on initial conditions. The resulting orbits or trajectories in the N-dimensional space defined by the N dynamical variables of the equations usually approach a region of negligible hyper-volume and fractal character. These so-called strange attractors have approximate self-similarity at small scales, which is to say that they look similar at any magnification. As a consequence these attractors often have considerable visual appeal. However, except in very special cases, which may be somewhat common in systems that describe real processes in the natural world [1], the attractors arising from general mathematical equations lack any global symmetries, thereby usually decreasing their aesthetic qualities.

It is possible to construct the equations used to generate the strange attractors so that the solutions have a desired symmetry [2, 3], but this constraint limits the universe of possible equations and complicates the mathematics. This paper describes an alternate approach in which the equations can be chosen arbitrarily, and the symmetry is introduced only when the solution is plotted. The resulting technique is therefore both simple and general.

Figures 1–4 show examples of the patterns that you can produce by this method. Here are the steps that will allow you to produce an unlimited sequence of similar cases:

(1) Choose an arbitrary system of nonlinear dynamical equations with three variables (x, y, and z) and some number of adjustable coefficients a_i. For example, a general, 2-D, quadratic, iterated map would be given by:

$$x_{\text{new}} = a_1 + a_2x + a_3x^2 + a_4xy + a_5y + a_6y^2$$

$$y_{\text{new}} = a_7 + a_8x + a_9x^2 + a_{10}xy + a_{11}y + a_{12}y^2$$

to which we add a third equation such as

$$z_{\text{new}} = x^2 + y^2$$

Three equations are used so that the solution is a sequence of points whose x and y values can be used to determine the horizontal and vertical positions, respectively, and whose z value can represent color and depth information. A fourth equation could be added to allow the color and depth to be independent of one another.

(2) Choose arbitrary values for the coefficients a_i and iterate the equations from an arbitrary starting point, while testing the solution for chaos. One way to do this test is to calculate the Lyapunov exponent, which is a measure of the sensitivity to initial conditions [4]. A positive value indicates chaos, and a negative value indicates that the solution is static or periodic, in which case the attractor is usually not visually interesting; it is a small collection of points, a closed curve, or a torus. Some solutions with a positive Lyapunov exponent are unbounded, and you should ignore these cases using some criterion for unboundedness such as $|x| + |y| + |z| > 10^6$. Continue selecting values of a_i until a chaotic case is found. Alternately, if you have a favorite chaotic system, you can enter its equation directly and skip the first two steps. Some chaotic maps have strange attractors whose dimensions are too low to be visually interesting. These cases can be eliminated by calculating their fractal dimension in real time [5] or, more simply, by placing a lower limit on the number of screen pixels that must be illuminated.

(3) Iterate the equation for about a hundred steps to be sure the solution has reached the attractor, and then iterate for another thousand or so steps to be sure the Lyapunov exponent is reasonably accurate and to determine the minimum and maximum values of x, y, and z so that the plot boundaries can be chosen appropriately.

(4) Choose a color palette, a background color, and a shadow color to enhance the illusion of depth. For example, if your display is capable of 256 colors, and each primary color can have one of 64 values,

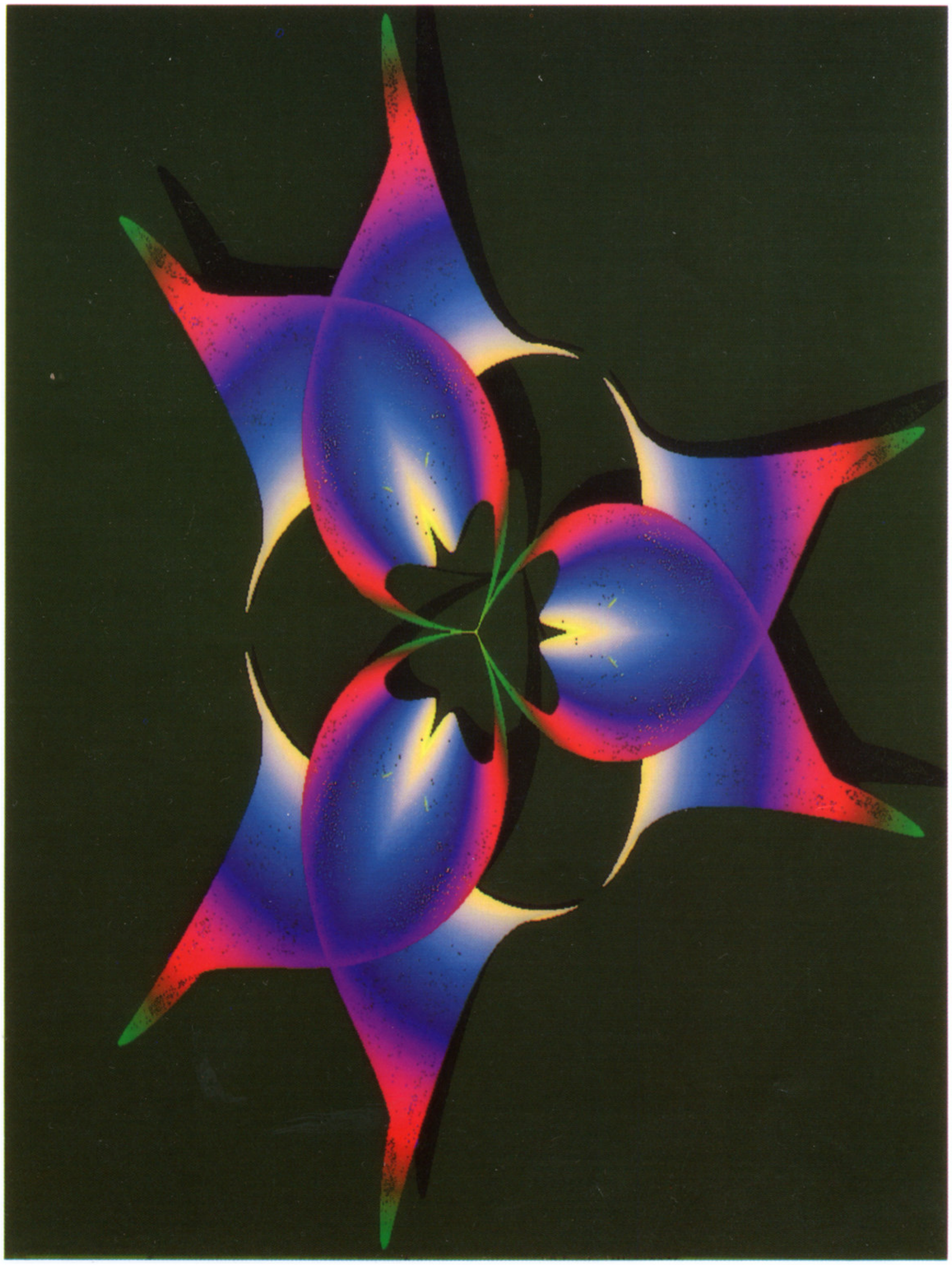

Fig. 1. Quadratic map with the coding[6] ELBIAJSHWHARL and six sectors with reflection symmetry.

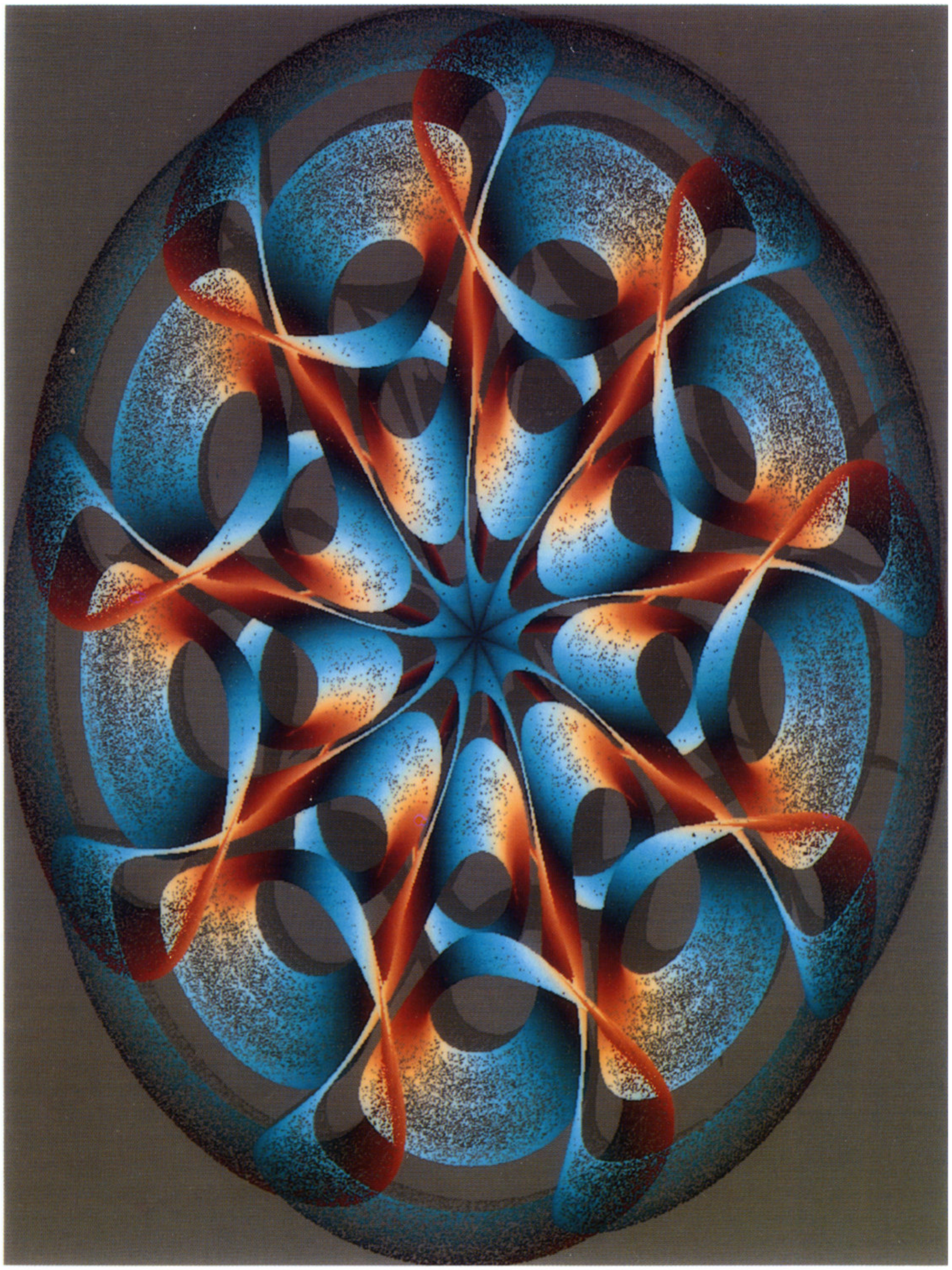

Fig. 2. Cubic flow from the Duffing Equation[7] with the coding[6] ^DHXNCOEUK and nine sectors.

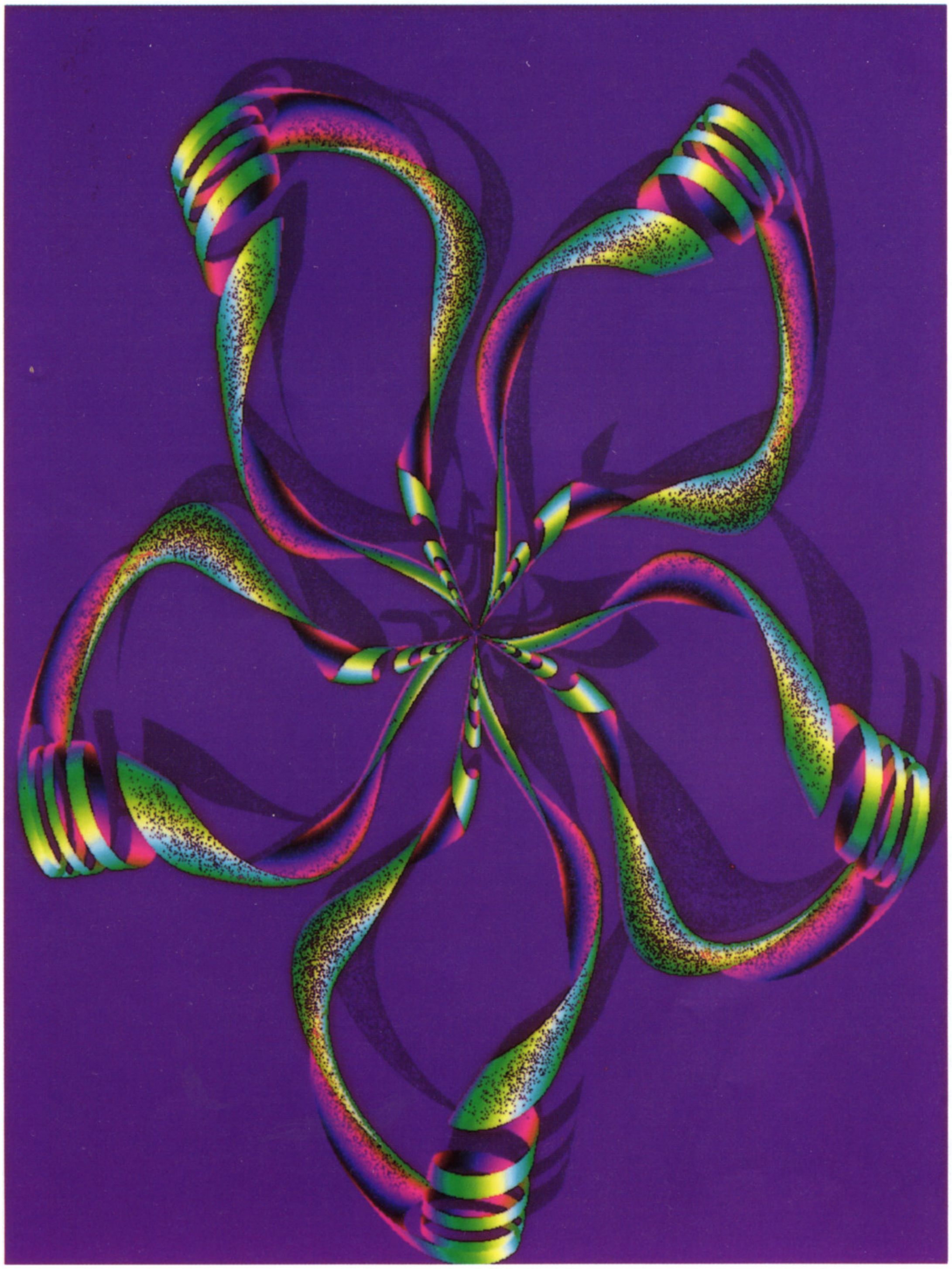

Fig. 3. Cubic flow from the Duffing Equation[7] with the coding[6] ^FMMETWLDV and five sectors.

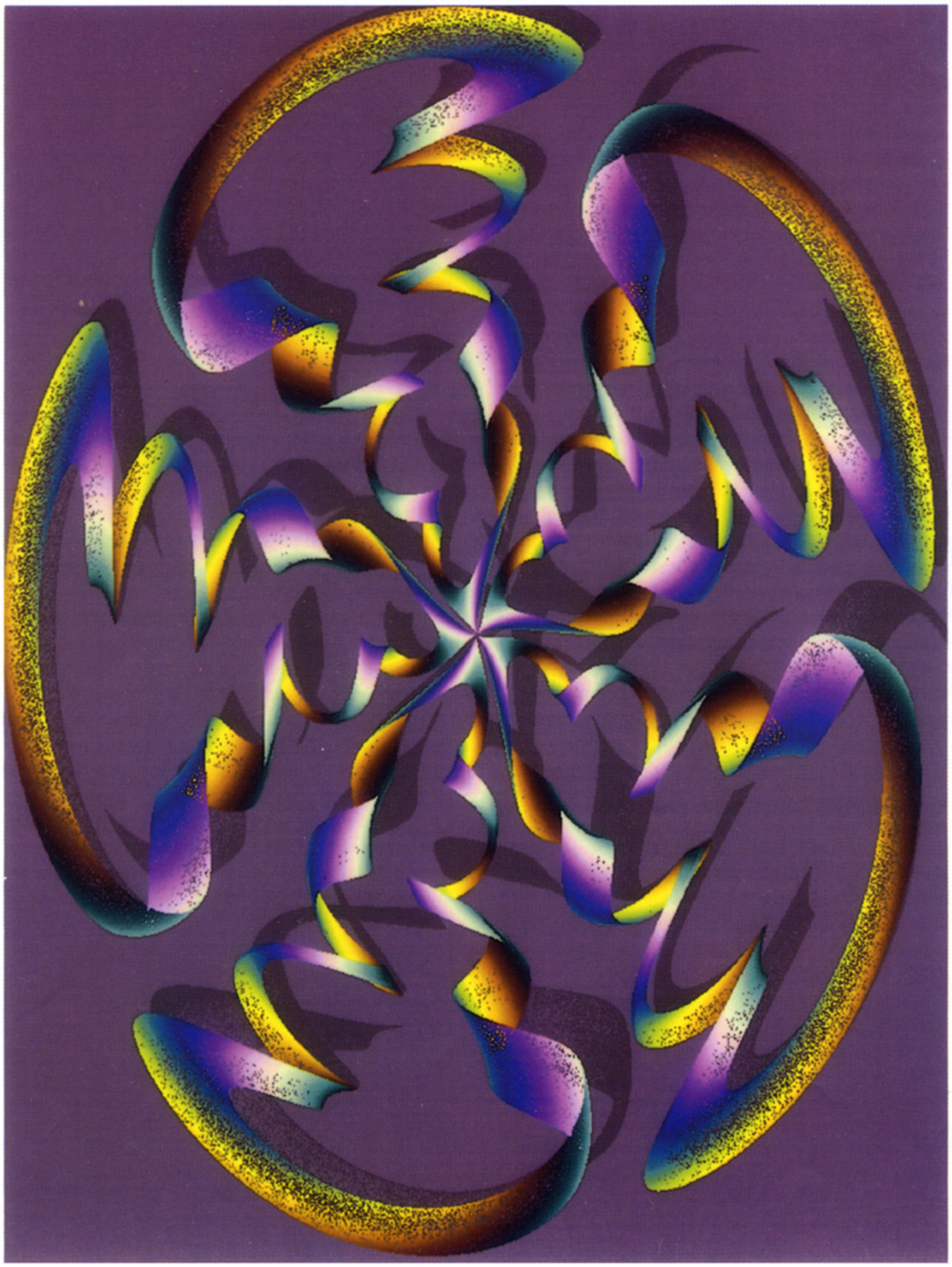

Fig. 4. Cubic flow from the Duffing Equation[7] with the coding[6] ^IDUJKYEYV and five sectors.

you might allow the red, green, and blue components of the color to vary sinusoidally so that the components of the ith color are given by

$$R_i = 32 + 32\ \sin[2\pi C_r(i/254 + \phi_r)]$$

$$G_i = 32 + 32\ \sin[2\pi C_g(i/254 + \phi_g)]$$

$$B_i = 32 + 32\ \sin[2\pi C_b(i/254 + \phi_b)]$$

where the values of C are small integers (typically 0 to 3) determining the number of cycles of each color component, and the values of ϕ are phases in the range 0 to 1, chosen randomly or by the user. The background color can be chosen arbitrarily, but a more systematic method is to use a half-intensity version of one of the palette colors, chosen either randomly or by some criterion such as the predominant color (determined by counting pixels of each color) or perhaps the color corresponding to the lowest or highest index i, which will correspond to the most distant or the closest point on the attractor, respectively. The shadow can be either black or a low (such as 1/3) intensity version of the background color.

(5) Iterate the equations some number of times. The number can be fixed (a value of 2×10^6 was used in Figs 1–4), or it can be determined by some criterion such as the probability of a point falling on a previously illuminated screen pixel reaching 50%. The appearance of the image is not strongly sensitive to this choice, but using too few iterations lowers the contrast, while using too many destroys some of the fine-scale detail and reduces the apparent color intensity variation.

(6) Convert each (x, y) iterate to polar coordinates (r, θ) using

$$r = (x - x_{\min})/(x_{\max} - x_{\min})$$

$$\theta = 2\pi[f(y - y_{\min})/(y_{\max} - y_{\min}) + S]/N_s$$

where f is an overlap factor, (typically chosen in the range of 0.5 to 2), S is the sector number (an integer in the range of 1 to N_s chosen randomly at each iteration), and N_s is the number of sectors (typically 2 to 9). Each iterate is plotted on the screen at position (x_p, y_p) in color c, given by

$$x_p = 0.5w(1 + r\ \sin\ \theta)$$

$$y_p = 0.5h(1 + r\ \cos\ \theta)$$

$$c = 254\ (z - z_{\min})/(z_{\max} - z_{\min})$$

where w is the screen width and h is the screen height in pixels. Note that the roles of sine and cosine have been reversed from the usual mathematical convention ($\theta = 0$ is downward) to enhance right–left symmetry, which is more common in nature than up–down symmetry and hence more visually appealing. (Consider, for example, a front view of an upright human.) Since the screen width-to-height ratio (w/h) is typically 4/3, the resulting pattern is not circular, but slightly elliptical. Note also that it would have been possible and reasonable to plot each iterate in the appropriate place in all the sectors, but selecting a sector randomly to plot each time gives the sectors minute but aesthetically desirable differences, making the patterns look slightly more natural and disguising their machine-generated origin. An additional embellishment, adding to the illusion of depth, is to plot only those points whose color index c is greater than the color index already associated with the pixel, to occlude those portions of the object that lie behind other portions. In the patterns with an even number of sectors, the sense of θ has been reversed in alternate sectors (θ is replaced by $2\pi/N_s - \theta$ if N_s and S are even), producing a reflection symmetry about certain radials.

(7) Finally, shadow points are added below and to the right of each point in the object as if the object were illuminated by a light emanating from above your left shoulder as is the usual convention. The displacement of the shadow point in each direction is given by

$$\delta = 0.05w(z - z_{\min})/(z_{\max} - z_{\min})$$

where the factor 0.05 is the tangent of the illumination angle. You might want to experiment with different values of this parameter. The shadow points are plotted only if they fall on the background. It is more complicated to produce a shadow of one part of the attractor on another, and this improvement will be left to your ingenuity.

A fully operational computer program that carries out the steps outlined above is given in the Appendix.[†] The program should run without modification under Microsoft QBASIC, QuickBASIC, and VisualBASIC for MS-DOS. It assumes VGA (320×200-pixel, 256-color) graphics. Figures 1–4 were produced with an enhancement of this program that differs only in that it uses a screen resolution of 1024×768[‡] and the equations describing the dynamical system were selected from a larger collection whose coefficients are coded into the names given in the figure captions [6]. Figure 1 is an example of a quadratic map with six sectors and reflection symmetry. Figures 2–4 are from a system of ordinary differential equations given by

$$\begin{aligned} dx/dt &= a_1 y \\ dy/dt &= a_2 x + a_3 x^3 + a_4 x^2 y + a_5 x y^2 + a_6 y \\ &\quad + a_7 y^3 + a_8 \sin z \\ dz/dt &= a_9 + 1.3 \end{aligned}$$

This system is a generalization of the Duffing

[†] Also available on the World Wide Web at http://sprott.physics.wisc.edu/software.htm.

[‡] High-resolution graphics drivers for QuickBASIC and other languages are available from Zephyr Software, P.O. Box 7704, Austin, TX 78713-7704, U.S.A.

equation, which models a mass on a nonlinear spring or a two-well oscillator [7].

The cases shown in the figures were selected from a collection of about a thousand similar cases in a two-step process. In the first step, a visually interesting system was selected from many cases with different equations or coefficients. In the second step, a few hundred variations were produced by randomly changing the palette, number of sectors, and overlap factor. Each case requires a few minutes to generate at the highest resolution. The program was usually run overnight to collect a few hundred cases for rapid inspection in the morning. The cases shown are typical, and almost all the cases found by the method are aesthetically pleasing.

Acknowledgements—I am grateful to Robin Chapman for useful suggestions and to Cliff Pickover for printing the figures.

REFERENCES

1. I. Stewart, *Does God Play Dice?: The Mathematics of Chaos*, Blackwell, Cambridge, MA (1989).
2. I. Stewart and M. Golubitsky, *Fearful Symmetry: Is God a Geometer?*, Blackwell, Cambridge, MA (1992).
3. M. Field and M. Golubitsky, *Symmetry in Chaos*, Oxford University Press, New York (1992).
4. J. C. Sprott, Automatic generation of strange attractors. *Computers & Graphics* **17**, 325–332 (1993).
5. J. C. Sprott, Automatic generation of iterated function systems. *Computers & Graphics* **18**, 417–425 (1994).
6. J. C. Sprott, *Strange Attractors: Creating Patterns in Chaos*, M&T Books, New York (1993).
7. J. M. T. Thompson and H. B. Stewart, *Nonlinear Dynamics and Chaos*, Wiley, New York (1986).

APPENDIX

```
DEFDBL A-Z                                'Use double precision
DIM A(12), PAL&(255)
NMAX = 61000                              'Maximum number of iterations
TWOPI = 8 * ATN(1)                        'A useful constant (2 pi)
RANDOMIZE TIMER                           'Reseed random number generator
SCREEN 13                                 'Set graphics mode
SW% = 320                                 'Screen width
SH% = 200                                 'Screen height
NC% = 254                                 'Number of colors
WHILE T% = 0
    GOSUB Init                            'Initialize
    WHILE T% = 1
        GOSUB Iteqn                       'Iterate equations
        GOSUB Display                     'Display results
        GOSUB Test                        'Test results
    WEND
WEND
PALETTE: CLS                              'Restore the palette colors
END

Init:                                     'Initialize
X = .05: Y = .05: Z = .05                 'Initial condition
XE = X + .000001: YE = Y: ZE = Z
T% = 1: LSUM = 0: N = 0: NL = 0
XL = 1000000!: XH = -XL: YL = XL: YH = XH: ZL = XL: ZH = XH
NS% = 2 + INT(8 * RND)                    'Number of sectors (2 to 9)
OF = .5 + 1.5 * RND                       'Overlap factor (.5 to 2)
FOR I% = 1 TO 12                          'Get random coefficients
    A(I%) = (INT(25 * RND) - 12) / 10
NEXT I%
RETURN

Iteqn:                                    'Iterate equations
XNEW = A(1) + X * (A(2) + A(3) * X + A(4) * Y) + Y * (A(5) + A(6) * Y)
YNEW = A(7) + X * (A(8) + A(9) * X + A(10) * Y) + Y * (A(11) + A(12) * Y)
ZNEW = X * X + Y * Y
N = N + 1
RETURN

Display:                                  'Display results
IF N > 100 AND N < 1000 THEN              'Get scale limits for graph
    IF X < XL THEN XL = X ELSE IF X > XH THEN XH = X
    IF Y < YL THEN YL = Y ELSE IF Y > YH THEN YH = Y
    IF Z < ZL THEN ZL = Z ELSE IF Z > ZH THEN ZH = Z
END IF
```

```
IF N = 1000 THEN                        'Set palette and clear screen
    RANB = RND: CB% = 1 + INT(3 * RND)  'Blue phase and period
    RANG = RND: CG% = 1 + INT(3 * RND)  'Green phase and period
    RANR = RND: CR% = 1 + INT(3 * RND)  'Red phase and period
    BC% = INT(2 + NC% * RND)            'Choose random background color
    FOR I% = 2 TO NC% + 1               'Redefine palette colors
        B% = INT(32 + 32 * SIN(CB% * TWOPI * (I% / NC% + RANB)))
        G% = INT(32 + 32 * SIN(CG% * TWOPI * (I% / NC% + RANG)))
        R% = INT(32 + 32 * SIN(CR% * TWOPI * (I% / NC% + RANR)))
        PAL&(I%) = 65536 * B% + 256 * G% + R%
        IF I% = BC% THEN                'Set background and shadow colors
            PAL&(0) = 65536 * INT(B% / 2) + 256 * INT(G% / 2) + INT(R% / 2)
            PAL&(1) = 65536 * INT(B% / 3) + 256 * INT(G% / 3) + INT(R% / 3)
        END IF
    NEXT I%
    CLS : PALETTE USING PAL&(0)
    XZ = .05 * SW% / (ZH - ZL)
    YZ = .05 * SH% / (ZH - ZL)
END IF
IF N > 1000 THEN                        'Plot point on screen
    IF XH > XL THEN XP = SW% * (X - XL) / (XH - XL)
    IF YH > YL THEN YP = SH% * (YH - Y) / (YH - YL)
    S% = INT(NS% * RND)                 'Choose sector randomly
    TH = TWOPI * (OF * YP / SH% + S%) / NS%
    IF (NS% MOD 2) = 0 AND (S% MOD 2) = 0 THEN TH = TWOPI / NS% - TH
    YP = .5 * SH% * (1 + XP * COS(TH) / SW%)
    XP = .5 * SW% * (1 + XP * SIN(TH) / SW%)
    C% = 2 + INT(NC% * (Z - ZL) / (ZH - ZL) + NC%) MOD NC%
    IF C% > POINT(XP, YP) THEN PSET (XP, YP), C%
    XP = XP + XZ * (Z - ZL): YP = YP + YZ * (Z - ZL)
    IF POINT(XP, YP) = 0 THEN PSET (XP, YP), 1
END IF
RETURN

Test:                                   'Test results
IF ABS(XNEW) + ABS(YNEW) + ABS(ZNEW) > 1000000! THEN T% = 0 'Unbounded
XSAVE = XNEW: YSAVE = YNEW: ZSAVE = ZNEW
X = XE: Y = YE: Z = ZE: N = N - 1
GOSUB Iteqn                             'Reiterate equations
DLX = XNEW - XSAVE: DLY = YNEW - YSAVE: DLZ = ZNEW - ZSAVE
DL2 = DLX * DLX + DLY * DLY + DLZ * DLZ
IF CSNG(DL2) > 0 THEN                   'Don't divide by zero
    DF = 1000000000000# * DL2
    RS = 1 / SQR(DF)
    XE = XSAVE + RS * (XNEW - XSAVE): XNEW = XSAVE
    YE = YSAVE + RS * (YNEW - YSAVE): YNEW = YSAVE
    ZE = ZSAVE + RS * (ZNEW - ZSAVE): ZNEW = ZSAVE
    LSUM = LSUM + LOG(DF): NL = NL + 1
    L = .721347 * LSUM / NL             'This is the Lyapunov exponent
END IF
IF N > 100 AND L < .005 THEN T% = 0     'Not chaotic
IF N > NMAX THEN T% = 0                 'Strange attractor found
IF LEN(INKEY$) THEN T% = 2              'Exit on keypress
X = XNEW: Y = YNEW: Z = ZNEW            'Update value of variables
RETURN
```